Section 26. CodeGuard™ Security

HIGHLIGHTS

This section of the manual contains the following major topics:

26.1 Code Protection Overview

Microchip's CodeGuard™ Security enables multiple parties to securely share resources (memory, interrupts and peripherals) on a single chip. Intellectual Property (IP) vendors, Original Design/Original Equipment Manufacturers (ODM/OEM) and Value-Added Resellers (VAR) now have an opportunity to reap the following benefits using these on-chip code protection features:

- System cost reduction
- Component reduction and associated benefits to inventory management
- Decreased risk of losing IP to unqualified partners
- Increased security during code distribution and Flash memory update

CodeGuard Security is a turnkey solution with basic, intermediate and advanced implementations on Microchip's 16-bit Flash memory-based dsPIC® Digital Signal Controllers (DSCs). The advanced implementation of CodeGuard Security can be found on many devices in the dsPIC30F product family.

The on-chip program Flash memory in a dsPIC30F device can be organized into three code space segments. Each of these segments has an implied security privilege level and system function.

1. The Boot Segment (BS) has the highest security privilege level. Generally speaking, it has unrestricted access to the other segments. The Boot Segment is intended for secure boot loader and device update functions.
2. The Secure Segment (SS) has the next highest security privilege. This segment is intended for storing proprietary algorithms from algorithm vendors.
3. The General Segment (GS) has the lowest security privilege. This segment is intended for the end user system code.

Segments of user data RAM space of the device can be allocated as secure RAM directly associated with the Boot or Secure segments. On devices that contain data EEPROM, segments of the data EEPROM can be allocated as secure EEPROM and associated with the Boot or Secure segments.

Any operation of the system that potentially allows exposure of the code or data contents is restricted based on the segment from which the operation originated or the segment the operation targets.

Restricted operations include the following:

- Programming, Erase or Verify Operations
- Reads or Writes of Code Space
- Code Flow Change into a Secure Segment from outside the segment
- Interrupt Vectors into a Secure Segment

Configuration bits are provided to enable access to the secure segments and their parameters. These bits allow configuration of both the sizes and restrictions of the program Flash memory, RAM and EEPROM segments.

26.2 Device Specific Code Protection Features

Three different subsets of CodeGuard Security features are available on dsPIC30F devices.

On many of the devices, all memory is allocated to General Segment space. It is not possible to allocate Boot Segment or Secure Segment code space, or to protect Data RAM or Data EEPROM.

On some devices with smaller memory sizes, memory can be allocated to Boot Segment space and General Segment code space. However, there is no Secure Segment, no RAM or Data EEPROM protection and no high security level.

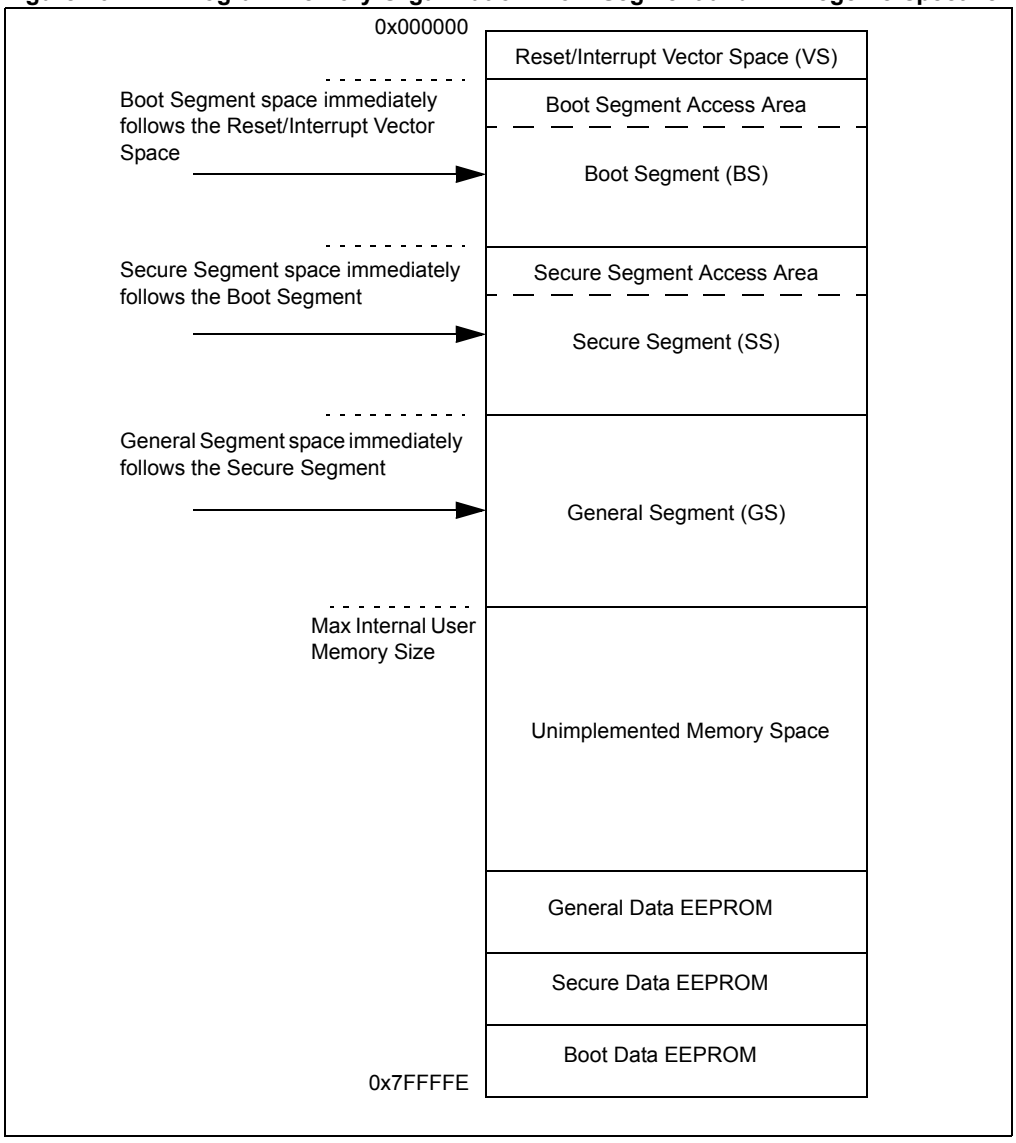
On some larger memory devices, memory can be allocated to Boot Segment, Secure Segment, and General Segment code space. Data RAM and EEPROM, if available, can be allocated to Boot and Secure Segments.

Refer to the specific data sheet to correlate these features with specific devices or product families.

26.3 Program Memory Organization

The user program memory can be allocated into the three segments: General, Secure and Boot. The size of these different segments is determined by configuration bits. The relative locations of the segments do not change. A Boot Segment, if present, occupies the memory area just after the device interrupt vector space. The Secure Segment, if present, occupies the space just after the Boot Segment, and the General Segment occupies the space just after the Secure Segment (see Figure 26-1).

Figure 26-1: Program Memory Organization From Segment and Privilege Perspective

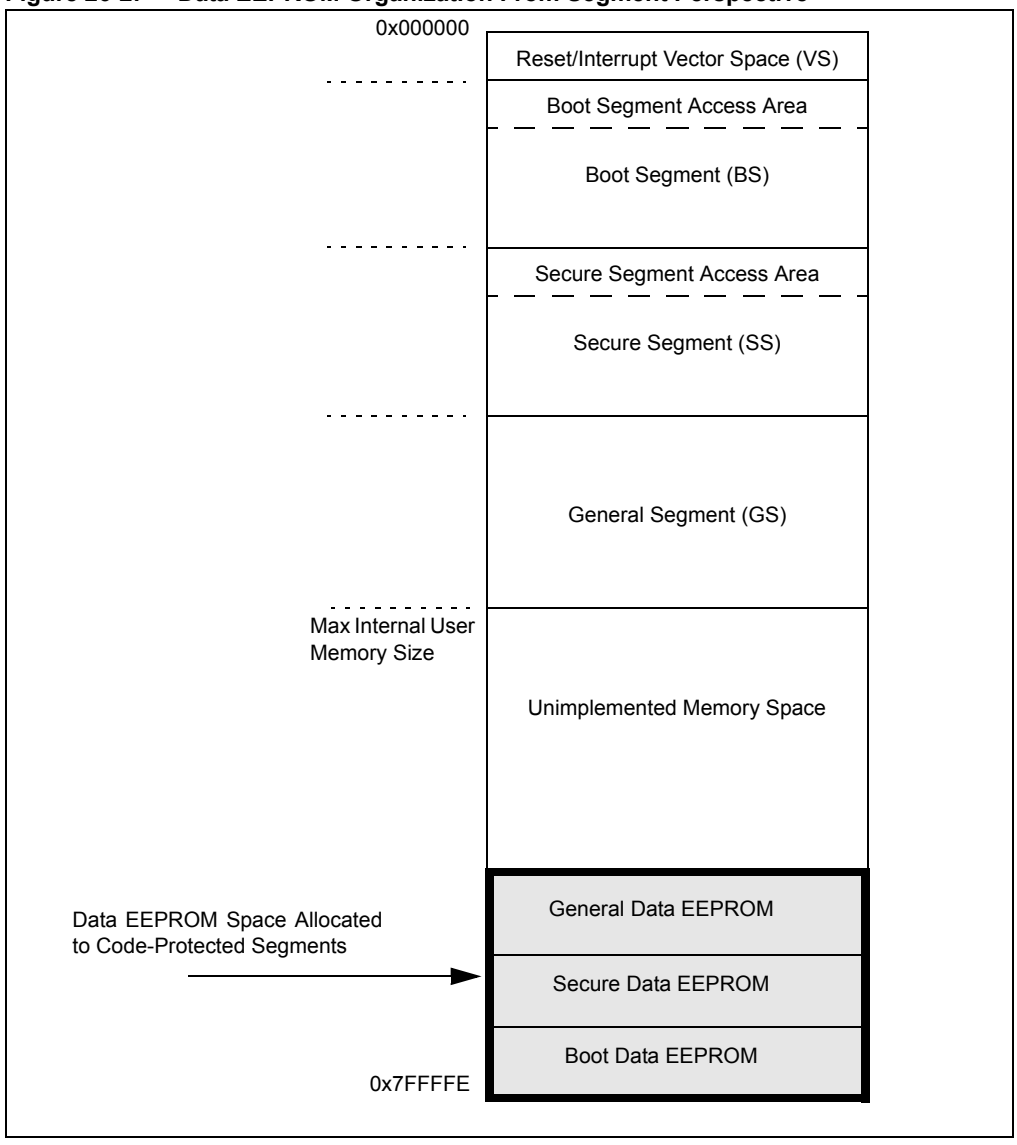


26.4 Data EEPROM Organization

The data EEPROM memory can be allocated into the three segments: General, Secure and Boot. Segment size is specified by configuration bits. The relative location of the segments will not change. Data EEPROM space allocated to the Boot Segment occupies the highest memory region of the data EEPROM. Data EEPROM allocated to the Secure Segment occupies the space immediately below the Boot Segment. The remainder of data EEPROM memory is allocated to the General Segment (see Figure 26-2)

Note:	Data EEPROM is not present on all devices. Refer to the specific device data sheet for Data EEPROM availability and size.

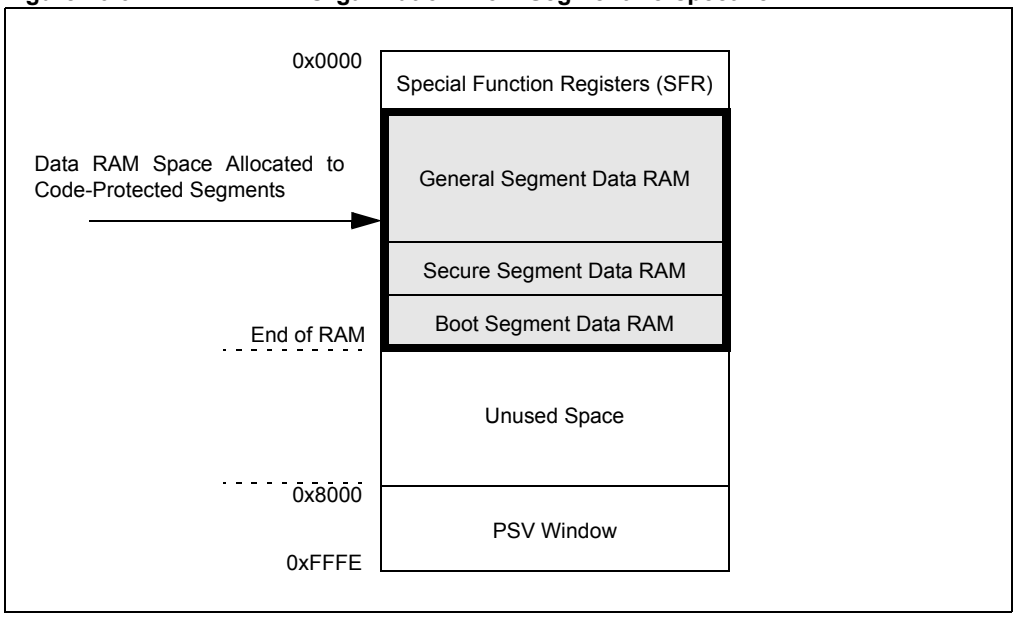
Figure 26-2: Data EEPROM Organization From Segment Perspective



26.5 Data RAM Organization

Data RAM memory can also be allocated into code protection segments: Boot, Secure and General. Segment size is primarily specified by configuration bits. The relative locations of the segments do not change, meaning that a Boot Segment RAM area occupies the memory region at the end of RAM, the Secure Segment RAM occupies the area just before the Boot Segment and the General Segment RAM occupies the remainder of the data RAM space (see Figure 26-3).

