Memory Access Partition on 8-Bit PIC® Microcontrollers

Author: *Mary Tamar Tan*
Microchip Technology Inc.

INTRODUCTION

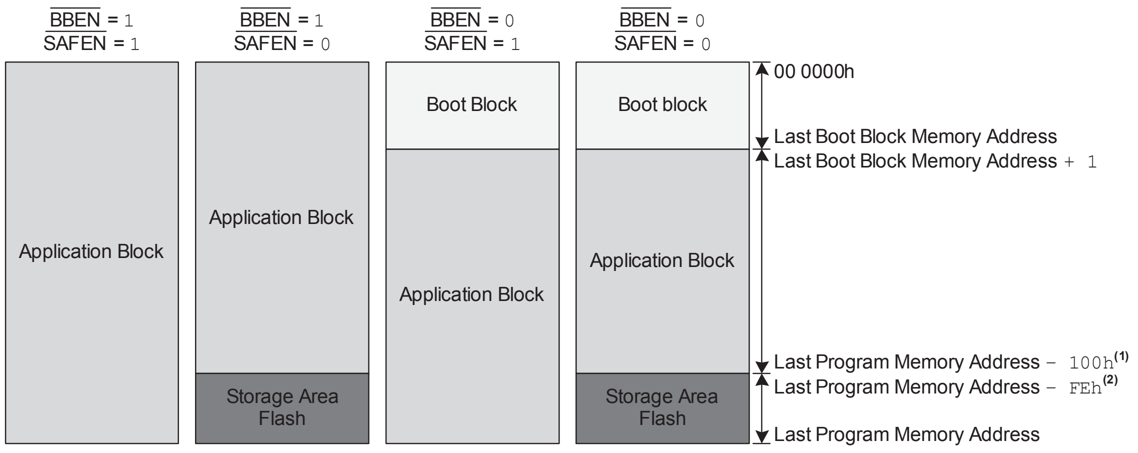
The Program Flash Memory (PFM) is a nonvolatile memory that can store executable code. In addition to instructions, it can also be used for data storage. The PFM size of 8-bit PIC® microcontrollers can span up to 128 K words, depending upon the selected device.

Applications, such as bootloaders, require PFM partitioning to provide isolation between the bootloader and the application code. To meet this requirement, several PIC MCUs have a dedicated boot block with a fixed size and address range. Fixed boot block sizes, however, can leave unused memory wasted. For increased flexibility in boot block size allocation, several PIC MCUs feature the Memory Access Partition (MAP). It allows the PFM to be partitioned in up to three blocks, namely the Application Block, the Boot Block and the Storage Area Flash (SAF) Block.

Another important requirement of a bootloader is that it should be able to protect itself from accidental overwrites. In other words, no part of the bootloader code should be modified either during the bootload process or during application run time. To address this requirement, PIC MCUs with MAP allow selected partition(s) to be write-protected. Each partition has a dedicated write protection Configuration bit to prohibit self-write and erase to that specific partition. MAP-invoked write protection is limited to self-write and erase, and does not affect a device programmer's ability to read, write or erase the device.

This Technical Brief provides information about the three PFM partitions. A sample bootloader implementation is included to help readers better understand how memory partitioning and write protection are done in devices with MAP, and its significance in such applications. Brief information on how to configure a project with MPLAB® X IDE and XC8 C Compiler to reflect the desired partition configuration on a PIC18 device is also provided.

FIGURE 1: PFM PARTITION


$\overline{\text{BBEN}}$ = 1, $\overline{\text{SAFEN}}$ = 1
$\overline{\text{BBEN}}$ = 1, $\overline{\text{SAFEN}}$ = 0
$\overline{\text{BBEN}}$ = 0, $\overline{\text{SAFEN}}$ = 1
$\overline{\text{BBEN}}$ = 0, $\overline{\text{SAFEN}}$ = 0
Application Block
Storage Area Flash
Boot Block
Boot block
00 0000h
Last Boot Block Memory Address
Last Boot Block Memory Address + 1
Last Program Memory Address – 100h(1)
Last Program Memory Address – FEh(2)
Last Program Memory Address

NOTE 1: `100h` for PIC18; `80h` for PIC16.
2: `FEh` for PIC18; `7Fh` for PIC16.