Figure 26-3: DATA RAM Organization From Segment Perspective



26.6 Control Registers

Several Configuration and Special Function Registers control the security functions. On basic and intermediate security implementations, some of these registers do not exist. The key registers for supporting the code security features are as follows:

- FBS – Boot Segment Configuration Register
- FSS – Secure Segment Configuration Register
- FGS – General Segment Configuration Register
- BSRAM – Boot Segment RAM Special Function Register
- SSRAM – Secure Segment RAM Special Function Register
- INTTREG – Interrupt Vector and Priority Status Special Function Register

26.7 The Boot Segment (BS)

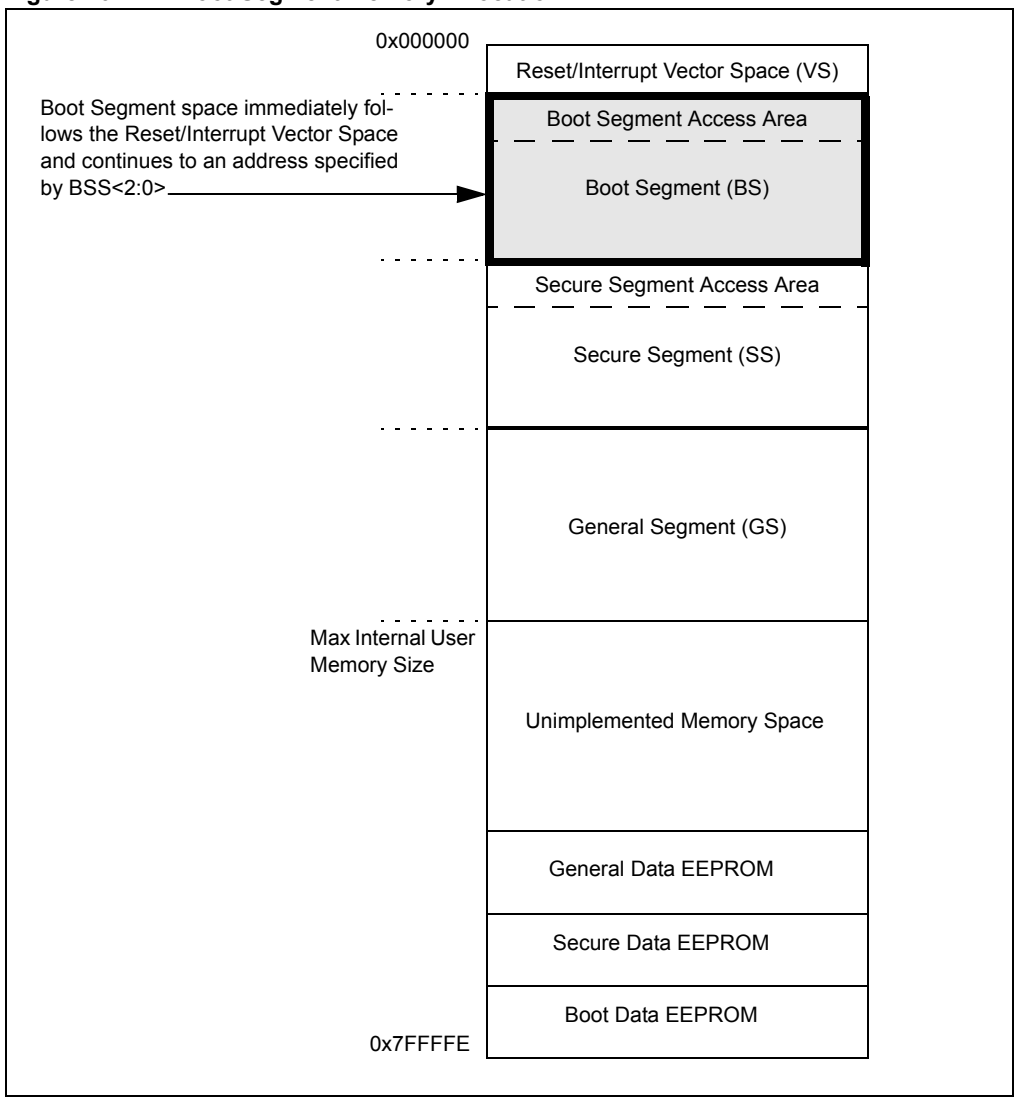
The Boot Segment has the highest privilege. The Boot Segment can be small, allowing a simple yet secure boot loader, or it can be large, enabling it to hold a more sophisticated secure operating system.

The Boot Segment can also rewrite its own locations, enabling it to store and update data such as “encryption keys”.

26.7.1 Allocating the Boot Segment

The existence and size of the Boot Segment are determined by configuration bits BSS<2:0> (FBS<3:1>). The default option on an erased, non-programmed device is to exclude the Boot Segment. When implemented, the Boot Segment begins at the end of the interrupt vector space and continues to an address specified by the BSS<2:0> bits.

Figure 26-4: Boot Segment Memory Allocation



26.7.1.1 Boot Segment Size Options

Table 26-1 is an example of Boot Segment (BS) size options for the dsPIC30F family. The start and end program memory addresses are typical. Refer to the device data sheet for specific program memory addresses for a given device.

Table 26-1: Boot Segment Size Options Example

BSS<2:0>	Security Level	BS Size	BS Start Address	BS End Address
x11	No Boot Program Flash Segment			
110	Standard	Small	0x000100	0x0003FE
010	High	Small	0x000100	0x0003FE
101	Standard	Medium	0x000100	0x00FFE
001	High	Medium	0x000100	0x00FFE
100	Standard	Large	0x000100	0x001FFE
000	High	Large	0x000100	0x001FFE

Register 26-1: FBS: Boot Segment Configuration Register

Upper Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 23							bit 16

Middle Byte:							
U-0	U-0	R/W-0	R/W-0	U-0	U-0	U-0	R/W-0
—	—	RBS1	RBS0	—	—	—	EBS
bit 15							bit 8

Lower Byte:							
U-0	U-0	U-0	U-0	R/P	R/P	R/P	R/P
—	—	—	—	BSS2	BSS1	BSS0	BWRP
bit 7							bit 0

bit 23-14 **Unimplemented:** Read as '0'

bit 13-12 **RBS<1:0>:** Boot Segment RAM Code Protection bits(1)
11 = No Boot RAM segment
10 = Small Boot RAM Segment
01 = Medium Boot RAM Segment
00 = Large Boot RAM Segment

bit 11-9 **Unimplemented:** Read as '0'

bit 8 **EBS:** Boot Segment Data EEPROM Code Protection bit
1 = No Boot Data EEPROM Segment
0 = Boot Data EEPROM Segment is 256 bytes
Data EEPROM configuration depends on EBS and ESS<1:0> (FSS<9:8>) bit settings

bit 7-4 **Unimplemented:** Read as '0'

bit 3-1 **BSS<2:0>:** Boot Segment Program Flash Code Protection bits(2)
x11 = No Boot program Flash segment
110 = Standard security, small Boot Segment
010 = High security, small Boot Segment
101 = Standard security, medium Boot Segment
001 = High security, medium Boot Segment
100 = Standard security, large Boot Segment
000 = High security, large Boot Segment

bit 0 **BWRP:** Boot Segment Program Flash Write Protection bit
1 = Boot segment can be written
0 = Boot segment is write-protected

Note 1: Not all devices have Boot RAM and Boot Data EEPROM protection. For specific device information, see Table 26-2, Table 26-3, Table 26-4, Table 26-5, Table 26-6, and Table 26-7.

2: The exact definitions of Small, Medium, and Large Boot Program Flash and Boot RAM Segments vary from one device to another. For device specific information, see Table 26-8, Table 26-9, Table 26-10, Table 26-11 and Table 26-12.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Table 26-2: Data RAM Segment Sizes for Devices with 4 KB RAM

CONFIGURATION BITS	RBS<1:0> = 11 OR RBS<1:0> = 10 AND RL_BSR = 1	RBS<1:0> = 10 AND RL_BSR = 0 OR RBS<1:0> = 01 AND RL_BSR = 1	RBS<1:0> = 01 AND RL_BSR = 0 OR RBS<1:0> = 00 AND RL_BSR = 1	RBS<1:0> = 00 AND RL_BSR = 0
RSS<1:0> = 11 OR RSS<1:0> = 10 AND RL_SSR = 1	TOTAL RAM = 4096 GS RAM = 4096 0x0800 0x17FF	TOTAL RAM = 4096 GS RAM = 3968 BS RAM = 128 0x0800 0x1780 0x17FF	TOTAL RAM = 4096 GS RAM = 3840 BS RAM = 256 0x0800 0x1700 0x17FF	TOTAL RAM = 4096 GS RAM = 3584 BS RAM = 512 0x0800 0x1600 0x17FF
RSS<1:0> = 10 AND RL_SSR = 0 OR RSS<1:0> = 01 AND RL_SSR = 1	TOTAL RAM = 4096 GS RAM = 3840 SS RAM = 256 0x0800 0x1700 0x17FF	TOTAL RAM = 4096 GS RAM = 3840 SS RAM = 128 BS RAM = 128 0x0800 0x1000 0x1700 0x1780 0x17FF	TOTAL RAM = 4096 GS RAM = 3840 BS RAM = 256 0x0800 0x1700 0x17FF	TOTAL RAM = 4096 GS RAM = 3584 BS RAM = 512 0x0800 0x1600 0x17FF
RSS<1:0> = 01 AND RL_SSR = 0 OR RSS<1:0> = 00 AND RL_SSR = 1	TOTAL RAM = 4096 GS RAM = 3072 SS RAM = 1024 0x0800 0x1400 0x17FF	TOTAL RAM = 4096 GS RAM = 3072 SS RAM = 896 BS RAM = 128 0x0800 0x1400 0x1780 0x17FF	TOTAL RAM = 4096 GS RAM = 3072 SS RAM = 768 BS RAM = 256 0x0800 0x1400 0x1700 0x17FF	TOTAL RAM = 4096 GS RAM = 3072 SS RAM = 512 BS RAM = 512 0x0800 0x1400 0x1600 0x17FF
RSS<1:0> = 00 AND RL_SSR = 0	TOTAL RAM = 4096 GS RAM = 2048 SS RAM = 2048 0x0800 0x1000 0x17FF	TOTAL RAM = 4096 GS RAM = 2048 SS RAM = 1920 BS RAM = 128 0x0800 0x1000 0x1780 0x17FF	TOTAL RAM = 4096 GS RAM = 2048 SS RAM = 1792 BS RAM = 256 0x0800 0x1000 0x1700 0x17FF	TOTAL RAM = 4096 GS RAM = 2048 SS RAM = 1536 BS RAM = 512 0x0800 0x1000 0x1600 0x17FF

Legend: OR = Logical OR, AND = Logical AND

Table 26-3: Data RAM Segment Sizes for Devices with 8 KB RAM

CONFIGURATION BITS	RBS<1:0> = 11 OR RBS<1:0> = 10 AND RL_BSR = 1	RBS<1:0> = 10 AND RL_BSR = 0 OR RBS<1:0> = 01 AND RL_BSR = 1	RBS<1:0> = 01 AND RL_BSR = 0 OR RBS<1:0> = 00 AND RL_BSR = 1	RBS<1:0> = 00 AND RL_BSR = 0
RSS<1:0> = 11 OR RSS<1:0> = 10 AND RL_SSR = 1	TOTAL RAM = 8192 GS RAM = 8192 0x0800 0x27FF	TOTAL RAM = 8192 GS RAM = 8064 BS RAM = 128 0x0800 0x2780 0x27FF	TOTAL RAM = 8192 GS RAM = 7936 BS RAM = 256 0x0800 0x2700 0x27FF	TOTAL RAM = 8192 GS RAM = 7168 BS RAM = 1024 0x0800 0x2400 0x27FF
RSS<1:0> = 10 AND RL_SSR = 0 OR RSS<1:0> = 01 AND RL_SSR = 1	TOTAL RAM = 8192 GS RAM = 7936 SS RAM = 256 0x0800 0x2700 0x27FF	TOTAL RAM = 8192 GS RAM = 7936 SS RAM = 128 BS RAM = 128 0x0800 0x2700 0x2780 0x27FF	TOTAL RAM = 8192 GS RAM = 7936 BS RAM = 256 0x0800 0x2700 0x27FF	TOTAL RAM = 8192 GS RAM = 7168 BS RAM = 1024 0x0800 0x2400 0x27FF
RSS<1:0> = 01 AND RL_SSR = 0 OR RSS<1:0> = 00 AND RL_SSR = 1	TOTAL RAM = 8192 GS RAM = 6144 SS RAM = 2048 0x0800 0x2000 0x27FF	TOTAL RAM = 8192 GS RAM = 6144 SS RAM = 1920 BS RAM = 128 0x0800 0x2000 0x2780 0x27FF	TOTAL RAM = 8192 GS RAM = 6144 SS RAM = 1792 BS RAM = 256 0x0800 0x2000 0x2700 0x27FF	TOTAL RAM = 8192 GS RAM = 6144 SS RAM = 1024 BS RAM = 1024 0x0800 0x2000 0x2400 0x27FF
RSS<1:0> = 00 AND RL_SSR = 0	TOTAL RAM = 8192 GS RAM = 4096 SS RAM = 4096 0x0800 0x1800 0x27FF	TOTAL RAM = 8192 GS RAM = 4096 SS RAM = 3968 BS RAM = 128 0x0800 0x1800 0x2780 0x27FF	TOTAL RAM = 8192 GS RAM = 4096 SS RAM = 3840 BS RAM = 256 0x0800 0x1800 0x2700 0x27FF	TOTAL RAM = 8192 GS RAM = 4096 SS RAM = 3072 BS RAM = 1024 0x0800 0x1800 0x2400 0x27FF

Legend: OR = Logical OR, AND = Logical AND

Table 26-4: Data RAM Segment Sizes for Devices with 6 KB RAM

CONFIGURATION BITS	RBS<1:0> = 11 OR RBS<1:0> = 10 AND RL_BSR = 1	RBS<1:0> = 10 AND RL_BSR = 0 OR RBS<1:0> = 01 AND RL_BSR = 1	RBS<1:0> = 01 AND RL_BSR = 0 OR RBS<1:0> = 00 AND RL_BSR = 1	RBS<1:0> = 00 AND RL_BSR = 0
RSS<1:0> = 11 OR RSS<1:0> = 10 AND RL_SSR = 1	GS RAM = 6144 0x0800 0x1FFF	GS RAM = 6016 0x0800 BS RAM = 128 0x1F80 0x1FFF	GS RAM = 5888 0x0800 BS RAM = 256 0x1F00 0x1FFF	GS RAM = 5632 0x0800 BS RAM = 512 0x1E00 0x1FFF
RSS<1:0> = 10 AND RL_SSR = 0 OR RSS<1:0> = 01 AND RL_SSR = 1	GS RAM = 5888 0x0800 SS RAM = 256 0x1F00 0x1FFF	GS RAM = 5888 0x0800 SS RAM = 128 0x1F00 BS RAM = 128 0x1F80 0x1FFF	GS RAM = 5888 0x0800 BS RAM = 256 0x1F00 0x1FFF	GS RAM = 5632 0x0800 BS RAM = 512 0x1E00 0x1FFF
RSS<1:0> = 01 AND RL_SSR = 0 OR RSS<1:0> = 00 AND RL_SSR = 1	GS RAM = 5120 0x0800 SS RAM = 1024 0x1C00 0x1FFF	GS RAM = 5120 0x0800 SS RAM = 896 0x1C00 BS RAM = 128 0x1F80 0x1FFF	GS RAM = 5120 0x0800 SS RAM = 768 0x1C00 BS RAM = 256 0x1F00 0x1FFF	GS RAM = 5120 0x0800 SS RAM = 512 0x1C00 BS RAM = 512 0x1E00 0x1FFF
RSS<1:0> = 00 AND RL_SSR = 0	GS RAM = 4096 0x0800 SS RAM = 2048 0x1800 0x1FFF	GS RAM = 4096 0x0800 SS RAM = 1920 0x1800 BS RAM = 128 0x1F80 0x1FFF	GS RAM = 4096 0x0800 SS RAM = 1792 0x1800 BS RAM = 256 0x1F00 0x1FFF	GS RAM = 4096 0x0800 SS RAM = 1536 0x1800 BS RAM = 512 0x1E00 0x1FFF

Legend: OR = Logical OR, AND = Logical AND

Table 26-5: Data EEPROM Segment Sizes for Devices with 1 KB Data EEPROM

Configuration Bits		
ESS <1:0>	**EBS = 1**	**EBS = 0**
11	TOTAL EE = 1024 GS EE = 1024 0x7FFC00 0x7FFFFE	TOTAL EE = 1024 GS EE = 896 0x7FFC00 BS EE = 128 0x7FFF80 0x7FFFFE
10	TOTAL EE = 1024 GS EE = 896 0x7FFC00 SS EE = 128 0x7FFF80 0x7FFFFE	TOTAL EE = 1024 GS EE = 896 0x7FFC00 BS EE = 128 0x7FFF80 0x7FFFFE
01	TOTAL EE = 1024 GS EE = 768 0x7FFC00 SS EE = 256 0x7FFF00 0x7FFFFE	TOTAL EE = 1024 GS EE = 768 0x7FFC00 SS EE = 128 0x7FFF00 BS EE = 128 0x7FFF80 0x7FFFFE
00	TOTAL EE = 1024 GS EE = 512 0x7FFC00 SS EE = 512 0x7FFE00 0x7FFFFE	TOTAL EE = 1024 GS EE = 512 0x7FFC00 SS EE = 384 0x7FFE00 BS EE = 128 0x7FFF80 0x7FFFFE

Table 26-6: Data EEPROM Segment Sizes for Devices with 4 KB Data EEPROM

Configuration Bits		
ESS <1:0>	**EBS = 1**	**EBS = 0**
11	TOTAL EE = 4096 GS EE = 4096 0x7FF000 0x7FFFFE	TOTAL EE = 4096 GS EE = 3840 0x7FF000 BS EE = 256 0x7FFF00 0x7FFFFE
10	TOTAL EE = 4096 GS EE = 3840 0x7FF000 SS EE = 256 0x7FFF00 0x7FFFFE	TOTAL EE = 4096 GS EE = 3840 0x7FF000 BS EE = 256 0x7FFF00 0x7FFFFE
01	TOTAL EE = 4096 GS EE = 3584 0x7FF000 SS EE = 512 0x7FFE00 0x7FFFFE	TOTAL EE = 4096 GS EE = 3584 0x7FF000 SS EE = 256 0x7FFE00 BS EE = 256 0x7FFF00 0x7FFFFE
00	TOTAL EE = 4096 GS EE = 2048 0x7FF000 SS EE = 2048 0x7FF800 0x7FFFFE	TOTAL EE = 4096 GS EE = 2048 0x7FF000 SS EE = 1792 0x7FF800 BS EE = 256 0x7FFF00 0x7FFFFE

Table 26-7: Data EEPROM Segment Sizes for Devices with 2 KB Data EEPROM

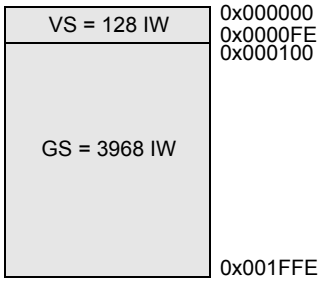
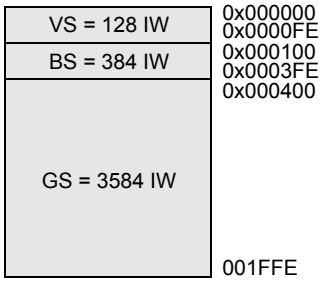
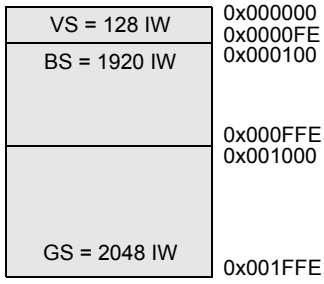
Configuration Bits		
ESS <1:0>	**EBS = 1**	**EBS = 0**
11	TOTAL EE = 2048 GS EE = 2048 0x7FF800 0x7FFFFE	TOTAL EE = 2048 GS EE = 1792 0x7FF800 BS EE = 256 0x7FFF00 0x7FFFFE
10	TOTAL EE = 2048 GS EE = 1792 0x7FF800 SS EE = 256 0x7FFF00 0x7FFFFE	TOTAL EE = 2048 GS EE = 1792 0x7FF800 BS EE = 256 0x7FFF00 0x7FFFFE
01	TOTAL EE = 2048 GS EE = 1536 0x7FF800 SS EE = 512 0x7FFE00 0x7FFFFE	TOTAL EE = 2048 GS EE = 1536 0x7FF800 SS EE = 256 0x7FFE00 BS EE = 256 0x7FFF00 0x7FFFFE
00	TOTAL EE = 2048 GS EE = 1024 0x7FF800 SS EE = 1024 0x7FFC00 0x7FFFFE	TOTAL EE = 2048 GS EE = 1024 0x7FF800 SS EE = 768 0x7FFC00 BS EE = 256 0x7FFF00 0x7FFFFE

Table 26-8: Program Flash Segment Sizes for Devices with 6 KB Program Flash

Configuration Bits	
BSS<2:0>=x11	**BSS<2:0>=x10**
VS = 128 IW 0x000000 0x0000FE 0x000100 GS = 1920 IW 0x000FFE	VS = 128 IW 0x000000 0x0000FE BS = 384 IW 0x000100 0x0003FE 0x000400 GS = 1536 IW 0x000FFE

Legend: IW = Instruction Words

Table 26-9: Program Flash Segment Sizes for Devices with 12 KB Program Flash

Configuration Bits		
BSS<2:0>=x11	**BSS<2:0>=x10**	**BSS<2:0>=x01**
		

Legend: IW = Instruction Words

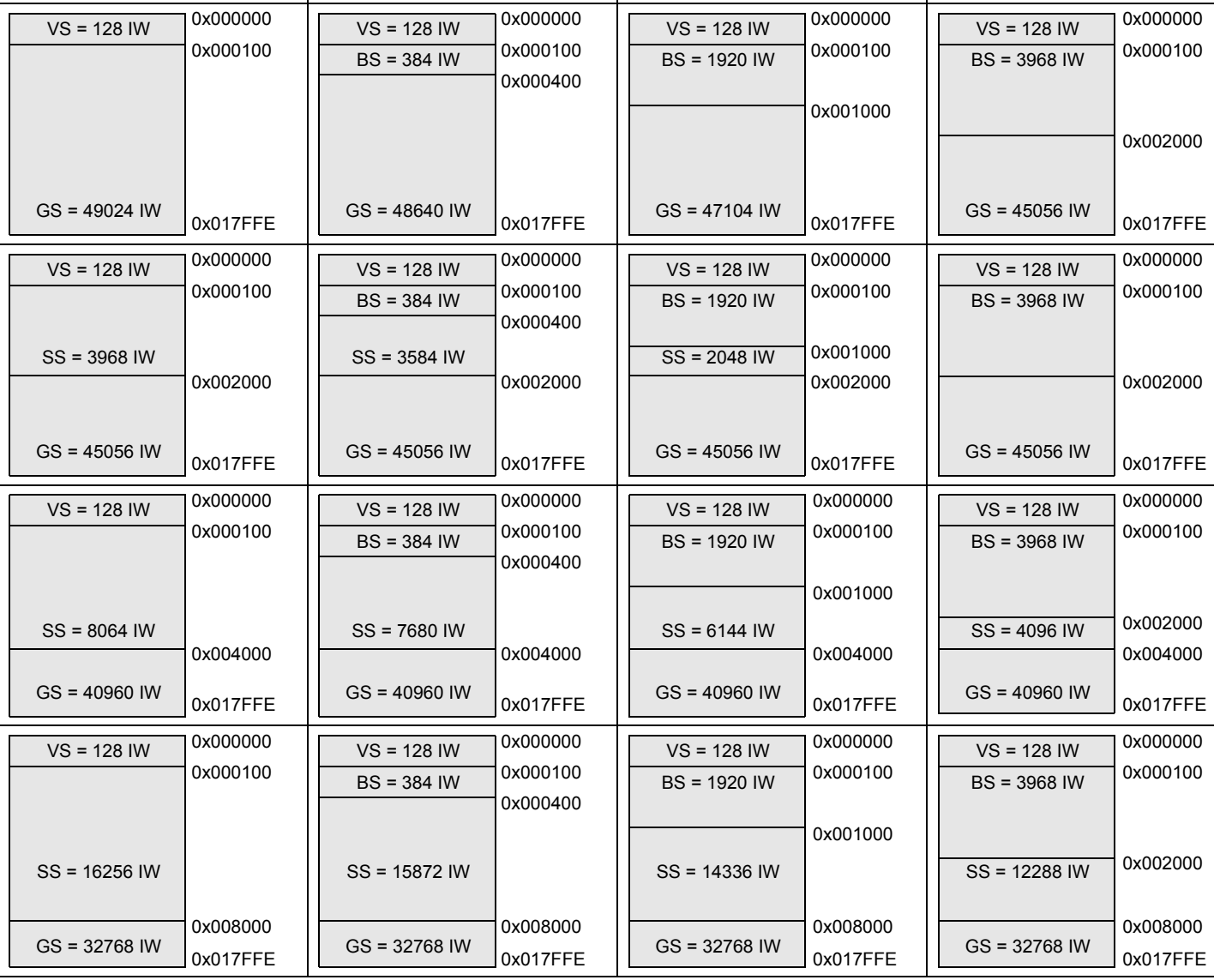
Table 26-10: Program Flash Segment Sizes for Devices with 66 KB Program Flash