PROGRAM FLASH MEMORY PARTITION

The Program Flash Memory (PFM) can be partitioned into the Application Block, Boot Block and Storage Area Flash (SAF) Block. Figure 1 shows the four possible block combinations. The selected configuration depends on the settings of the $\overline{\text{BBEN}}$ and $\overline{\text{SAFEN}}$ Configuration bits.

Application Block

The Application Block is where the user's program resides. The default settings of the Configuration bits ($\overline{\text{BBEN}}$ = `1` and $\overline{\text{SAFEN}}$ = `1`) assign all memory in the PFM to the Application Block. To enable the write-protect of the application block, the $\overline{\text{WRTAPP}}$ Configuration bit must be cleared to '`0`'.

Boot Block

The Boot Block is an area in the PFM which is ideal for storing the bootloader code. The Boot Block code is executable by the CPU. The Boot Block size is configured through the BBSIZE<2:0> Configuration bits. The maximum boot block size is half the size of the available program memory. For example, all settings of BBSIZE<2:0> = `000` through BBSIZE<2:0> = `100`, provide a Boot Block size of 4 kW on an 8 kW device. Table 1 and Table 2 show the Boot Block size bits settings for different PIC18 and PIC16 device sizes. Refer to the data sheet for the device size.

The Boot Block is enabled by setting $\overline{\text{BBEN}}$ to '`1`'.

Devices with MAP allow more flexibility in terms of Boot Block memory allocation compared to devices without MAP, wherein the Boot Block size is fixed or there is no allocated Boot Block at all. With MAP, there will be less wasted memory for small bootloader code and more memory (up to half the PFM size) for large bootloader code.

The Boot Block can also be write-protected to avoid accidental self-writes to the bootloader. The $\overline{\text{WRTB}}$ Configuration bit is cleared to '`0`' to enable the write protection of the Boot Block.

TABLE 1: PIC18 BOOT BLOCK SIZE BITS

$\overline{\text{BBEN}}$	BB SIZE<2:0>	Device Size (Words)							
		8K		16K		32K		64K	
		BB Size (Words)	BB End Address	BB Size (Words)	BB End Address	BB Size (Words)	BB End Address	BB Size (Words)	BB End Address
0	111	512	00 03FFh	512	00 03FFh	512	00 03FFh	512	00 03FFh
0	110	1024	00 07FFh	1024	00 07FFh	1024	00 07FFh	1024	00 07FFh
0	101	2048	00 0FFFh	2048	00 0FFFh	2048	00 0FFFh	2048	00 0FFFh
0	100	4096	00 1FFFh	4096	00 1FFFh	4096	00 1FFFh	4096	00 1FFFh
0	011	4096	00 1FFFh	8192	00 3FFFh	8192	00 3FFFh	8192	00 3FFFh
0	010	4096	00 1FFFh	8192	00 3FFFh	16384	00 7FFFh	16384	00 7FFFh
0	001	4096	00 1FFFh	8192	00 3FFFh	16384	00 7FFFh	32768	00 FFFFh
0	000	4096	00 1FFFh	8192	00 3FFFh	16384	00 7FFFh	32768	00 FFFFh

TABLE 2: PIC16 BOOT BLOCK SIZE BITS

$\overline{\text{BBEN}}$	BB SIZE<2:0>	Device Size (Words)					
		8K		16K		32K	
		BB Size (Words)	BB End Address	BB Size (Words)	BB End Address	BB Size (Words)	BB End Address
0	111	512	01FFh	512	01FFh	512	01FFh
0	110	1024	03FFh	1024	03FFh	1024	03FFh
0	101	2048	07FFh	2048	07FFh	2048	07FFh
0	100	4096	0FFFh	4096	0FFFh	4096	0FFFh
0	011	4096	0FFFh	8192	1FFFh	8192	1FFFh
0	010	4096	0FFFh	8192	1FFFh	16384	3FFFh
0	001	4096	0FFFh	8192	1FFFh	16384	3FFFh
0	000	4096	0FFFh	8192	1FFFh	16384	3FFFh

Storage Area Flash

The Storage Area Flash (SAF) Block is used for data storage. When enabled through the $\overline{\text{SAFEN}}$ Configuration bit, the SAF spans 128 words at the end of the PFM.