CONFIGURATION BITS	BSS<2:0> = x11	BSS<2:0> = x10	BSS<2:0> = x01	BSS<2:0> = x00
SSS<2:0> = x11	VS = 128 IW GS = 22400 IW 0x000000 0x000100 0x00AFFE	VS = 128 IW BS = 384 IW GS = 22016 IW 0x000000 0x000100 0x000400 0x00AFFE	VS = 128 IW BS = 1920 IW GS = 20480 IW 0x000000 0x000100 0x001000 0x00AFFE	VS = 128 IW BS = 3968 IW GS = 18432 IW 0x000000 0x000100 0x002000 0x00AFFE
SSS<2:0> = x10	VS = 128 IW SS = 3968 IW GS = 18432 IW 0x000000 0x000100 0x001FFE 0x00AFFE	VS = 128 IW BS = 384 IW SS = 3584 IW GS = 18432 IW 0x000000 0x000100 0x000400 0x002000 0x00AFFE	VS = 128 IW BS = 1920 IW SS = 2048 IW GS = 18432 IW 0x000000 0x000100 0x001000 0x002000 0x00AFFE	VS = 128 IW BS = 3968 IW GS = 18432 IW 0x000000 0x000100 0x002000 0x00AFFE
SSS<2:0> = x01	VS = 128 IW SS = 8064 IW GS = 14336 IW 0x000000 0x000100 0x004000 0x00AFFE	VS = 128 IW BS = 384 IW SS = 7680 IW GS = 14336 IW 0x000000 0x000100 0x000400 0x004000 0x00AFFE	VS = 128 IW BS = 1920 IW SS = 6144 IW GS = 14336 IW 0x000000 0x000100 0x001000 0x004000 0x00AFFE	VS = 128 IW BS = 3968 IW SS = 4096 IW GS = 14336 IW 0x000000 0x000100 0x002000 0x004000 0x00AFFE
SSS<2:0> = x00	VS = 128 IW SS = 16256 IW GS = 6144 IW 0x000000 0x000100 0x008000 0x00AFFE	VS = 128 IW BS = 384 IW SS = 15872 IW GS = 6144 IW 0x000000 0x000100 0x000400 0x008000 0x00AFFE	VS = 128 IW BS = 1920 IW SS = 14336 IW GS = 6144 IW 0x000000 0x000100 0x001000 0x008000 0x00AFFE	VS = 128 IW BS = 3968 IW SS = 12288 IW GS = 6144 IW 0x000000 0x000100 0x002000 0x008000 0x00AFFE

Legend: IW = Instruction Words

Table 26-11: Program Flash Segment Sizes for Devices with 144 KB Program Flash

CONFIGURATION BITS	BSS<2:0> = x11	BSS<2:0> = x10	BSS<2:0> = x01	BSS<2:0> = x00
SSS<2:0> = x11	VS = 128 IW GS = 49024 IW	VS = 128 IW BS = 384 IW GS = 48640 IW	VS = 128 IW BS = 1920 IW GS = 47104 IW	VS = 128 IW BS = 3968 IW GS = 45056 IW
SSS<2:0> = x10	VS = 128 IW SS = 3968 IW GS = 45056 IW	VS = 128 IW BS = 384 IW SS = 3584 IW GS = 45056 IW	VS = 128 IW BS = 1920 IW SS = 2048 IW GS = 45056 IW	VS = 128 IW BS = 3968 IW GS = 45056 IW
SSS<2:0> = x01	VS = 128 IW SS = 8064 IW GS = 40960 IW	VS = 128 IW BS = 384 IW SS = 7680 IW GS = 40960 IW	VS = 128 IW BS = 1920 IW SS = 6144 IW GS = 40960 IW	VS = 128 IW BS = 3968 IW SS = 4096 IW GS = 40960 IW
SSS<2:0> = x00	VS = 128 IW SS = 16256 IW GS = 32768 IW	VS = 128 IW BS = 384 IW SS = 15872 IW GS = 32768 IW	VS = 128 IW BS = 1920 IW SS = 14336 IW GS = 32768 IW	VS = 128 IW BS = 3968 IW SS = 12288 IW GS = 32768 IW



Legend: IW = Instruction Words

Table 26-12: Program Flash Segment Sizes for Devices with 132 KB Program Flash

CONFIGURATION BITS	BSS<2:0> = x11	BSS<2:0> = x10	BSS<2:0> = x01	BSS<2:0> = x00
SSS<2:0> = x11	VS = 128 IW (0x000000–0x000100) GS = 44928 IW (0x000100–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 384 IW (0x000100–0x000400) GS = 44544 IW (0x000400–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 1920 IW (0x000100–0x001000) GS = 43008 IW (0x001000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 3968 IW (0x000100–0x002000) GS = 40960 IW (0x002000–0x015FFE)
SSS<2:0> = x10	VS = 128 IW (0x000000–0x000100) SS = 3968 IW (0x000100–0x002000) GS = 40960 IW (0x002000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 384 IW (0x000100–0x000400) SS = 3584 IW (0x000400–0x002000) GS = 40960 IW (0x002000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 1920 IW (0x000100–0x001000) SS = 2048 IW (0x001000–0x002000) GS = 40960 IW (0x002000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 3968 IW (0x000100–0x002000) GS = 40960 IW (0x002000–0x015FFE)
SSS<2:0> = x01	VS = 128 IW (0x000000–0x000100) SS = 8064 IW (0x000100–0x004000) GS = 36864 IW (0x004000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 384 IW (0x000100–0x000400) SS = 7680 IW (0x000400–0x004000) GS = 36864 IW (0x004000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 1920 IW (0x000100–0x001000) SS = 6144 IW (0x001000–0x004000) GS = 36864 IW (0x004000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 3968 IW (0x000100–0x002000) SS = 4096 IW (0x002000–0x004000) GS = 36864 IW (0x004000–0x015FFE)
SSS<2:0> = x00	VS = 128 IW (0x000000–0x000100) SS = 16256 IW (0x000100–0x008000) GS = 28672 IW (0x008000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 384 IW (0x000100–0x000400) SS = 15872 IW (0x000400–0x008000) GS = 28672 IW (0x008000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 1920 IW (0x000100–0x001000) SS = 14336 IW (0x001000–0x008000) GS = 28672 IW (0x008000–0x015FFE)	VS = 128 IW (0x000000–0x000100) BS = 3968 IW (0x000100–0x002000) SS = 12288 IW (0x002000–0x008000) GS = 28672 IW (0x008000–0x015FFE)

Legend: IW = Instruction Words

26.7.2 Selecting the Security Level of the Boot Segment

The security level of the Boot Segment is determined by the configuration bit BSS2 (FBS<3>):

1 = Standard security

0 = High security

When the Boot Segment is configured for high security, the number of access methods is more limited than with standard security. The differences are noted in the following paragraphs. See **Section 26.12 "Rules Concerning Program Flow"** for additional information.

26.7.3 Write Protection of the Boot Segment

The Boot Segment can be write-protected by programming configuration bit BWRP (FBS<0>):

1 = Boot segment can be written

0 = Boot segment is write-protected

When write-protected, row erase and programming operations targeting the Boot Segment of program Flash are disabled. Setting the WR bit within the NVMCON Special Function Register will not start an operation. Erase operations that erase the entire Boot Segment are allowed. However, the Secure and General Segments are also erased.

26.7.4 Allocating Boot Segment Data EEPROM

On devices that contain data EEPROM, the Boot Segment can allocate a portion of the data EEPROM memory for exclusive access by code executing within the Boot Segment. This protects the data integrity of the algorithms executing within the Boot Segment. If a Boot Segment is not allocated, BSS<2:0> = x11 (FBS<3:1>), then a data EEPROM segment cannot be allocated.

The existence and size of the Boot Segment data EEPROM are determined by configuration bit EBS (FBS<8>). One of the options is to exclude Boot Segment data EEPROM, which is the default option on an erased, non-programmed device. The Boot Segment data EEPROM is located at the end of the code space and starts at an address specified by the EBS bit (see **Section 26.4 "Data EEPROM Organization"**). For device-specific information, see Table 26-5, Table 26-6 and Table 26-7.

Table 26-13 is an example of Boot Segment data EEPROM size options. The start addresses listed are typical. Refer to the device data sheet for specific data EEPROM addresses.

Table 26-13: Boot Segment Data EEPROM Size Options Example

EBS	BS Size	BS Start Address	BS End Address
1	No Boot Segment		
0	Large	0x7FFF00	EOM

Note: EOM refers to the last location of data EEPROM.

26.7.5 Allocating Boot Segment RAM

The Boot Segment can also allocate a portion of the device's data RAM memory for exclusive access by code executing within the Boot Segment. This protects the data integrity of algorithms executing within the Boot Segment.

If a Boot Segment is not allocated, BSS<2:0> = x11 (FBS<3:1>), then a RAM segment cannot be allocated. The existence and size of Boot Segment RAM are determined by the RBS<1:0> (FBS<7:6>) configuration bits.

One of the options is to exclude the Boot Segment RAM, which is the default option on an erased, non-programmed device.

The Boot Segment RAM is located at the end of data RAM, as shown in Figure 26-5. The Boot Segment RAM starts at an address specified by the RBS<1:0> bits.

Figure 26-5: Secure Segment Data RAM Allocation

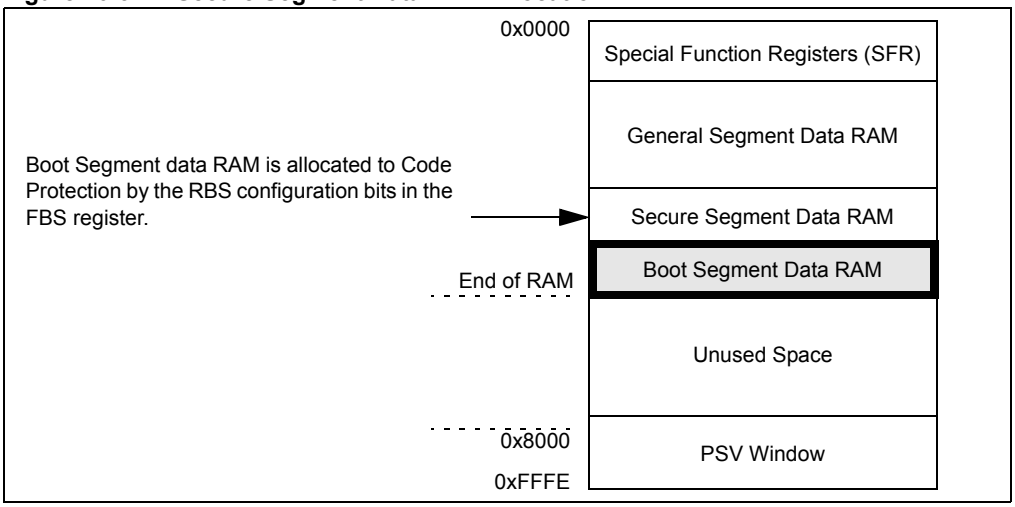


Table 26-14 shows an example of Boot Segment RAM size options on dsPIC30F devices. The start addresses listed are typical. Refer to the device data sheet for specific addresses.

Table 26-14: Boot Segment RAM Size Options Example

RBS<1:0>	BS Size/Bytes	BS Start Address	BS End Address
11	No Boot Segment		
10	Small/128	EOM-0x007F	EOM
01	Medium/256	EOM-0x00FF	EOM
00	Large/512 or Large/1024	EOM-0x01FF EOM-0x03FF	EOM EOM

Note: EOM refers to the last location of data RAM.

26.7.6 Run Time Release of Boot Segment RAM

When an algorithm within the Boot Segment completes its task and prepares to return execution to code within a lower priority segment, it can be helpful to release some of the RAM allocated to the Boot Segment. The Boot Segment RAM control Special Function Register (SFR) contains the RL_BSR (BSRAM<0>) bit (see Register 26-2). When this bit is set, the system releases a portion of the BS RAM to the next lower priority segment. Table 26-15 is an example of RAM mapped with RL_BSR = '0' and RL_BSR = '1'.

Table 26-15: Boot Segment RAM Release

RBS<1:0>	BS Size when:	
	RL_BSR = 0	RL_BSR = 1
11	No Boot Segment	
10	Small	No Boot Segment
01	Medium	Small
00	Large	Medium

26.7.7 Releasing the Secure RAM for General Use

At any time during operation, the secure code segment can release some of its allocation of secure RAM for general use. For example, the minimum allocation can be reserved for the BS to store sensitive volatile variables during General Segment code run time. Then when the code branches to the BS segment to execute an algorithm, the BS code can clear the RL_BSR bit to secure the maximum allocation of secure RAM for its use. After the BS code execution completes, it can reset the RL_BSR bit to again minimize the allocated secure RAM.

Both the Boot and Secure segments have associated BSRAM and SSRAM registers, which contain the RL_BSR and RL_SSR bits, respectively. Only the BS segment has write access to the BSRAM register, and only the Secure Segment has write access to the SSRAM register.

Note: On any reset, the maximum allocations are in a secure state, because the RL_BSR and RL_SSR bits are Reset.

The RAM security bits determine whether RAM is secured or not.

- If RSS<1:0> = 11 (FSS<13:12>), no boot RAM is allocated and the RL_SSR bit is ignored.
- If RBS<1:0> = 11 (FBS<13:12>), no boot RAM is allocated and the RL_BSR bit is ignored.

Register 26-2: BSRAM Register

Middle Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
U-0	U-0	U-0	U-0	U-0	R-0	R-0	R/W-0
—	—	—	—	—	IW_BSR	IR_BSR	RL_BSR
bit 7							bit 0

bit 15-3 **Unimplemented:** Read as '0'

bit 2 **IW_BSR** (Read-Only Status bit)
0 = No illegal write has been attempted since this register was last read
1 = At least one illegal write has been attempted since the last read of this register
IW_BSR bit is cleared on any Reset. It is also cleared after the BSRAM register is read while executing in BS.

bit 1 **IR_BSR** (Read-Only Status bit)
0 = No invalid read of the protected BS RAM section has occurred since this register was last read
1 = At least one invalid read has occurred since last read of this register
IR_BSR bit is cleared on any Reset. It is also cleared after the BSRAM register is read while executing in BS.

bit 0 **RL_BSR**
0 = BSRAM is held secure for BS only
1 = BS has released the secure RAM for general use. All but the highest 128 bytes are released.
RL_BSR bit is cleared to zero on any Reset.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

26.8 The Secure Segment

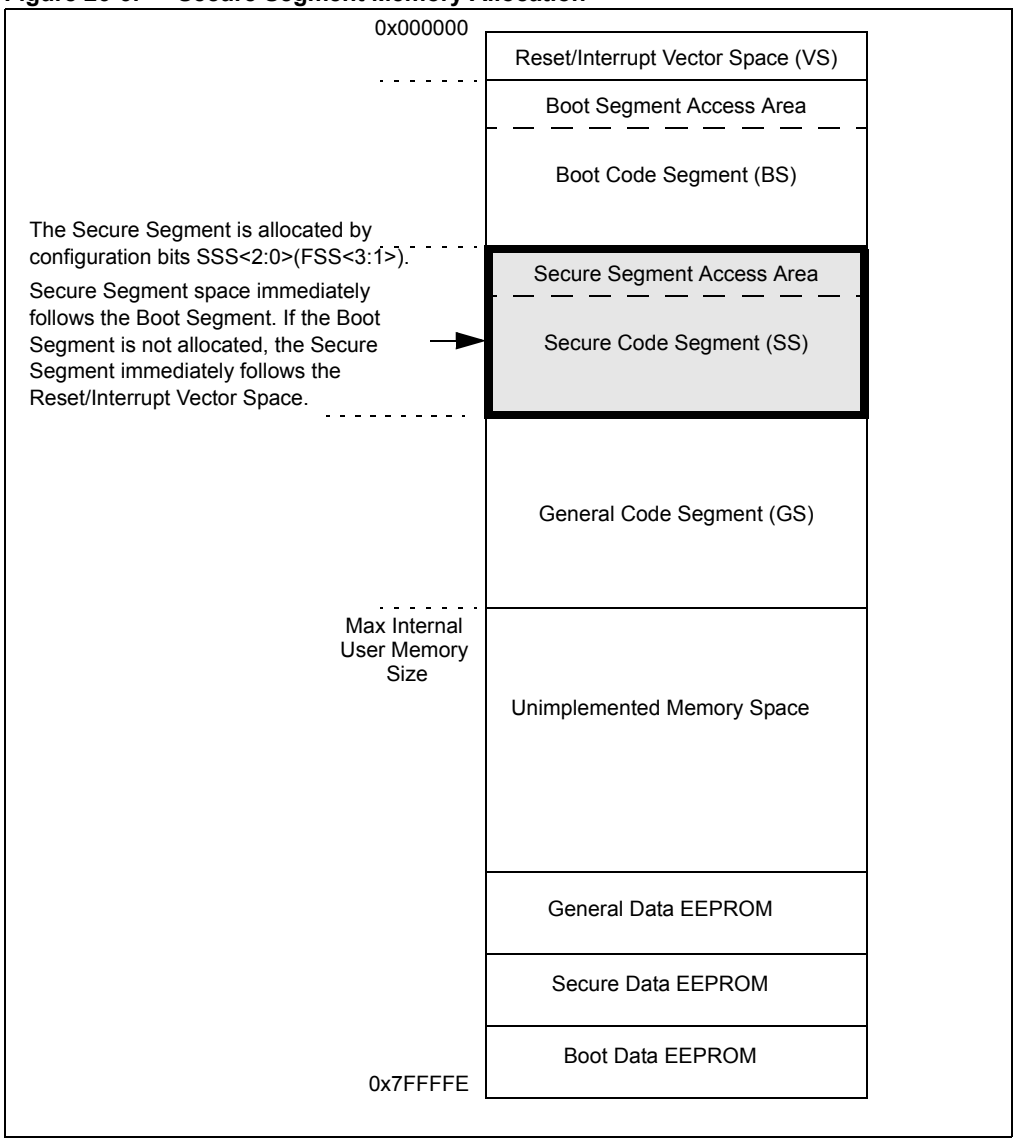
The Secure Segment has the second highest privilege and is ideal for storing proprietary algorithm routines. Access to the Secure Segment from lower priority segments is limited to “calls” to the Secure Segment.

26.8.1 Allocating the Secure Segment

The Secure Segment is allocated by the configuration bits SSS<2:0> (FSS<3:1>).

The Secure Segment begins immediately following the Boot Segment, as shown in Figure 26-6. If there is no Boot Segment, the Secure Segment starts at the end of the Reset/Interrupt Vector Space. The default is to exclude a Secure Segment.

Figure 26-6: Secure Segment Memory Allocation



The Secure Segment continues to an address specified by the SSS<2:0> bits. Table 26-16 shows an example of Secure Segment options for dsPIC30F devices. The end addresses listed in these tables are typical. Refer to the device data sheet for specific addresses for a given device.

Table 26-16: Secure Segment Size Options Summary Example

SSS<2:0>	Security Level	SS Size	SS Start Address	SS End Address
x11	No Secure Program Flash Segment			
110	Standard	Small	E.O. BS + 1	0x001FFE
010	High	Small	E.O. BS + 1	0x001FFE
101	Standard	Medium	E.O. BS + 1	0x003FFE
001	High	Medium	E.O. BS + 1	0x003FFE
100	Standard	Large	E.O. BS + 1	0x007FFE
000	High	Large	E.O. BS + 1	0x007FFE

Note: E.O. BS refers to the last location of the Boot Segment.

Register 26-3: FSS: Secure Segment Configuration Register

Upper Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 23							bit 16

Middle Byte:							
U-0	U-0	R/P	R/P	U-0	U-0	R/P	R/P
—	—	RSS1	RSS0	—	—	ESS1	ESS0
bit 15							bit 8

Lower Byte:							
U-0	U-0	U-0	U-0	R/P	R/P	R/P	R/P
—	—	—	—	SSS2	SSS1	SSS0	SWRP
bit 7							bit 0

bit 23-14 **Unimplemented:** Read as '0'

bit 13-12 **RSS<1:0>:** Secure Segment RAM Code Protection bits(1)
Device Secure Segment size depends on RSS<1:0>, RBS<1:0> (FBS<13:12>), RL_SSR (SSRAM<0>) and RL_BSR (BSRAM<0>) bit settings

bit 11-10 **Unimplemented:** Read as '0'

bit 9-8 **ESS<1:0>:** Secure Segment Data EEPROM Code Protection bits
11 = No Secure Data EEPROM Segment
1x = Reserved
0x = Reserved
00 = Secure Data EEPROM Segment is 2048 bytes

bit 7-4 **Unimplemented:** Read as '0'

bit 3-1 **SSS<2:0>:** Secure Segment Program Flash Code Protection bits(2)
x11 = No Secure program Flash segment
110 = Standard security, small Secure Segment
010 = High security, small Secure Segment
101 = Standard security, medium Secure Segment
001 = High security, medium Secure Segment
100 = Standard security, large Secure Segment
000 = High security, large Secure Segment

bit 0 **SWRP:** Secure Segment Program Flash Write Protection bit
1 = Secure segment can be written
0 = Secure segment is write-protected

Note 1: Not all devices have Secure RAM and Secure Data EEPROM protection. For specific device information, see Table 26-2, Table 26-3, Table 26-4, Table 26-5, Table 26-6, and Table 26-7.

2: The exact definitions of Small, Medium, and Large Secure Segment vary from one device to another. For device specific information, see Table 26-8, Table 26-9, Table 26-10, Table 26-11, and Table 26-12.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

26.8.2 Selecting the Security Level of the Secure Segment

The security level of the secure code segment is determined by the configuration bit SSS2 (FSS<3>):

1 = Standard security

0 = High security

When the Secure Segment is configured for high security, the number of access methods is more limited than with standard security. The differences are noted in the following paragraphs.

26.8.3 Write Protection of the Secure Segment

The Secure Segment can be write-protected by programming the SWRP (FSS<0>) configuration bit:

1 = Secure segment can be written

0 = Secure segment is write-protected

When write-protected, row erase and programming operations targeting the Secure Segment of program Flash are disabled. Setting the WR bit within the NVMCON Special Function Register will not start an operation. Erase operations that erase the entire Secure Segment are allowed. However, the General Segment is also erased.

26.8.4 Allocating Secure Segment Data EEPROM

On devices that contain data EEPROM, the Secure Segment can allocate a portion of the data EEPROM memory of the device for exclusive access by code executing within the Secure Segment. This protects the data integrity of the algorithms executing within the Secure Segment. If no Secure Segment is allocated, SSS<2:0> = x11 (FSS<3:1>), then a data EEPROM segment cannot be allocated.

The existence and size of the Secure Segment data EEPROM are configured by the ESS<1:0> (FSS<9:8>) configuration bits. Figure 26-7 illustrates the data EEPROM memory space. The Secure Segment data EEPROM ends at the last location before the Boot Segment data EEPROM. The Secure Segment data EEPROM starts at an address specified by the ESS<1:0> bits. See Table 26-5, Table 26-6 and Table 26-7 for device specific information.

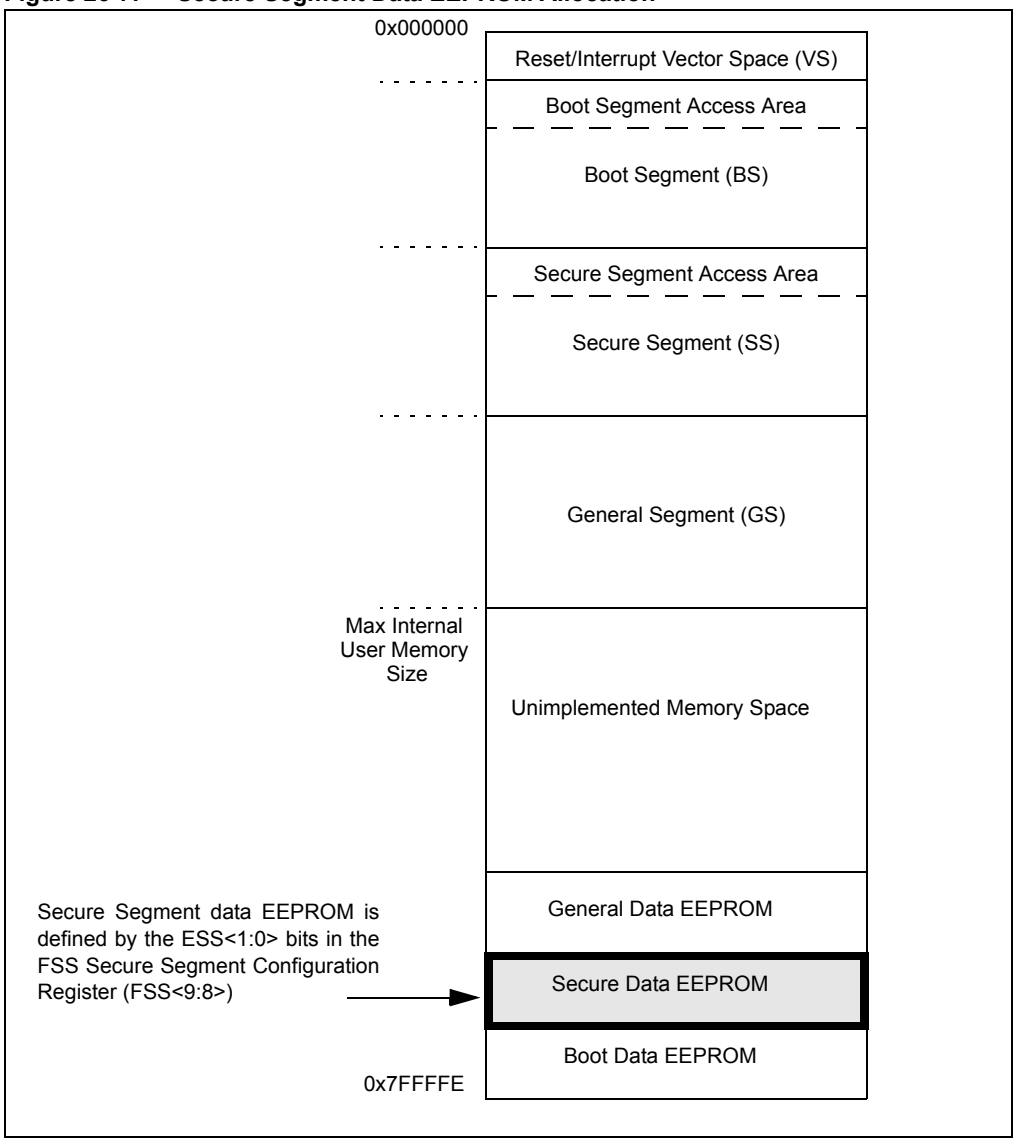
Table 26-17 is an example of Secure Segment data EEPROM size options. The start addresses listed are typical. Refer to the device data sheet for specific data EEPROM addresses.

Table 26-17: Secure Segment data EEPROM Size Options Example

ESS<1:0>	SS Size	SS Start Address	SS End Address
11	No Secure Segment		
10	Small	0x7FFF00	S.O. BS – 1
01	Medium	0x7FFE00	S.O. BS – 1
00	Large	0x7FF800	S.O. BS – 1

Note: S.O. BS refers to the first location of Boot Segment data EEPROM.

Figure 26-7: Secure Segment Data EEPROM Allocation



26.8.5 Allocating Secure Segment RAM

The Secure Segment can also allocate a portion of the data RAM for code protection. However, if a Secure Segment is not allocated, SSS<2:0> = x11 (FSS<3:1>), then a RAM segment cannot be allocated. The existence and size of Secure Segment RAM are determined by the RSS<1:0> (FSS<13:12>) configuration bits.

One of the options is to exclude the Secure Segment RAM, which is the default option on an erased, non-programmed device.

The Secure Segment RAM ends at the last location before the Boot Segment RAM. The Secure Segment RAM starts at an address specified by the RSS<1:0> bits.