Table 3 shows the start and end addresses of SAF for different PIC18 and PIC16 device sizes with $\overline{\text{SAFEN}}$ = 0. When SAF is disabled ($\overline{\text{SAFEN}}$ = 1), the given address range becomes part of the Application Block.

TABLE 3: SAF RANGES WITH $\overline{\text{SAFEN}}$ = 0

Device Size (Words)	SAF Size (Words)	PIC18		PIC16	
		Start Address	End Address	Start Address	End Address
8K	128	00 3F00h	00 3FFFh	3F80h	3FFFh
16K	128	00 7F00h	00 7FFFh	7F80h	7FFFh
32K	128	00 FF00h	00 FFFFh	FF80h	FFFFh
64K	128	01 FF00h	01 FFFFh	—	—

In C, a variable will be deliberately inserted into Flash if:

- It is either file or static scope, and
- It is qualified as const; or
- It is a string initializer

Example 1 shows the three possible scenarios.

EXAMPLE 1: PLACING VARIABLES IN PFM

```
const   int flash1 = 12;                            // file scope + const ==> in flash
void    myFunction(void)
{
    static const int flash2 = 34;                   // static scope + const ==> flash
    char text[] = "This text is in flash";          // but text[] itself is in RAM

    ...some code....
}
```

The const qualified variables can be explicitly placed into SAF through absolute addressing. In Example 2, a struct saf_t is composed of three integer members and an array of integers with 125 elements. For PIC18, an integer is equivalent to 16 bits or 1 word. This makes the struct size equal to 256 bytes or 128 words corresponding to the size of SAF. In this example, it is assumed that the device size is 16KW. A saf_t variable saf is const qualified and placed at an absolute address of 0x7F00. This places the constants, 0x12, 0x34 and 0x56, to PFM addresses 0x7F00, 0x7F02 and 0x7F04, respectively. The use of structures is recommended because the layout of the data will be unchanged across program revisions, and a single header file can give the bootloader and the application the same layout.

Note: Example 2 is only applicable to PIC18 devices. For PIC16 devices, wherein a word is equivalent to 14 bits, the compiler assigns only const_bytes_. The const integer becomes a 2-word object, where only the low bytes are used.

EXAMPLE 2: PLACING DATA STRUCTURES IN SAF

```
struct   saf_t
{
    int  flash1;
    int  flash2;
    int  flash3;
    int  flashArray[125];     // pad to 128 words
};

volatile const struct saf_t saf @ 0x7F00 =
{
    0x12,
    0x34,
    0x56,
};
```

The XC8 compiler always treats const variables as read-only and any attempt to write to these will result in an error during compilation. Placing a const variable in SAF instead of inside the Application Block ensures that it is stored as data and not as an instruction. This is because the SAF Block is not available for program execution.

If SAF is enabled, the SAF block becomes an invalid execution area. Executing an instruction being fetched from outside the valid execution area generates a memory execution violation. When a memory execution violation is generated, the $\overline{\text{MEMV}}$ flag bit in the PCON1 register is cleared to signal a Reset.

The SAF Block can be write-protected by setting the $\overline{\text{WRTSAF}}$ Configuration bit to '0'. All data stored in SAF becomes read-only after enabling write protection.

Using the PFM as data storage has a drawback in terms of access time. Accessing data located in the program memory is much slower than accessing objects in the data memory. For example, data in a PIC18 program memory is read one byte at a time using TBLRD instructions, thus requiring longer access time.

BOOTLOADER IMPLEMENTATION

A bootloader is firmware that resides within the memory to provide self-programming capability to the microcontroller. One requirement of a bootloader is that it should be protected from accidental overwrites. Write-protect can be implemented either in hardware or in software. Using hardware write-protect has the advantage of a smaller code footprint compared to software write-protect. Unlike devices with a fixed Boot Block, devices with MAP allow the user to select a Boot Block size where the bootloader can fit with the least possible wasted memory.

Microchip provides several bootloader tools for PIC MCUs, such as the MPLAB Code Configurator (MCC) Bootloader Generator to easily configure and generate a bootloader, and the Unified Bootloader Host Application for loading the bootloadable application code into the device.