Figure 26-8: Secure Segment Data RAM Allocation

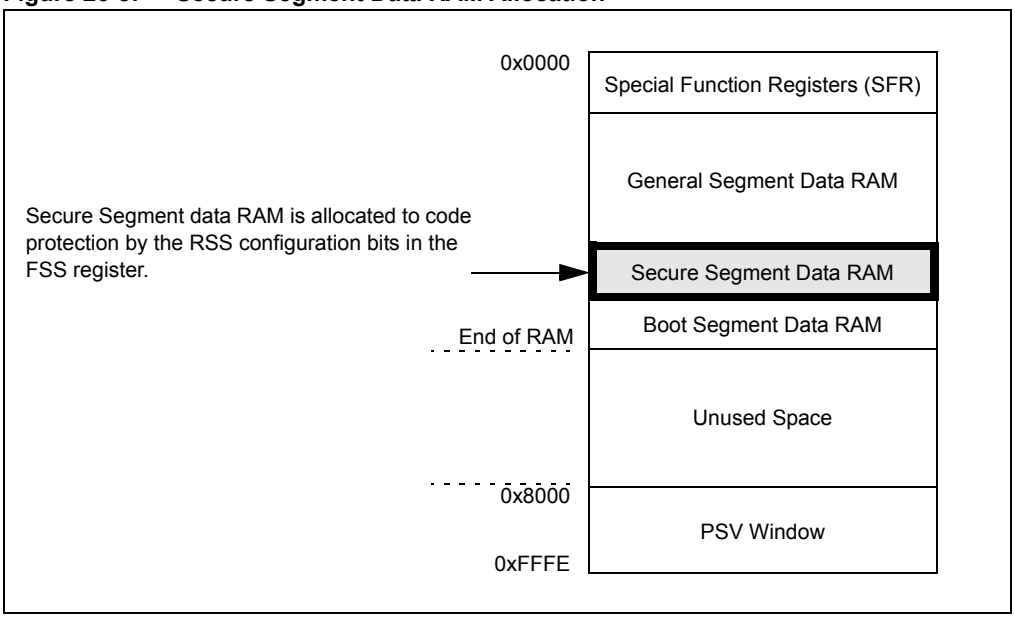


Table 26-18 shows an example of Secure Segment RAM allocations. The start addresses listed are typical. Refer to the device data sheet for specific addresses for a given device.

Table 26-18: Secure Segment RAM Size Options Example

RSS<1:0>	SS Size	SS Start Address	SS End Address
11	No Secure Segment		
10	Small	See Tables 26-2, 26-3, and 26-4	S.O. BS – 1
01	Medium	See Tables 26-2, 26-3, and 26-4	S.O. BS – 1
00	Large	See Tables 26-2, 26-3, and 26-4	S.O. BS – 1

Note: S.O. BS refers to the first location of basic segment data RAM.

26.8.6 Run Time Release of Secure Segment RAM

Like the Boot Segment, the Secure Segment can allocate and release RAM. The Secure Segment RAM control Special Function Register (SFR) contains the RL_SSR (SSRAM<0>) bit (see Register 26-4). When this bit is set, the system releases a portion of the Secure Segment RAM to the next lower priority segment. Table 26-19 shows an example of RAM mapped with RL_SSR = '0' and RL_SSR = '1'.

Table 26-19: Secure Segment RAM Release

RSS<1:0>	Secure Segment Size When:	
	RL_SSR = 0	RL_SSR = 1
11	No Secure Segment	
10	Small	No Secure Segment
01	Medium	Small
00	Large	Medium

26.8.7 Releasing Secure Segment RAM for General Use

The secure code segments can release some of their allocation of secure RAM for general use at any time during operation. For example, the minimum allocation can be reserved for the Secure Segment to store sensitive volatile variables during General Segment code run time. When the code branches to the Secure Segment to execute an algorithm, the Secure Segment code can clear the RL_SSR bit to secure the maximum allocation of secure RAM for its use. After the Secure Segment code execution completes, it can set the RL_SSR bit to again minimize the allocated secure RAM.

Both the Boot and Secure Segments have associated BSRAM and SSRAM registers, which contain the RL_BSR and RL_SSR bits, respectively. Only the Boot Segment has write access to the BSRAM register, and only the Secure Segment has write access to the SSRAM register.

Note:	On any Reset, the maximum allocations are in a secure state, because the RL_SSR bit is Reset.

The RAM security bits determine if RAM is secured or not. If RSS<1:0> = 11 (FSS<13:12>), no secure RAM is allocated and the RL_SSR bit is ignored.

Register 26-4: SSRAM Register

Middle Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	R-0	R-0
—	—	—	—	—	IW_SSR	IR_SSR	RL_SSR
bit 7							bit 0

bit 15-3 **Unimplemented:** Read as '0'

bit 2 **IW_SSR** (Read-Only Status bit)
0 = No illegal write of protected SSRAM has been attempted since this register was last read
1 = At least one illegal write has been attempted since the last read of this register
IW_SSR bit is cleared on any Reset. It is also cleared after SSRAM register is read while executing in the Secure Segment.

bit 1 **IR_SSR** (Read-Only Status bit)
0 = No invalid read of protected SSRAM has occurred since this register was last read
1 = At least one invalid read has occurred since the last read of this register
IR_SSR bit is cleared on any Reset. It is also cleared after the SSRAM register is read while executing in the Secure Segment.

bit 0 **RL_SSR**
0 = SSRAM is held secure for Secure Segment use only
1 = Secure Segment has released the secure RAM for general use. All but the highest 128 bytes are released.
RL_SSR bit is cleared to zero on any Reset.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

26.9 The General Segment (GS)

The General Segment has the lowest security privilege level. The General Segment is intended to contain the majority of the application code. Its size is essentially the on-chip memory minus the Boot and Secure Segments. If there are no Boot or Secure Segments, the General Segment uses all of the on-chip memory.

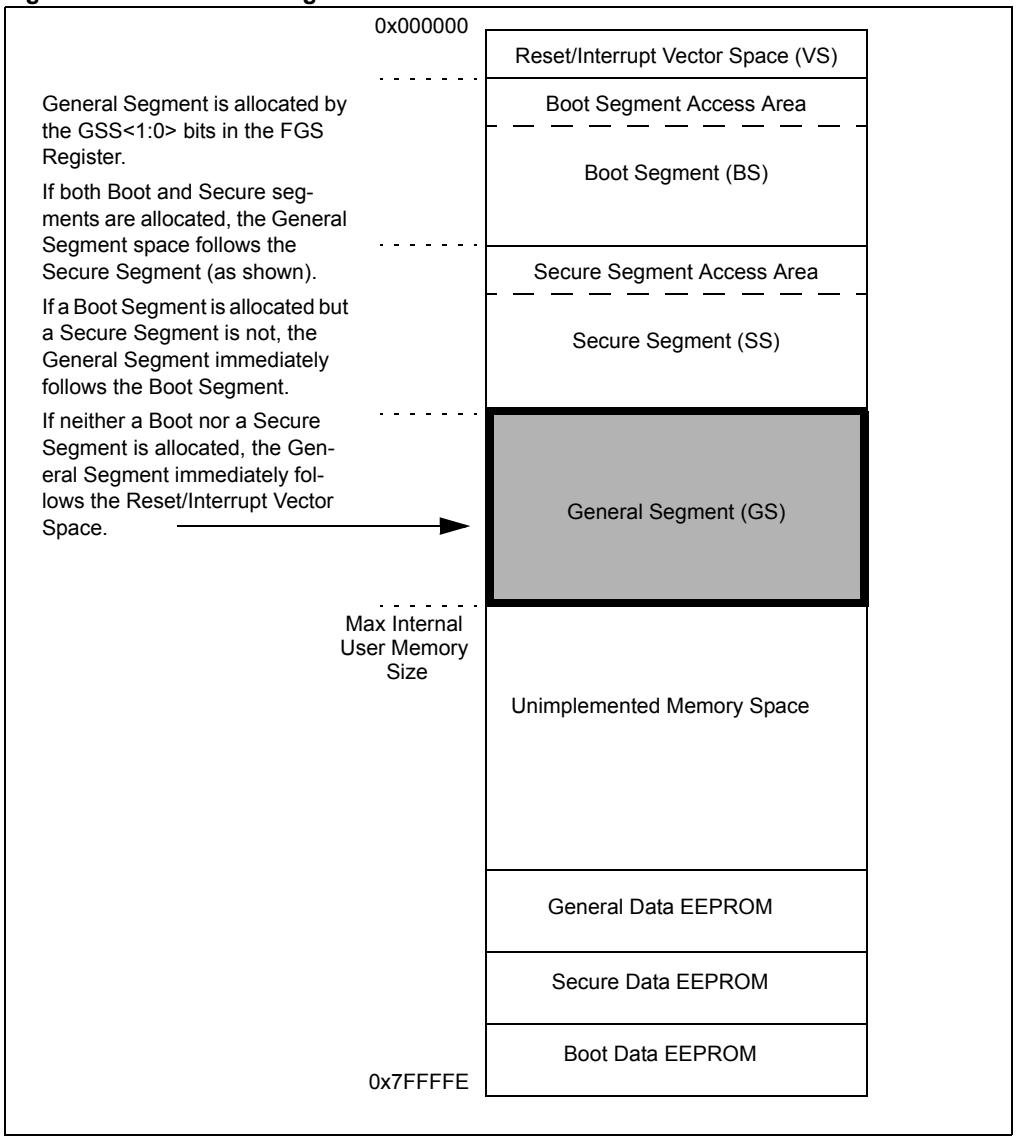
26.9.1 Allocating the General Segment

The General Segment always exists, regardless of whether Boot and Secure Segments have been allocated. It is specified by the GSS<1:0> configuration bits in the FGS register. The location of the General Segment depends on the existence of the Boot and Secure Segments.

If both Boot and Secure Segments are allocated, the General Segment immediately follows the Secure Segment, as shown in Figure 26-9. If a Boot Segment is allocated, but a Secure Segment is not, the General Segment immediately follows the Boot Segment. If neither a Boot Segment nor a Secure Segment is allocated, the General Segment immediately follows the Reset/Interrupt Vector Space (VS).

The device default is to exclude Boot and Secure Segments. By default, the entire program memory is allocated as General Segment.

Figure 26-9: General Segment Allocation



Register 26-5: FGS: General Segment Configuration Register for Devices with Advanced Security

Upper Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 23							bit 16

Middle Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
U-0	U-0	U-0	U-0	U-0	R/P	R/P	R/P
—	—	—	—	—	GSS1	GSS0	GWRP
bit 7							bit 0

bit 23-3 **Unimplemented:** Read as '0'

bit 2-1 **GSS<1:0>:** General Segment Program Flash Code Protection bits
11 = No Protection
10 = Standard security; general program Flash segment starts at the end of SS and ends at EOM
0X = High security; general program Flash segment starts at the end of SS and ends at EOM

bit 0 **GWRP:** General Segment Program Flash Write Protection bit
1 = General segment can be written
0 = General segment is write-protected

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Register 26-6: FGS: General Segment Configuration Register for Devices with Basic or Intermediate Security

Upper Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 23							bit 16

Middle Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	R/P	R/P
—	—	—	—	—	—	GCP	GWRP
bit 7							bit 0

bit 23-2 **Unimplemented:** Read as '0'

bit 1 **GCP:** General Segment Program Flash Code Protection bit
1 = General Segment is not code-protected
0 = General Segment is code-protected

bit 0 **GWRP:** General Segment Program Flash Write Protection bit
1 = General Segment can be written
0 = General Segment is write-protected

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

26.9.2 Selecting the Security Level of the General Segment

Depending on the device, there are either two or three levels of security to choose from for the General Segment. Refer to the specific device data sheet to determine how many options are available.

On devices with no high security level support, configuration bit GCP (FGS<1>) determines the level of protection for this segment:

1 = No protection

0 = Standard security

On all other devices, two configuration bits, GSS<1:0> (FGS<2:1>), determine the level of protection for this segment:

11 = No protection

10 = Standard security

0X = High security

26.9.3 Write Protection of the General Segment

The General Segment can be write-protected by programming the GWRP (FGS<0>) configuration bit, similar to write protecting the Boot Segment.

1 = General code segment can be written

0 = General code segment is write-protected

26.10 The Reset, Trap and ISR Vector Space (VS)

The first 128 instruction words are reserved for the RESET Instruction, trap and interrupt vectors.

Protection of this segment depends on the state of the BSS<2:0> (FBS<3:1>) and GSS<1:0> (FGS<2:1>) or GCP (FGS<1>) code protection bits. If a Boot Segment is allocated, the Vector Space protection is the same as the Boot Segment. In other words, if a Boot Segment is defined, erase and programming operations of the Vector Space can only be performed via Boot Segment code. If a Boot Segment is not allocated, the Vector Space protection is the same as the General Segment and erase and programming operations of the Vector Space can be performed via General Segment code.

A write to this segment is enabled or disabled by the BWRP bit if a Boot Segment is allocated or by the GWRP bit if a Boot Segment is excluded.

26.11 Definition of Security Privileges

It is important to understand the relative privilege levels of the three code protection segments. Operations can be described as being relative to higher or lower privilege segments. The Boot Segment has the highest privilege level and can directly access code in the lower segments. The Secure Segment can directly access code in the General Segment but can only issue calls to code in the Boot Segment. The General Segment can only access code from either of the higher segments by issuing calls.

Rules governing access privileges are discussed in sections **26.12 "Rules Concerning Program Flow"** through **26.16.1 "Rules for Programming Devices in RTSP"**. A summary overview of these rules during normal run-time operation is presented in Table 26-20.

Table 26-20: Privileged Operations Rules Summary

Target Segment		General Segment						Secure Segment				Boot Segment				IVT & AIVT					
Protection Level		None		Standard		High		Standard		High		Standard		High		None		Standard		High	
Write-Protected		No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Requested Operation (Yes/No)																					
PC Rollover into Target Segment		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	N/A	N/A	N/A	N/A **(Note 3)**					
PFC from reset vector instruction to Target Segment **(Note 5)**		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Note 2**	**Note 2**	Yes	Yes	**Note 2**	**Note 2**	**Note 4**					
VFC (Vector Flow Change) to Target Segment **(Note 5)**		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Note 2**	**Note 2**	Yes	Yes	**Note 2**	**Note 2**	**Note 4**					
PFC from BS to Target Segment **(Note 1)**		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Note 4**					
PFC from SS to Target Segment **(Note 1)**		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Note 2**	**Note 2**	**Note 4**					
PFC from GS to Target Segment **(Note 1)**		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Note 2**	**Note 2**	Yes	Yes	**Note 2**	**Note 2**	**Note 4**					
R/W of Target Segment RAM while executing from: Note: Stack assumed to be in GS RAM space, access needed	BS	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes	N/A					
	SS	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No						
	GS	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No						
Table Read/PSV of Target Segment Program Flash while executing from: **(Note 7)**	BS	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	SS	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	GS	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Table Read/PSV of Target Segment Data EEPROM space while executing from: **(Note 7)**	BS	Yes	Yes	Yes	Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	N/A					
	SS	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No	No	No	No						
	GS	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No						
Table Write of Target Segment (load write latches)		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Prog/Erase row of Target Segment program Flash while executing from:	BS	Yes		Yes				Yes				Yes		Yes		Yes		Yes			
	SS	Yes		Yes				Yes		Yes						**Note 6**		**Note 6**			
	GS	Yes		Yes		Yes										**Note 6**		**Note 6**			
Prog/Erase word or row of Target Segment Data EEPROM while executing from:	BS	Yes	Yes	Yes	Yes							Yes	Yes	Yes	Yes	N/A					
	SS	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes										
	GS	Yes	Yes	Yes	Yes	Yes	Yes														
Erase Target Segment data flash (Set WR bit with NVMCON = 1100110,1010110,1000110) while executing from:	BS	Yes	Yes	Yes	Yes							Yes	Yes	Yes	Yes	N/A					
	SS	Yes	Yes	Yes	Yes			Yes	Yes	Yes	Yes										
	GS	Yes	Yes	Yes	Yes	Yes	Yes														

Note 1: PFC (Program Flow Change) occurs when the PC is loaded with a new value instead of the normal automatic increment. This happens during a JUMP, CALL, RETURN, RETFIE, Computed Jump, etc.
2: PFC is only allowed to the first 32 instruction locations of the segment.
3: Since execution is not permitted in the VS segment, this condition is not possible.
4: A PFC operation (i.e., branch, call etc.) into the IVT and AIVT segments is possible. But as soon as execution is attempted out of this segment, an illegal address trap will result (unless pointed to the Reset vector at address 0x000000).
5: VFC (Vector Flow Change) occurs when the PC is loaded with an Interrupt or trap vector address.
6: Operation allowed if BSS<2:0> = X11 (no Boot Segment defined).
7: TBLRD or DS read will execute but return all 0s if not allowed.

Table 26-20: Privileged Operations Rules Summary (Continued)

Target Segment	General Segment						Secure Segment				Boot Segment				IVT & AIVT					
Protection Level	None		Standard		High		Standard		High		Standard		High		None		Standard		High	
Write-Protected	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Erase All (Set WR bit with NVMCON = 1XX1111)	Command not valid in RTSP mode																			
Erase BS Segment/code-protect (Set WR bit with NVMCON = 1101110)	Erase GS/SS/BS/VS segments and GS/SS/BS code protection fuses																			
Erase SS Segment/code-protect (Set WR bit with NVMCON = 1011110)	Erase GS/SS segments and GS/SS code protection fuses Erase VS if no Boot Segment defined																			
Erase GS Segment/code-protect (Set WR bit with NVMCON = 1001110)	Erase GS segment and GS code protection fuses Erase VS if no Boot Segment defined																			
Erase GS Segment (Set WR bit with NVMCON = 1000010)	Erase GS Segment Only																			
Prog config register (Set WR bit with NVMCON = 0XX1000)	Yes																			

Note 1: PFC (Program Flow Change) occurs when the PC is loaded with a new value instead of the normal automatic increment. This happens during a JUMP, CALL, RETURN, RETFIE, Computed Jump, etc.

2: PFC is only allowed to the first 32 instruction locations of the segment.

3: Since execution is not permitted in the VS segment, this condition is not possible.

4: A PFC operation (i.e., branch, call etc.) into the IVT and AIVT segments is possible. But as soon as execution is attempted out of this segment, an illegal address trap will result (unless pointed to the Reset vector at address 0x000000).

5: VFC (Vector Flow Change) occurs when the PC is loaded with an Interrupt or trap vector address.

6: Operation allowed if BSS<2:0> = X11 (no Boot Segment defined).

7: TBLRD or DS read will execute but return all 0s if not allowed.

26.12 Rules Concerning Program Flow

Program flow refers to the execution sequence of program instructions in program memory. Normally, instructions are executed sequentially as the Program Counter (PC) increments. When code protection is implemented, program flow conforms to privilege level. That is, a program executing from code-protected memory can flow from a higher security segment to a lower segment, but not vice versa. For example, a program executing from the Secure Segment can flow into the General Segment but not into the Boot Segment.

Program Flow Change (PFC) occurs when the Program Counter is reloaded as a result of a Call, Jump, Computed Jump, Return, Return from Subroutine, or other form of branch instruction. A Program Flow Change allows the program flow to follow an alternate path. A normal Program Flow Change only allows the program to branch within the same segment. A Restricted Program Flow Change allows the program to branch to a special Segment Access Area of a higher security segment.

Vector Flow Change (VFC) occurs when the Program Counter is reloaded with an Interrupt or Trap vector.

Jumping into secure code at unintended locations can expose code to algorithm detection. Therefore, PFC and VFC operations are restricted if they violate the privilege hierarchy.

Figure 26-10: Program Flow Rules

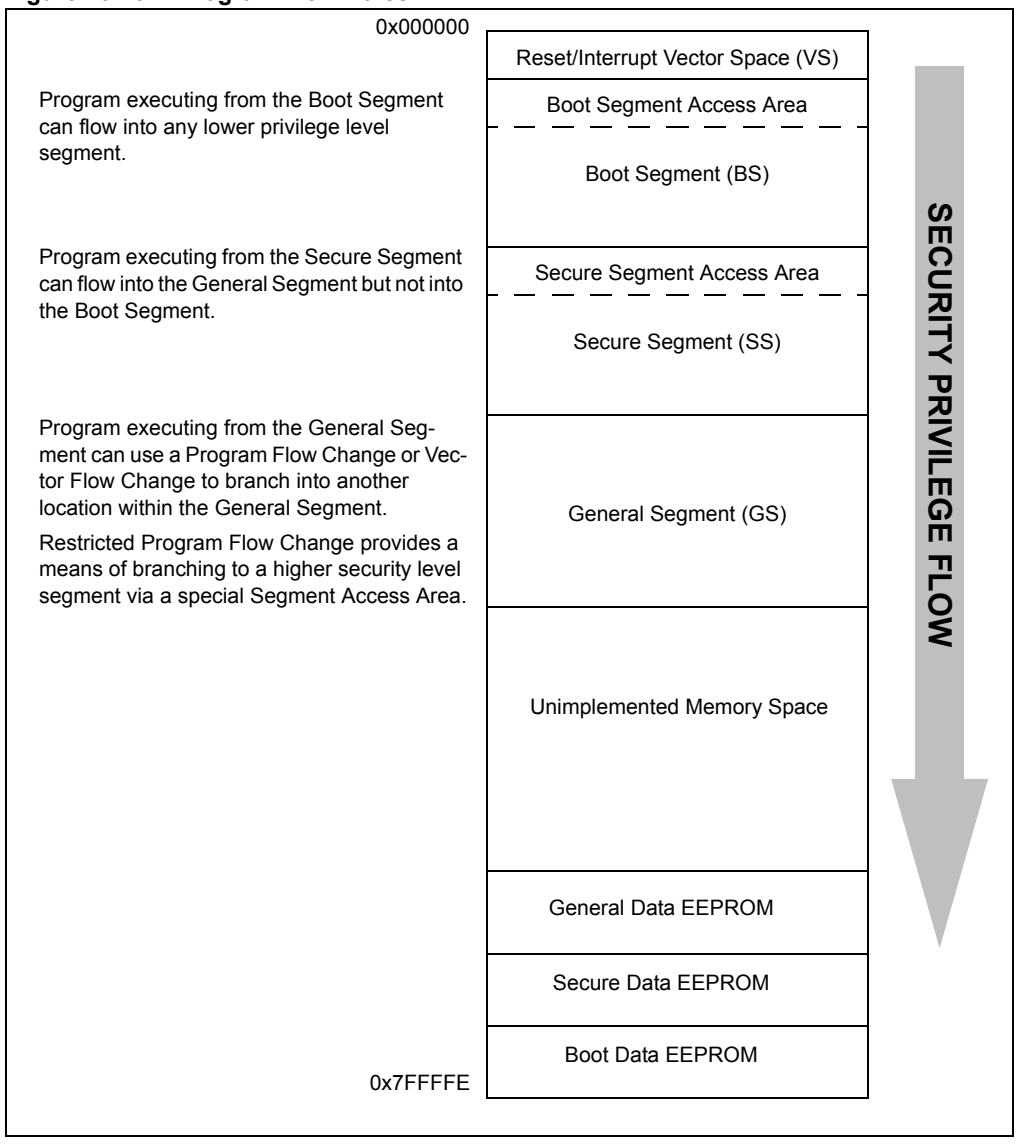


Table 26-21 presents an overview matrix of possible operations between program memory segments. Blank cells indicate conditions where read/write/erase and PFC operations can not be performed. When the Boot Segment or Secure Segment is implemented with high security level, a PFC from a lower privilege level must be restricted to the segment access area. Any PFC attempt outside the segment access area results in a security Reset (see **Section 26.12.3 "Program Flow Errors"**).

Table 26-21: Possible Operations Between Program Memory Segments

Code Executed From:		Operation To:						
		Boot Segment Security Level		Secure Segment Security Level		General Segment Security Level		
		Standard	High	Standard	High	Standard	High	None
BS	Standard	R, P, PFC		R, P, PFC	PFC*	R, P, PFC	PFC	R, P, PFC
	High		R, P, PFC	R, P, PFC	PFC*	R, P, PFC	PFC	R, P, PFC
SS	Standard	PFC	PFC*	R, P, PFC		R, P, PFC	PFC	R, P, PFC
	High	PFC	PFC*		R, P, PFC	R, P, PFC	PFC	R, P, PFC
GS	Standard	PFC	PFC*	PFC	PFC*	R, P, PFC		
	High	PFC	PFC*	PFC	PFC*		R, P, PFC	
	None	PFC	PFC*	PFC	PFC*			R, P, PFC

Legend: R – Read
P – Program (write)/Erase
PFC – Program Flow Change allowed to anywhere in the segment
PFC* – Restricted Program Flow Changes (can branch to segment access areas only)

26.12.1 Flow Changes

Program Flow Changes within a segment are unrestricted. Generally, PFC and VFC changes from one segment to another segment are not restricted, except as follows.

To ensure the integrity of the operations of code within the Boot and Secure segments, the user must restrict program flow options to those segments. Program flow can be limited to only allow the segment access areas to be a branch target. The segment access areas are the first 32 instruction locations of the Boot Segment or Secure Segment code space.

If the security level of the Boot Segment or Secure Segment is high and a PFC originates from a lower priority segment, the target of the PFC must be within the access area.

If the security level of the Boot Segment or Secure Segment is high and a VFC occurs, the target of the VFC must be within the segment access area.

The owners of the code within the Boot Segment or Secure Segment code space can ensure that the access area contains branches to specified sections of the application code that are verified to not expose the algorithm.

26.12.2 Reset Instruction

A typical device reset will reset the PC to 0x000000, then begin execution at that location. There must be a branch instruction at locations 0x000000-0x000002 that will branch to the beginning of the code.

The instruction at the reset vector can branch to any location unless the Boot Segment or Secure Segment is in high security; in that case, the target must be to the specific segment access area. Typically, the target of the `RESET` instruction will be the General Segment code space.

26.12.3 Program Flow Errors

If a PFC or VFC targets a restricted location, that operation will cause a security reset. The device will Reset and set the IOPUWR (RCON<14>) status bit, indicating an illegal operation.

In addition to this specific security reset, there are also program flow checks that are built into all devices.

If a program flow or vector flow change targets unimplemented program memory or data EEPROM space, an illegal address trap will occur.

Code execution from the vector segment (other than the instruction at the Reset location), is not allowed. If code execution is attempted from the vector segment, it will result in an Address Error Trap.

26.12.4 Re-targeting Reset After Boot

The Reset operation is independent of the segment the device is operating in when the reset occurs. To prevent code probing, the Reset vector is protected when a Boot Segment is allocated. If a Boot Segment exists, the Vector Space (including the Reset vector) is part of the Boot Segment and restricted by Boot Segment rules. If a Boot Segment does not exist, the Vector Space (including the Reset vector) is part of the General Segment and can be modified by the General Segment.