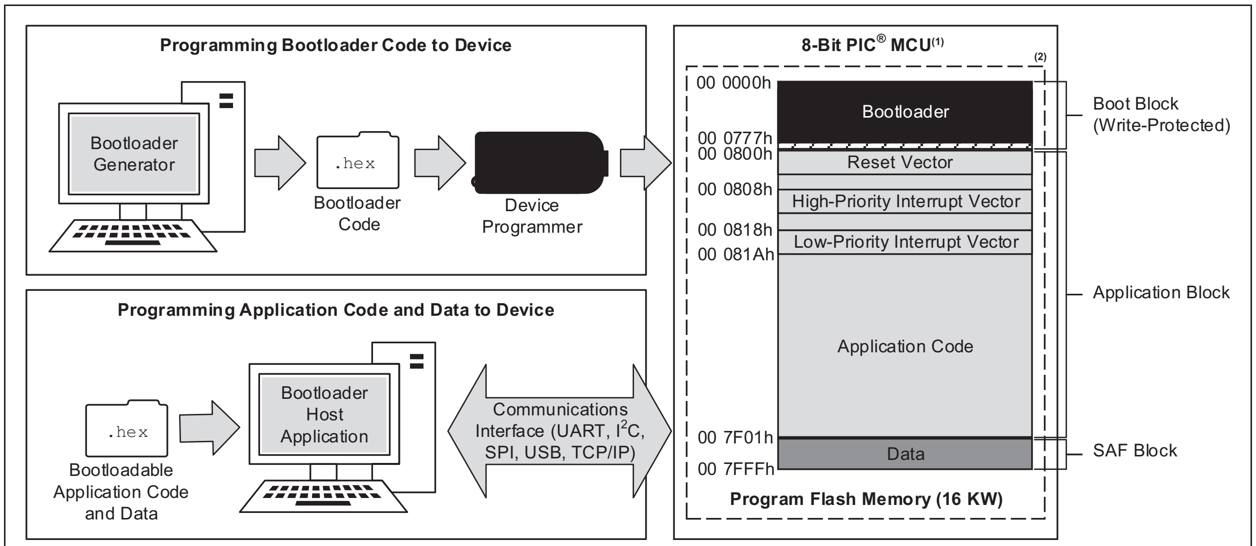
Figure 2 shows how to program the bootloader and the application code on the device. First, the bootloader code is loaded to the device using an external programmer, such as PICkit™ 2, PICkit™ 3, MPLAB® ICD 3 or MPLAB® REAL ICE™. The application code and data to be stored in the PFM are then programmed to the device using a host application. The host application sends the bootloader commands and data through a serial communications interface. These commands tell the bootloader to erase, write or read certain addresses in the application block. For PIC18 devices, access (read, write, erase) to the PFM is controlled via the NVMCON registers. These registers are used in conjunction with the `TBLRD` and `TBLWT` instructions. On the other hand, PIC16 devices access the PFM through the NVMREG interface. For details on program memory access, please refer to the device data sheet.

Note: The PFM of PIC16 devices consists of an array of 14-bit words as user memory and the Hex file is always word-oriented. The PFM of PIC18 devices consists of an array of 16-bit words and the Hex file is always byte-oriented.

The next sections will show how to configure each block for a PIC18 device using the Configuration Words and how to reflect each block in MPLAB X IDE. Please refer to the PFM partitions in Figure 2 for the following examples.

In this example, all three partitions are enabled which means that both the $\overline{\text{BBEN}}$ and $\overline{\text{SAFEN}}$ Configuration bits are set to '`0`'.

FIGURE 2: BOOTLOADER IMPLEMENTATION


Programming Bootloader Code to Device
Bootloader Generator
.hex
Bootloader Code
Device Programmer
Programming Application Code and Data to Device
.hex
Bootloadable Application Code and Data
Bootloader Host Application
Communications Interface (UART, I2C, SPI, USB, TCP/IP)
8-Bit PIC® MCU(1)
(2)
00 0000h
Bootloader
00 0777h
00 0800h
Reset Vector
00 0808h
High-Priority Interrupt Vector
00 0818h
Low-Priority Interrupt Vector
00 081Ah
Application Code
00 7F01h
Data
00 7FFFh
Program Flash Memory (16 KW)
Boot Block (Write-Protected)
Application Block
SAF Block

Note 1: PIC18 CPU.
2: Memory areas are not shown to scale.

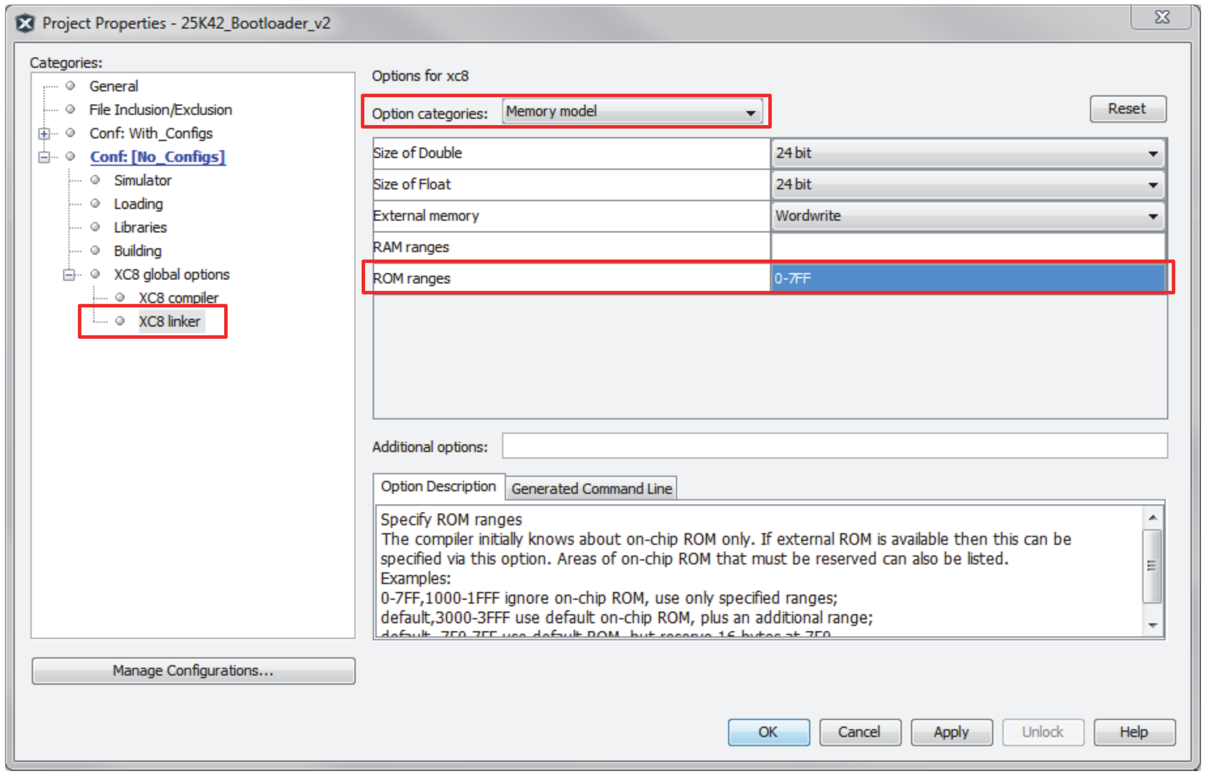
Boot Block Configuration

For the Boot Block configuration shown in Figure 2, BBEN = 0 to enable the Boot Block, BBSIZE<2:0> = 110 for 1024 words, the bootloader end address is at 07FFh and WRTB = 0 to write-protect the Boot Block. Figure 2 shows that the actual bootloader only occupies addresses, from 0000h to 0777h. Reading addresses, 0778h to 07FFh, will always return 0xFF.

When using MPLAB X IDE and XC8, it is necessary to set the ROM (program memory space) ranges to the specified bootloader size to ensure that the bootloader resides in the appropriate memory range after programming. The user can follow the following steps:

1. In MPLAB X IDE, right click the main project, select **Projects** and select the project configuration needed inside the **Categories** box.
2. Click **XC8 linker**. Select **Memory model** on the Option categories drop-down menu.
3. For this example, the ROM ranges are set to 0-7FF, corresponding to the bootloader start address of 0000h and end address of 07FFh (refer to Figure 3).

FIGURE 3: SETTING ROM RANGES IN MPLAB® X IDE



Option categories: Memory model

Size of Double	24 bit
Size of Float	24 bit
External memory	Wordwrite
RAM ranges	
ROM ranges	0-7FF

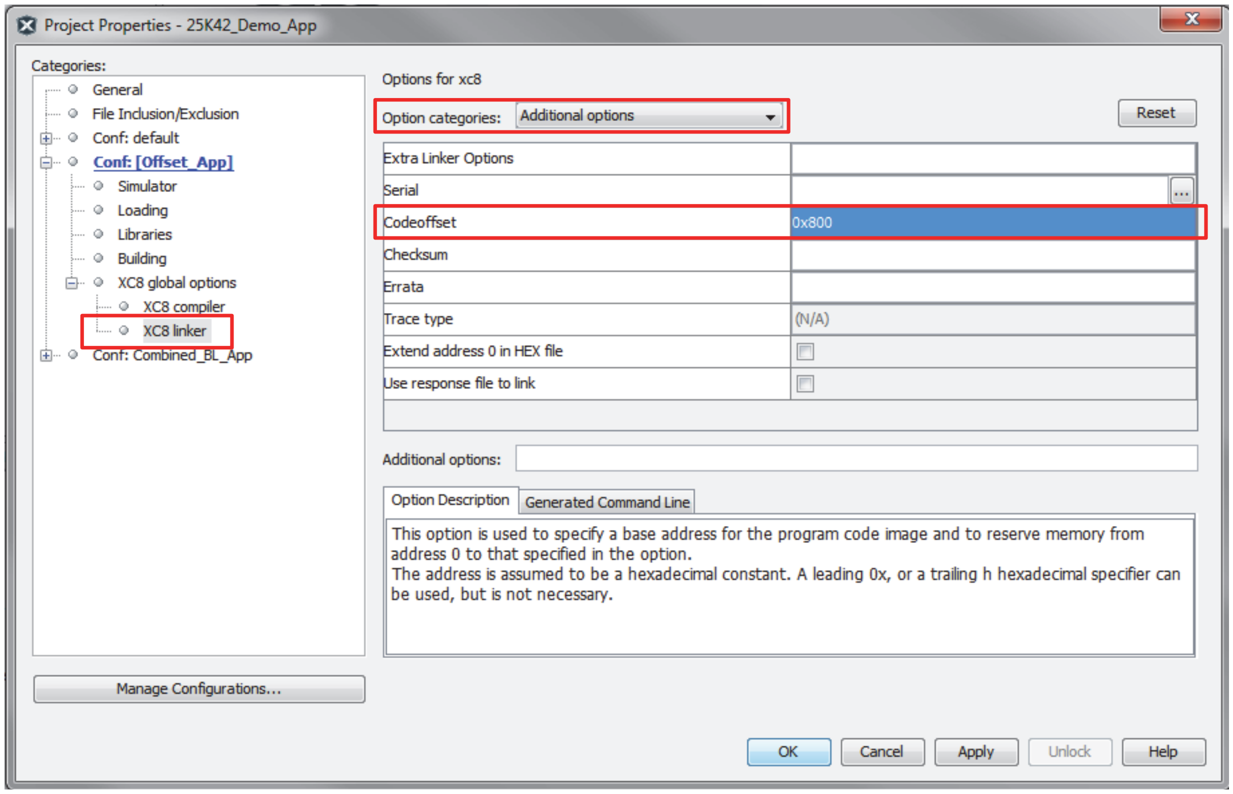
Specify ROM ranges
The compiler initially knows about on-chip ROM only. If external ROM is available then this can be specified via this option. Areas of on-chip ROM that must be reserved can also be listed.
Examples:
0-7FF,1000-1FFF ignore on-chip ROM, use only specified ranges;
default,3000-3FFF use default on-chip ROM, plus an additional range;

Application Block Configuration

As shown in Figure 4, the Application Block is not write-protected and sits on addresses, 0800h to 7F00h. To disable write-protect, set $\overline{\text{WRTAPP}}$ = `0`. To reflect the desired Application Block of 0800h to 7F00h in MPLAB X IDE, the code offset must be configured. The code offset option is used to specify a base address for the application code and to reserve memory for the boot-loader from address 0 to the one specified in the option. This can be done by following the steps below:

1. In MPLAB X IDE, right click the main project, select **Properties** and select the project configuration needed inside the **Categories** box.
2. Click **XC8 linker** and select **Additional options** on the Option categories drop-down menu.
3. Set the Code offset to 0x800 (refer to Figure 4).

FIGURE 4: SETTING CODE OFFSET IN MPLAB® X IDE



Option categories: Additional options

Extra Linker Options	
Serial	
Codeoffset	0x800
Checksum	
Errata	
Trace type	(N/A)
Extend address 0 in HEX file	☐
Use response file to link	☐

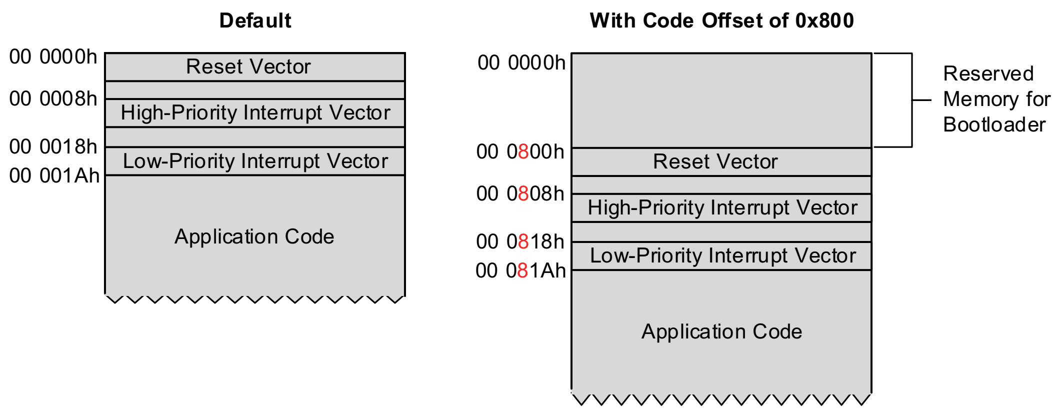
Additional options:

This option is used to specify a base address for the program code image and to reserve memory from address 0 to that specified in the option.
The address is assumed to be a hexadecimal constant. A leading 0x, or a trailing h hexadecimal specifier can be used, but is not necessary.

The code offset option differs from the ROM ranges field previously discussed. Instead of placing the application code inside the specified address range, it will move the Reset and interrupt vectors by the number of words specified in the offset, followed by the application code (refer to Figure 5).

The application code starts at the remapped Reset vector. The bootload will jump into this address when a bootload request is not detected upon start-up. The bootloader should also define the new high and low-priority vectors, based on the code offset, to enable the application code to use interrupts.

FIGURE 5: CODE OFFSET


Default
00 0000h
00 0008h
00 0018h
00 001Ah
Reset Vector
High-Priority Interrupt Vector
Low-Priority Interrupt Vector
Application Code
With Code Offset of 0x800
00 0000h
00 0800h
00 0808h
00 0818h
00 081Ah
Reserved Memory for Bootloader
Reset Vector
High-Priority Interrupt Vector
Low-Priority Interrupt Vector
Application Code

Note: Memory areas are not shown to scale.

SAF Block Configuration

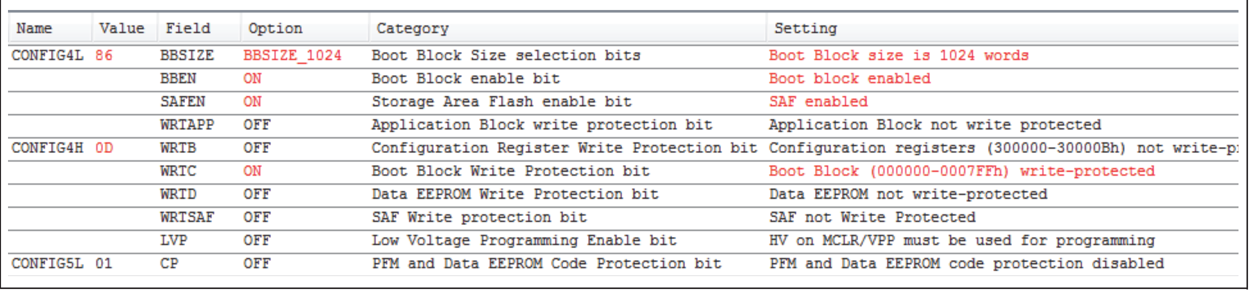
To enable the SAF Block with no write protection, set $\overline{\text{SAFEN}}$ = 0 and $\overline{\text{WRTSAF}}$ = 1. This will allocate the last 128 words (256 bytes for PIC18) for data storage. The allocated memory addresses can be accessed during run time. For PIC18 devices, this is done through Table Reads. It is always important to take note that this block is for storing and accessing data only; otherwise, a memory access violation will occur.

One way of ensuring that executable code does not fill any part of the SAF block is by using structures as shown previously in Example 2. In this example, the size of structure declared is equal to the SAF size; and the structure is explicitly placed in SAF. This technique allows the linker to allocate memory for the `const struct` starting from the specified address, which is also the SAF start address, up to the PFM end address, leaving no space for any executable code in between these addresses.

Configuration Bits Summary

Figure 6 shows a summary of the Configuration bits settings used in the given example. To set the Configuration bits in MPLAB X IDE, click the **Window** drop-down menu, select **PIC Memory views**, then **Configuration bits**. This gives the users an option to generate a source code from the provided settings.

FIGURE 6: SETTING CONFIGURATION BITS IN MPLAB® X IDE



Name	Value	Field	Option	Category	Setting
CONFIG4L	86	BBSIZE	BBSIZE_1024	Boot Block Size selection bits	Boot Block size is 1024 words
		BBEN	ON	Boot Block enable bit	Boot block enabled
		SAFEN	ON	Storage Area Flash enable bit	SAF enabled
		WRTAPP	OFF	Application Block write protection bit	Application Block not write protected
CONFIG4H	0D	WRTB	OFF	Configuration Register Write Protection bit	Configuration registers (300000-30000Bh) not write-p
		WRTC	ON	Boot Block Write Protection bit	Boot Block (000000-0007FFh) write-protected
		WRTD	OFF	Data EEPROM Write Protection bit	Data EEPROM not write-protected
		WRTSAF	OFF	SAF Write protection bit	SAF not Write Protected
		LVP	OFF	Low Voltage Programming Enable bit	HV on MCLR/VPP must be used for programming
CONFIG5L	01	CP	OFF	PFM and Data EEPROM Code Protection bit	PFM and Data EEPROM code protection disabled

For detailed information about bootloaders, please refer to AN851, "A *FLASH Bootloader for PIC16 and PIC18 Devices*" (DS00851) and AN1310, "*High-Speed Serial Bootloader for PIC16 and PIC18 Devices*" (DS01310). Devices mentioned in these documents may not implement the MAP. It is important to take note that the only difference between devices with MAP and devices without MAP, in bootloader implementation, is that the devices with MAP have additional Configuration bits for partition enable, write protection and Boot Block size selection.

CONCLUSION

This Technical Brief covered all the basic details about the Memory Access Partition on 8-bit PIC microcontrollers. A sample bootloader implementation for PIC18 was shown to demonstrate the purpose of each block and how to configure them given certain memory sizes and write-protect requirements. Project configurations in MPLAB X IDE were also discussed to help readers understand the necessary adjustments before programming the bootloader and application code on the device. When properly implemented, the MAP can be very useful in developing a wide variety of embedded applications, especially those requiring a bootloader, write-protect and extra data storage.

REFERENCES

- *"PIC18(L)F24/25K42 Data Sheet"* (DS40001869)
- *"PIC16(L)F15325/45 Data Sheet"* (DS40001865)
- *"MPLAB® XC8 C Compiler User's Guide"* (DS50002053)

NOTES:

ISBN: 978-1-5224-1365-3

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

11/07/16