For example, assume that a part has a boot sector that contains a boot loader. At Reset, the device will initially Reset to a location within the boot loader. The boot loader will run and load user code into the General Segment. When this operation is complete, the boot loader can rewrite the Reset vector instruction to point to the user code. At the next Reset, the Reset will go to the user code. However, the user code cannot then rewrite the Reset vector instruction.

26.13 Rules Concerning Interrupts

26.13.1 Interrupts and Traps In Secure Modes

Interrupt handling is restricted for the following reasons:

1. A Return from Interrupt is one way to corrupt intended program flow (by changing the return address in the stack).
2. The secure code should have the opportunity to clear sensitive information before responding to an interrupt.

26.13.1.1 BS and SS Interrupt Vectors

If an interrupt occurs while the program is running in the Boot or Secure Segment, the processor obtains the interrupt vector from the special Boot Segment interrupt vector location at (BS + 0x20) or the special Secure Segment interrupt vector location at (SS + 0x20).

Note: There are two special interrupt vectors, one within the Boot Segment and the other within the Secure Segment. Interrupts that occur while code executes in one of these segments will cause the processor to vector to the special interrupt vector for that segment. Users may employ a special Interrupt Service Routine (ISR) within the protected segment to hide critical data and then manually vector to the real ISR by reading the INTTREG SFR.

26.13.1.2 Interrupt and Trap Handling Sequence

The sequence for handling interrupts and traps that occur while code is executing in a Secure Segment is as follows:

1. Interrupt or trap occurs while code is executing in a Secure Segment (for example, in the Boot Segment).
2. Return address is pushed on the Stack.
3. The contents of location (BS + 0x20) are loaded into the Program Counter instead of the usual interrupt vector.
4. A special ISR is executed at the address pointed to by (BS + 0x20).
5. Sensitive information from the W registers is stored to the secure RAM area, whether this be the Boot Segment or Secure Segment RAM space.
6. Actual return address is retrieved from the Stack and saved in secure RAM.
7. Actual return address is replaced with new return address. For example, BS + 0x30. BS + 0x30 is located in the BS to BS + 0x3E address range. Keep in mind that in this range Program Flow Change is allowed from outside the Boot Segment.
8. The INTTREG SFR is read to determine which interrupt vector to jump to.
9. The interrupt vector is read from the vector table and executes an indirect jump.
10. User's ISR begins execution.
11. User code executes.
12. Return from interrupt (back to location BS + 0x30).
13. Read the actual return address from the secure RAM area.
14. Restore the W registers.
15. Execute an indirect jump to go back to the Boot Segment.

Table 26-22: Vector Operations In Normal User Mode

Vector Operation	Result in Normal User Mode
Hardware Interrupt while in BS	Obtain vector from special BS ISR vector location at BS + 0x20
Hardware Interrupt while in SS	Obtain vector from special SS ISR vector location at SS + 0x20
Hardware Interrupt while in GS	Obtain vector from normal ISR vector location
Software Interrupt and Trap while in BS	Obtain vector from special BS ISR vector location at BS + 0x20
Software Interrupt and Trap while in SS	Obtain vector from special SS ISR vector location at SS + 0x20
Software Interrupt and Trap while in GS	Obtain vector from normal ISR vector location

26.14 Rules for Accessing RAM Data

26.14.1 Using Segment RAM

If the Boot Segment or the Secure Segment has allocated protected RAM space, that RAM is not accessible by code running outside of that segment. For example, code running in the Secure or General Segment cannot access RAM protected by the Boot Segment.

If an instruction contains an unauthorized read of a protected RAM location, the instruction will execute and the read operation will occur; however, the resulting write of the instruction will be disabled. For example, "`mov ssram, w0`" will read from the SSRAM location; however, the write to `w0` will not occur. The results from the ALU are zeroed out and the write does not occur.

An unauthorized read of a protected RAM location within the boot RAM segment will set the IR_BSR (BSRAM<0>) bit. This bit will remain set until a device Reset or until the BSRAM register is read by code executing from within the Boot Segment.

Similarly, an unauthorized read from the secure RAM segment will set the IR_SSR (SSRAM<0>) bit. This bit will remain set until a device Reset or until the SSRAM register is read by code executing from within the Secure Segment.

An unauthorized write to a protected RAM location will cause a write of '`0`' to the protected location.

An unauthorized write of a protected RAM location will set either the IW_BSR (BSRAM<2>) bit or IW_SSR (SSRAM<1>) bit to the "illegal write" status. These bits are cleared by a device Reset or when the BSRAM or SSRAM register is read by code executing from within the Boot or Secure Segments, respectively.

26.14.2 Stack Allocation

The user can allocate the stack space anywhere in the RAM. However, the General Segment RAM is the only area that both the Boot and Secure Segment can access. Therefore, the stack should be allocated in the General Segment RAM area.

If the stack accidentally intrudes into the "BSRAM" or "SSRAM" area, then writes due to push operations and reads due to pop operations will act like unauthorized RAM accesses as described above.

This can and should be prevented by using the Stack Pointer Limit register (SPLIM) and setting the contents of the SPLIM to an appropriate address.

26.14.3 Register Dumping Protection

The device will initialize W registers on all resets. Data RAM is not initialized and resets can leave valid data in data RAM. The security of the RAM contents must be maintained.

26.15 Rules for Reading Data EEPROM

26.15.1 Using Segment Data EEPROM

If the Boot Segment or the Secure Segment has allocated protected data EEPROM, that data EEPROM is not accessible by code running outside of the respective segment.

An unauthorized read of a protected data EEPROM location will read as '0'. This includes table reads or PSV reads of the data EEPROM area.

26.15.2 Rules for Reading Program Flash

TBLRD and instructions that address program memory through PSV addressing can be restricted. An unauthorized read of a protected program Flash location will read as '0'. A lower priority segment cannot read code space from a higher priority segment. Also, if a segment has invoked high security level, no other segment can read the code space. The Vector Space is always readable, regardless of the settings of other segments.

26.15.3 Rules for Writing Code or data EEPROM

The TBLWT instruction is not restricted from writing to the code space. This is because the TBLWT operation will cause a write to the Flash write data latches but will not invoke a programming operation. Thus, the TBLWT instruction is incapable of corrupting the code contents on its own.

26.16 Security Features and Device Operational Mode

Security functions are dependent on the operational mode of the device. Each device can operate in one of following modes:

- In Run-Time Self-Programming (RTSP) mode (normal device operation), the application code is running and the application code can invoke self programming.
- In In-Circuit Serial Programming™ (ICSP™) mode, the programming mode provides native, low-level programming capability to erase, program and verify the chip. The device is under the command of a device programmer such as a PRO MATE® 3 or MPLAB® ICD 2.

26.16.1 Rules for Programming Devices in RTSP

In RTSP, the device programs itself by using erase commands to first clear a portion of the code or data EEPROM. It then writes the new code or data into the write latches and, finally, uses a programming command to program the write latch contents into the Flash array. Erase or programming commands are specified by the device-specific NVMCON Special Function Register. The NVMOP bit field selects the particular function, and the ERASE bit selects between programming and erase functions. The WR bit within the NVMCON register invokes programming operations for either the code or data EEPROM. Consequently, to protect code and data EEPROM integrity, the device restricts the operations that occur on setting the WR bit.

26.16.1.1 Erasing and Programming Code Rows

Depending on the implementation of the Flash array, the NVMOP specifies erasing or programming a row of the program Flash array.

- If segment write protection is enabled, then no erase or programming operations will occur within that segment.
- Code running within a segment can erase or program part of its own segment.
- Code running within higher priority segments can erase or program part of a lower priority segment, unless the lower priority segment has selected high security.
- If the Vector Space has inherited high security associated with the Boot or General Segment, then no segment can erase or program part of the Vector Space. If a Boot Segment is defined, only the Boot Segment can erase or program part of the Vector Space. If no Boot Segment is defined, any segment can erase or program part of the Vector Space.

26.16.1.2 Erasing or Programming Data EEPROM Words or Rows

The NVMOP bit field selects erasing or programming, either words or rows, within the data EEPROM array.

- Code running within a segment can erase or program data EEPROM associated with its own segment.
- Code running within higher priority segments can erase or program the General Segment of data EEPROM, unless the General Segment has selected high security.
- Commands that bulk erase all of an allocated segment of data EEPROM follow the same rules as commands that erase a portion of the data EEPROM.

Note:	Segment write protection does not apply to data EEPROM.

26.16.1.3 Erasing a Segment and Clearing Code Protection

There are several variants of `NVMOP` commands that will erase an entire segment of program Flash, erase all lower priority segments of program Flash and clear the code protection configuration bits all in one operation. This is the only way to release code protection on a segment. These commands can be executed when running from any segment. These commands will not erase any contents from the data EEPROM nor clear the contents of the segment RAM.

26.16.2 Rules for Programming Devices Using In-Circuit Serial Programming (ICSP)

When the device is connected to a device programmer, the allowable operations are limited to erasing, programming and verifying the device code and data EEPROM memory.

- The device programmer uses segment erase commands to erase the device and clear the code protection.
- Programming commands are ignored if any level of code protection is selected. To program, there must be no Boot Segment or Secure Segment specified, and the General Segment must have no code protection.
- Devices with any level of code protection cannot be verified. Attempts to verify code-protected devices will result in reading '0's.

Once the device is programmed with the desired code, the configuration bits are written to enable the code protection level. After this operation, the only way to change the device code is by the code itself, or by erasing and clearing the code protection once more.

26.17 Typical Procedures for Boot Loading a Device

A typical scenario that requires boot loading a device that's using code protection, is a system field upgrade. Here, the device uses two segments, the Boot Segment and the General Segment. The General Segment contains the application. The Boot Segment contains a secure boot loader. Both segments have high security enabled.

At system Reset, the device vectors to the application in the General Segment.

As the system is operating in the field, a technician connects a reprogramming tool to the system. The application recognizes this connection and branches to a location within the Boot Segment access area. This branch is highly secure and the attempt to modify this branch will likely result in a device Reset.

The Boot Segment contains code that allows encrypted communication with the tool. The encryption keys are safe within the contents of the Boot Segment code because only the Boot Segment can access them. If serialized programming had been used when the boot loader was initially programmed into the system, the encryption key could be specific to a particular system, further enhancing the strength of the encrypted communication.

Once the boot loader verifies valid communication with the external programming tool, it can then erase the code within the General Segment and clear the General Segment code protection.

The boot loader then receives the encrypted code update from the tool, decrypts it and programs it into the general space.

As the boot loader is running, it is immune from disruption by interrupts or traps as it can vector those to a secure location within the boot loader itself.

As the boot loader finishes, it can program the configuration bits to re-protect the general space, make any necessary updates to the vectors and then return to the general application.

26.18 Typical Installation of Third Party Protected Algorithm

In this scenario, the system integrator is purchasing an algorithm from a third party vendor. Here, the system integrator wants to protect his system code from problems with the third party algorithm, and the third party algorithm vendor wants to ensure that no one in the system integrator's company compromises his code.

Normally, this is a matter of trust for the third party vendor, since typically the third party vendor must provide the integrator with native code that can be linked into the integrator's system code and programmed into the device.

If the third party vendor can provide his code in an encrypted fashion and the device can isolate the third party code, the third party vendor can ensure that his code remains proprietary.

In this scenario, the device allocates Boot, Secure and General segments. The system integrator's code resides in the General Segment and a boot loader similar to the one described in the previous section is contained in the Boot Segment.

As the system integrator builds his system, his boot loader and application code are programmed into the device.

A special loader, provided by the third party vendor, is programmed into the Secure Segment. This loader decrypts and programs the third party's algorithm using a key provided by the third party vendor to the system integrator. Once the Secure Segment is protected, the algorithm inside is not accessible by the system integrator's code in the Boot or General Segments. The third party code can only be accessed through calls to the Secure Segment access area.

The protected algorithm should maintain critical data parameters within protected RAM or protected data EEPROM areas. When the algorithm is about to finish and return to code within another segment, it should "cleanse" its RAM and data EEPROM areas.

26.19 Design Tips or FAQs

Question 1: ***Can I boot load a device with basic code protection?***

Answer: Remember that devices with basic code protection only have one segment, the General Segment. Because there is only one segment, it is not possible to erase the segment and clear code protection without also erasing any boot loader that might be resident within the General Segment.

This limits the options for booting but does not prevent it. The boot loader will need to erase and reprogram Flash in “less than segment” partitions, and the loader cannot select write protection for the General Segment. It will also not be possible to protect the loaded code from compromises caused by the boot loader itself.

Question 2: ***Can the system load part of the code now and the rest of the code later?***

Answer: As long as neither write protection nor high security is selected for the segment, “incremental” loads are possible. Incremental loads are still possible in high security segments as long as the loader resides within that segment. However, once the segment is write-protected, it cannot be changed until the entire segment is erased and code protection is cleared by a segment erase command.

You can choose to locate a jump-table for interrupt vectors in an unprotected segment and update the jump-table with changing interrupt vectors. This will allow Boot Segment write protection.

26.20 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the dsPIC30F Product Family, but the concepts are pertinent and could be used with modification and possible limitations. Current application notes related to the CodeGuard Security include the following:

Title	Application Note #
CodeGuard™ Security: Protecting Intellectual Property in Collaborative System Designs	DS70179

Note: Please visit the Microchip web site (www.microchip.com/codeguard) for additional Application Notes and code examples for the dsPIC30F family of devices.

26.21 Revision History

Revision A (March 2007)

This is the initial release of this document.

NOTES: