Section 3. Memory Organization

HIGHLIGHTS

This section of the manual contains the following topics:

3.1 INTRODUCTION

The PIC32MX microcontrollers provide 4 GB of unified virtual memory address space. All memory regions, including program memory, data memory, SFRs, and Configuration registers reside in this address space at their respective unique addresses. The program and data memories can be optionally partitioned into user and kernel memories. In addition, the data memory can be made executable, allowing the PIC32MX to execute from data memory.

Key features of PIC32MX memory organization include the following:

- 32-bit native data width
- Separate User and Kernel mode address spaces
- Flexible program Flash memory partitioning
- Flexible data RAM partitioning for data and program space
- Separate boot Flash memory for protected code
- Robust bus-exception handling to intercept runaway code
- Simple memory mapping with Fixed Mapping Translation (FMT) unit
- Cacheable and non-cacheable address regions

3.2 CONTROL REGISTERS

This section lists the Special Function Registers (SFRs) registers used for setting the RAM and Flash memory partitions for data and code (for both User and Kernel mode).

The following is a list of available SFRs:

- BMXCON: Configuration Register

  BMXCONCLR, BMXCONSET, BMXCONINV: Atomic Bit Manipulation Registers for BMXCON
- BMXxxxBA: Memory Partition Base Address Registers

  BMXxxxBACLR, BMXxxxBASET, BMXxxxBAINV: Atomic Bit Manipulation Registers for BMXxxxBA
- BMXDRMSZ: Data RAM Size Register
- BMXPFMSZ: Program Flash Size Register
- BMXBOOTSZ: Boot Flash Size Register

Table 3-1 provides a brief summary of all Memory Organization-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register.

Table 3-1: Memory Organization SFR Summary

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BMXCON	31:24	—	—	—	—	—	BMX-CHEDMA	—	—
	23:16	—	—	—	BMXER-RIXI	BMXER-RICD	BMXER-RDMA	BMXER-RDS	BMXERRIS
	15:8	—	—	—	—	—	—	—	—
	7:0	—	BMXWS-DRM	—	—	—	BMXARB		
BMXCONCLR	31:0	Write clears selected bits in BMXCON, read yields undefined value							
BMXCONSET	31:0	Write sets selected bits in BMXCON, read yields undefined value							
BMXCONINV	31:0	Write inverts selected bits in BMXCON, read yields undefined value							
BMXDKPBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BMXDKPBA<15:8>							
	7:0	BMXDKPBA<7:0>							
BMXDKPBACLR	31:0	Write clears selected bits in BMXDKPBA, read yields undefined value							
BMXDKPBASET	31:0	Write sets selected bits in BMXDKPBA, read yields undefined value							
BMXDKPBAINV	31:0	Write inverts selected bits in BMXDKPBA, read yields undefined value							
BMXDUDBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BMXDUDBA<15:8>							
	7:0	BMXDUDBA<7:0>							
BMXDUDBACLR	31:0	Write clears selected bits in BMXDUDBA, read yields undefined value							
BMXDUDBASET	31:0	Write sets selected bits in BMXDUDBA, read yields undefined value							
BMXDUDBAINV	31:0	Write inverts selected bits in BMXDUDBA, read yields undefined value							
BMXDUPBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	BMXDUPBA<15:8>							
	7:0	BMXDUPBA<7:0>							
BMXDUPBACLR	31:0	Write clears selected bits in BMXDUPBA, read yields undefined value							
BMXDUPBASET	31:0	Write sets selected bits in BMXDUPBA, read yields undefined value							
BMXDUPBAINV	31:0	Write inverts selected bits in BMXDUPBA, read yields undefined value							

Table 3-1: Memory Organization SFR Summary (Continued)

Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BMXDRMSZ	31:24	BMXDRMSZ<31:24>							
	23:16	BMXDRMSZ<23:16>							
	15:8	BMXDRMSZ<15:8>							
	7:0	BMXDRMSZ<7:0>							
BMXPUPBA	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	BMXPUPBA<19:16>			
	15:8	BMXPUPBA<15:8>							
	7:0	BMXPUPBA<7:0>							
BMXPUPBACLR	31:0	Write clears selected bits in BMXPUPBA, read yields undefined value							
BMXPUPBASET	31:0	Write sets selected bits in BMXPUPBA, read yields undefined value							
BMXPUPBAINV	31:0	Write inverts selected bits in BMXPUPBA, read yields undefined value							
BMXPFMSZ	31:24	BMXPFMSZ<31:24>							
	23:16	BMXPFMSZ<23:16>							
	15:8	BMXPFMSZ<15:8>							
	7:0	BMXPFMSZ<7:0>							
BMXBOOTSZ	31:24	BMXBOOTSZ<31:24>							
	23:16	BMXBOOTSZ<23:16>							
	15:8	BMXBOOTSZ<15:8>							
	7:0	BMXBOOTSZ<7:0>							

Register 3-1: BMXCON: Bus Matrix Configuration Register

r-x	r-x	r-x	r-x	r-x	R/W-0	r-x	r-x
—	—	—	—	—	BMX-CHEDMA	—	—
bit 31							bit 24

r-x	r-x	r-x	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	—	—	BMXERRIXI	BMXER-RICD	BMXER-RDMA	BMXER-RDS	BMXERRIS
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 15							bit 8

r-x	R/W-1	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0
—	BMXWS-DRM	—	—	—	BMXARB<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-27	**Reserved:** Write ‘0’; ignore read
bit 26	**BMXCHEDMA:** BMX PFM Cacheability for DMA Accesses bit 1 = Enable program Flash memory (data) cacheability for DMA accesses (requires cache to have data caching enabled) 0 = Disable program Flash memory (data) cacheability for DMA accesses (hits are still read from the cache, but misses do not update the cache)
bit 25 - 21	**Reserved:** Write ‘0’; ignore read
bit 20	**BMXERRIXI:** Enable Bus Error from IXI bit 1 = Enable bus error exceptions for unmapped address accesses initiated from IXI shared bus 0 = Disable bus error exceptions for unmapped address accesses initiated from IXI shared bus
bit 19	**BMXERRICD:** Enable Bus Error from ICD Debug Unit bit 1 = Enable bus error exceptions for unmapped address accesses initiated from ICD 0 = Disable bus error exceptions for unmapped address accesses initiated from ICD
bit 18	**BMXERRDMA:** Bus Error from DMA bit 1 = Enable bus error exceptions for unmapped address accesses initiated from DMA 0 = Disable bus error exceptions for unmapped address accesses initiated from DMA
bit 17	**BMXERRDS:** Bus Error from CPU Data Access bit (disabled in DEBUG mode) 1 = Enable bus error exceptions for unmapped address accesses initiated from CPU data access 0 = Disable bus error exceptions for unmapped address accesses initiated from CPU data access
bit 16	**BMXERRIS:** Bus error from CPU Instruction Access bit (disabled in DEBUG mode) 1 = Enable bus error exceptions for unmapped address accesses initiated from CPU instruction access 0 = Disable bus error exceptions for unmapped address accesses initiated from CPU instruction access
bit 15 - 7	**Reserved:** Write ‘0’; ignore read

Register 3-1: BMXCON: Bus Matrix Configuration Register (Continued)

bit 6	**BMXWSDRM:** CPU Instruction or Data Access from Data RAM Wait State bit `1` = Data RAM accesses from CPU have one wait state for address setup `0` = Data RAM accesses from CPU have zero wait states for address setup
bit 5-3	**Reserved:** Write '`0`'; ignore read
bit 2-0	**BMXARB<2:0>:** Bus Matrix Arbitration Mode bits `111...011` = Reserved (using these Configuration modes will produce undefined behavior) `010` = Arbitration Mode 2 `001` = Arbitration Mode 1 `000` = Arbitration Mode 0

Register 3-2: BMXCONCLR: BMXCON Clear Register

Write clears selected bits in BMXCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXCON**

A write of '`1`' in one or more bit positions clears the corresponding bit(s) in BMXCON register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXCONCLR = 0x00000101` will clear bits 15 and 0 in BMXCON register.

Register 3-3: BMXCONSET: BMXCON Set Register

Write sets selected bits in BMXCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXCON**

A write of '`1`' in one or more bit positions sets the corresponding bit(s) in BMXCON register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXCONSET = 0x00000101` will set bits 15 and 0 in BMXCON register.

Register 3-4: BMXCONINV: BMXCON Invert Register

Write inverts selected bits in BMXCON, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXCON**

A write of '`1`' in one or more bit positions inverts the corresponding bit(s) in BMXCON register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXCONINV = 0x00000101` will invert bits 15 and 0 in BMXCON register.

Register 3-5: BMXDKPBA: Data RAM Kernel Program Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXDKPBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXDKPBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16	**Reserved:** Write ‘0’; ignore read
bit 15-11	**BMXDKPBA<15:11>:** DRM Kernel Program Base Address bits When non-zero, this value selects the relative base address for kernel program space in RAM
bit 10-0	**BMXDKPBA<10:0>:** Read-Only bits Value is always ‘0’, which forces 2 KB increments

Register 3-6: BMXDKPBACLR: BMXDKPBA Clear Register

Write clears selected bits in BMXDKPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXDKPBA**

A write of ‘`1`’ in one or more bit positions clears the corresponding bit(s) in BMXDKPBA register and does not affect unimplemented or read-only bits. A write of ‘`0`’ will not affect the register.

Example: `BMXDKPBACLR = 0x00000101` will clear bits 15 and 0 in BMXDKPBA register.

Register 3-7: BMXDKPBASET: BMXDKPBA Set Register

Write sets selected bits in BMXDKPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXDKPBA**

A write of ‘`1`’ in one or more bit positions sets the corresponding bit(s) in BMXDKPBA register and does not affect unimplemented or read-only bits. A write of ‘`0`’ will not affect the register.

Example: `BMXDKPBASET = 0x00000101` will set bits 15 and 0 in BMXDKPBA register.

Register 3-8: BMXDKPBAINV: BMXDKPBA Invert Register

Write inverts selected bits in BMXDKPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXDKPBA**

A write of ‘`1`’ in one or more bit positions inverts the corresponding bit(s) in BMXDKPBA register and does not affect unimplemented or read-only bits. A write of ‘`0`’ will not affect the register.

Example: `BMXDKPBAINV = 0x00000101` will invert bits 15 and 0 in BMXDKPBA register.

Register 3-9: BMXDUDBA: Data RAM User Data Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXDUDBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXDUDBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16	**Reserved:** Write ‘0’; ignore read
bit 15-11	**BMXDUDBA<15:11>:** DRM User Data Base Address bits When non-zero, the value selects the relative base address for User mode data space in RAM **Note:** If non-zero, the value must be greater than BMXDKPBA.
bit 10-0	**BMXDUDBA<10:0>:** Read-Only bits Value is always ‘0’, which forces 2 KB increments

Register 3-10: BMXDUDBACLR: BMXDUDBA Clear Register

Write clears selected bits in BMXDUDBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXDUDBA**

A write of '`1`' in one or more bit positions clears the corresponding bit(s) in BMXDUDBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXDUDBACLR = 0x00000101` will clear bits 15 and 0 in BMXDUDBA register.

Register 3-11: BMXDUDBASET: BMXDUDBA Set Register

Write sets selected bits in BMXDUDBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXDUDBA**

A write of '`1`' in one or more bit positions sets the corresponding bit(s) in BMXDUDBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXDUDBASET = 0x00000101` will set bits 15 and 0 in BMXDUDBA register.

Register 3-12: BMXDUDBAINV: BMXDUDBA Invert Register

Write inverts selected bits in BMXDUDBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXDUDBA**

A write of '`1`' in one or more bit positions inverts the corresponding bit(s) in BMXDUDBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXDUDBAINV = 0x00000101` will invert bits 15 and 0 in BMXDUDBA register.

Register 3-13: BMXDUPBA: Data RAM User Program Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXDUPBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXDUPBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-16	**Reserved:** Write ‘0’; ignore read
bit 15-11	**BMXDUPBA<15:11>:** DRM User Program Base Address bits When non-zero, the value selects the relative base address for User mode program space in RAM **Note:** If non-zero, BMXDUPBA must be greater than BMXDUDBA.
bit 10-0	**BMXDUPBA<10:0>:** Read-Only bits Value is always ‘0’, which forces 2 KB increments

Register 3-14: BMXDUPBACLR: BMXDUPBA Clear Register

Write clears selected bits in BMXDUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXDUPBA**

A write of '`1`' in one or more bit positions clears the corresponding bit(s) in BMXDUPBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXDUPBACLR = 0x00000101` will clear bits 15 and 0 in BMXDUPBA register.

Register 3-15: BMXDUPBASET: BMXDUPBA Set Register

Write sets selected bits in BMXDUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXDUPBA**

A write of '`1`' in one or more bit positions sets the corresponding bit(s) in BMXDUPBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXDUPBASET = 0x00000101` will set bits 15 and 0 in BMXDUPBA register.

Register 3-16: BMXDUPBAINV: BMXDUPBA Invert Register

Write inverts selected bits in BMXDUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXDUPBA**

A write of '`1`' in one or more bit positions inverts the corresponding bit(s) in BMXDUPBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXDUPBAINV = 0x00000101` will invert bits 15 and 0 in BMXDUPBA register.

Register 3-17: BMXDRMSZ: Data RAM Size Register

R	R	R	R	R	R	R	R
BMXDRMSZ<31:24>							
bit 31							bit 24

R	R	R	R	R	R	R	R
BMXDRMSZ<23:16>							
bit 23							bit 16

R	R	R	R	R	R	R	R
BMXDRMSZ<15:8>							
bit 15							bit 8

R	R	R	R	R	R	R	R
BMXDRMSZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0 **BMXDRMSZ:** Data RAM Memory (DRM) Size bits

Static value that indicates the size of the Data RAM in bytes:
.......0x00002000 = device has 8 KB RAM
0x00004000 = device has 16 KB RAM
0x00008000 = device has 32 KB RAM

Register 3-18: BMXPUPBA: Program Flash (PFM) User Program Base Address Register

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	BMXPUPBA<19:16>			
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
BMXPUPBA<15:8>							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
BMXPUPBA<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-20	Unimplemented: Read as ‘0’
bit 19-11	**BMXPUPBA<19:11>:** Program Flash (PFM) User Program Base Address bits
bit 10-0	**BMXPUPBA<10:0>:** Read-Only bits Value is always ‘0’, which forces 2 KB increments

Register 3-19: BMXPUPBACLR: BMXPUPBA Clear Register

Write clears selected bits in BMXPUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Clears selected bits in BMXPUPBA**

A write of '`1`' in one or more bit positions clears the corresponding bit(s) in BMXPUPBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXPUPBACLR = 0x00000101` will clear bits 15 and 0 in BMXPUPBA register.

Register 3-20: BMXPUPBASET: BMXPUPBA Set Register

Write sets selected bits in BMXPUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Sets selected bits in BMXPUPBA**

A write of '`1`' in one or more bit positions sets the corresponding bit(s) in BMXPUPBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXPUPBASET = 0x00000101` will set bits 15 and 0 in BMXPUPBA register.

Register 3-21: BMXPUPBAINV: BMXPUPBA Invert Register

Write inverts selected bits in BMXPUPBA, read yields undefined value	
bit 31	bit 0

bit 31-0 **Inverts selected bits in BMXPUPBA**

A write of '`1`' in one or more bit positions inverts the corresponding bit(s) in BMXPUPBA register and does not affect unimplemented or read-only bits. A write of '`0`' will not affect the register.

Example: `BMXPUPBAINV = 0x00000101` will invert bits 15 and 0 in BMXPUPBA register.

Register 3-22: BMXPFMSZ: Program Flash (PFM) Size Register

R	R	R	R	R	R	R	R
BMXPFMSZ<31:24>							
bit 31							bit 24

R	R	R	R	R	R	R	R
BMXPFMSZ<23:16>							
bit 23							bit 16

R	R	R	R	R	R	R	R
BMXPFMSZ<15:8>							
bit 15							bit 8

R	R	R	R	R	R	R	R
BMXPFMSZ<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: ('0', '1', x = Unknown)		

bit 31-0 **BMXPFMSZ:** Program Flash Memory (PFM) Size bits

Static value that indicates the size of the PFM in bytes:
0x00008000 = device has 32 KB Flash
.......0x00010000 = device has 64 KB Flash
0x00020000 = device has 128 KB Flash
0x00040000 = device has 256 KB Flash
0x00080000 = device has 512 KB Flash

Register 3-23: BMXBOOTSZ: Boot Flash (IFM) Size Register

R	R	R	R	R	R	R	R
BMXBOOTSZ<31:24>							
bit 31							bit 24

R	R	R	R	R	R	R	R
BMXBOOTSZ<23:16>							
bit 23							bit 16

R	R	R	R	R	R	R	R
BMXBOOTSZ<15:8>							
bit 15							bit 8

R	R	R	R	R	R	R	R
BMXBOOTSZ<7:0>							
bit 7							bit 0

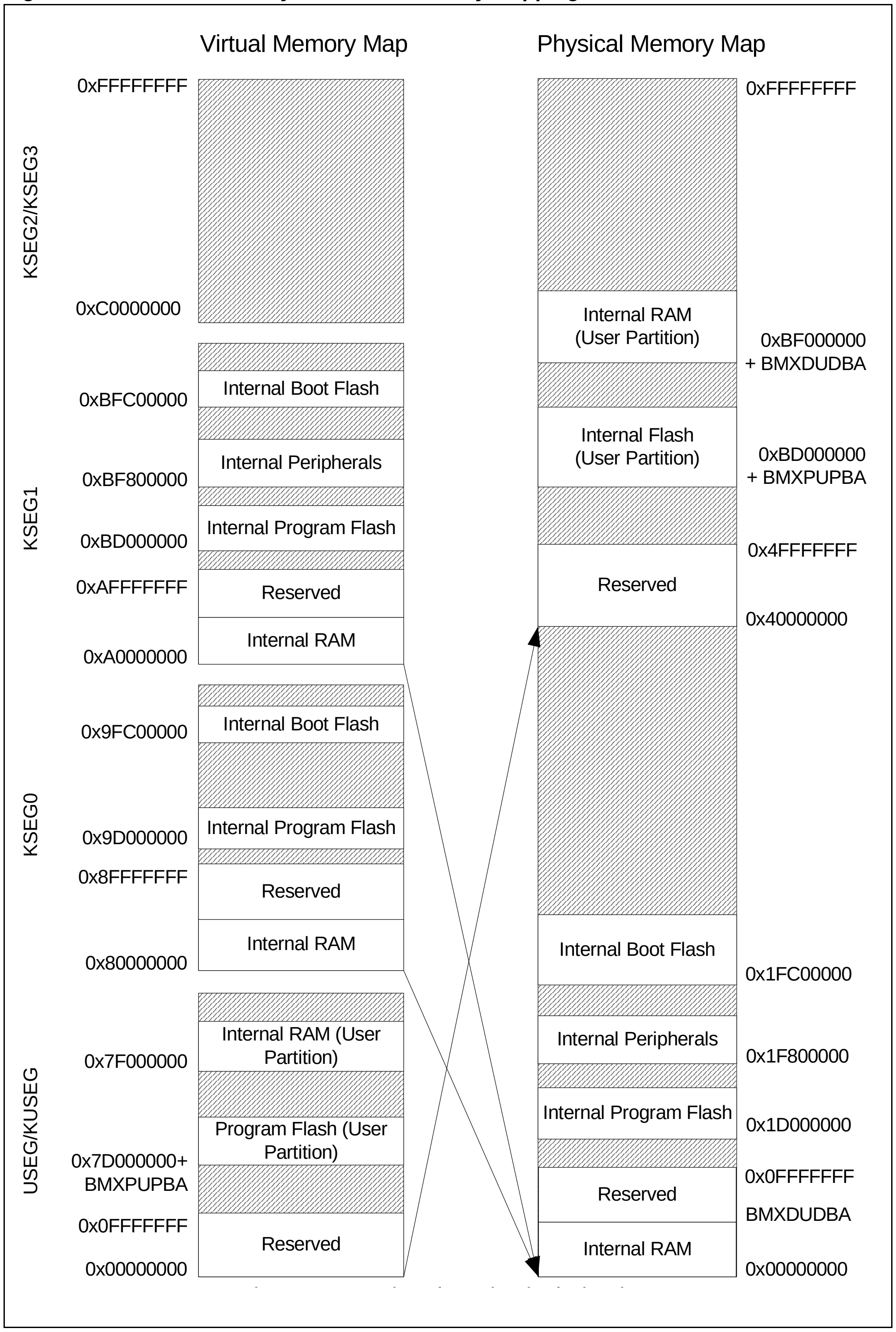
Legend:			
R = Readable bit	W = Writable bit	P = Programmable bit	r = Reserved bit
U = Unimplemented bit	-n = Bit Value at POR: (‘0’, ‘1’, x = Unknown)		

bit 31-0	**BMXBOOTSZ:** Boot Flash Memory (BFM) Size bits Static value that indicates the size of the Boot PFM in bytes: 0x00003000 = device has 12 KB boot Flash

3.3 PIC32MX MEMORY LAYOUT

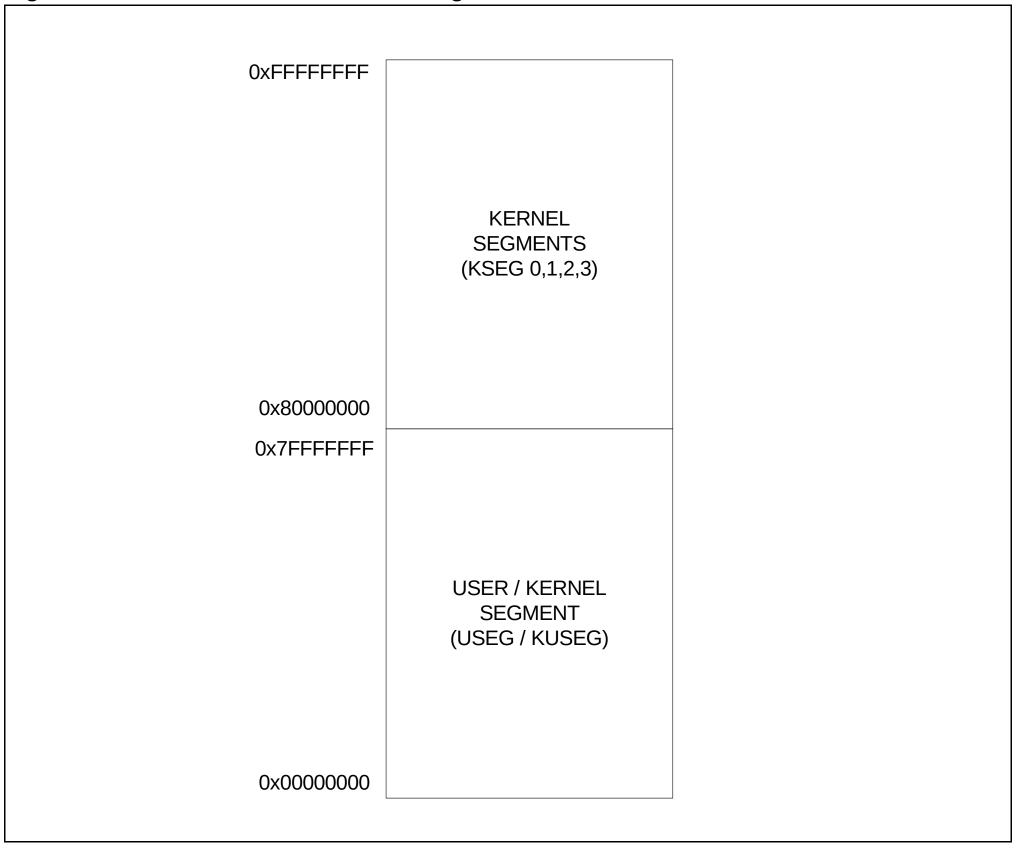
The PIC32MX microcontrollers implement two address spaces: virtual and physical. All hardware resources, such as program memory, data memory and peripherals, are located at their respective physical addresses. Virtual addresses are exclusively used by the CPU to fetch and execute instructions. Physical addresses are used by peripherals, such as DMA and Flash controllers, that access memory independently of the CPU.

Figure 3-1: Virtual to Physical Fixed Memory Mapping



The entire 4 GB virtual address space is divided into two primary regions: user and kernel space. The lower 2 GB of space from the User mode segment is called useg/kuseg. A User mode application must reside and execute in the useg segment. The useg segment is also available to all Kernel mode applications, which is why it is also named kuseg – to indicate that it is available to both User and Kernel modes. When operating in User mode, the bus matrix must be configured to make part of the Flash and data memory available in the useg/kuseg segment. See Section 3.4 for more information.

Figure 3-2: User/Kernel Address Segments


0xFFFFFFFF
KERNEL SEGMENTS (KSEG 0,1,2,3)
0x80000000
0x7FFFFFFF
USER / KERNEL SEGMENT (USEG / KUSEG)
0x00000000

The upper 2 GB of virtual address space forms the kernel only space. The kernel space is divided into four segments of 512 MB each: kseg 0, kseg 1, kseg 2 and kseg 3. Only Kernel mode applications can access kernel space memory. The kernel space includes all peripheral registers. Consequently, only Kernel mode applications can monitor and manipulate peripherals. Only kseg 0 and kseg 1 segments point to real memory resources. Segment kseg 2 is available to the EJTAG probe debugger, as explained in the MIPS documentation (refer to the EJTAG specification). The PIC32MX only uses kseg 0 and kseg 1 segments. The Boot Flash Memory (BFM), Program Flash Memory (PFM), Data RAM Memory (DRM), and peripheral SFRs are accessible from either kseg 0 or kseg 1.

The Fixed Mapping Translation (FMT) unit translates the memory segments into corresponding physical address regions. Figure 3-1 shows the fixed mapping scheme implemented by the PIC32MX core between the virtual and physical address space. A virtual memory segment may also be cached, provided the cache module is available on the device. Please note that the kseg-1 memory segment is not cacheable, while kseg-0 and useg/kuseg are cacheable.

The mapping of the memory segments depend on the CPU error level (set by the ERL bit in the CPU Status register). Error Level is set (ERL = 1) by the CPU on a Reset, Soft Reset, or NMI. In this mode, the processor runs in Kernel mode and useg/kuseg are treated as unmapped and uncached regions, and the mapping in Figure 3-1 does not apply. This mode is provided for compatibility with other MIPS processor cores that use a TLB-based MMU. The C start-up code clears the ERL bit to zero, so that when application software starts up, it sees the proper virtual to physical memory mapping as depicted in Figure 3-1.

Segments kseg 0 and kseg 1 are always translated to physical address 0x0. This translation arrangement allows the CPU to access identical physical addresses from two separate virtual addresses: one from kseg 0 and the other from kseg 1. As a result, the application can choose to execute the same piece of code as either cached or uncached. See **Section 4. "Prefetch Cache Module"** for more information. The on-chip peripherals are visible through kseg 1 segment only (uncached access).

3.4 PIC32MX ADDRESS MAP

The Program Flash Memory is divided into kernel and user partitions. The kernel program Flash space starts at physical address 0x1D000000, whereas the user program Flash space starts at physical address 0xBD000000 + BMXPUDBA register value. Similarly, the internal RAM is also divided into kernel and user partitions. The kernel RAM space starts at physical address 0x00000000, whereas the user RAM space starts at physical address 0xBF000000 + BMXDUDBA register value. By default, the full Flash memory and RAM are mapped to Kernel mode application only.

Please note that the BMXxxxBA register settings must match the memory model of the target software application. If the linked code does not match the register values, the program may not run and may generate bus error exceptions on start-up.

Note: The Program Flash Memory is not writable through its address map. A write to the PFM address range causes a bus error exception.

3.4.1 Virtual to Physical Address Calculation (and Vice-Versa)

To translate the kernel address (KSEG0 or KSEG1) to a physical address, perform a “Bitwise AND” operation of the virtual address with 0x1FFFFFFF:

Physical Address = Virtual Address and 0x1FFFFFFF

For physical address to KSEG0 virtual address translation, perform a “Bitewise OR” operation of the physical address with 0x80000000:

KSEG0 Virtual Address = Physical Address | 0x80000000

For physical address to KSEG1 virtual address translation, perform a “Bitewise OR” operation of the physical address with 0xA0000000:

KSEG1 Virtual Address = Physical Address | 0xA0000000

To translate from KSEG0 to KSEG1 virtual address, perform a “Bitewise OR” operation of the KSEG0 virtual address with 0x20000000:

KSEG1 Virtual Address = KSEG0 Virtual Address | 0x20000000

Table 3-2: PIC32MX Address Map

		Virtual Addresses		Physical Addresses		Size in Bytes
	Memory Type	**Begin Address**	**End Address**	**Begin Address**	**End Address**	**calculation**
Kernel Address Space	**Boot Flash**	0xBFC00000	0xBFC02FFF	0x1FC00000	0x1FC02FFF	12 KB
	Program Flash[(1)]	0xBD000000	0xBD000000 + BMXPUPBA - 1	0x1D000000	0x1D00000 + BMXPUPBA - 1	BMXPUPBA
	Program Flash[(2)]	0x9D000000	0x9D000000 + BMXPUPBA - 1	0x1D000000	0x1D000000 + BMXPUPBA - 1	BMXPUPBA
	RAM (Data)	0x80000000	0x80000000 + BMXDKPBA - 1	0x00000000	BMXDKPBA - 1	BMXDKPBA
	RAM (Prog)	0x80000000 + BMXDKPBA	0x80000000 + BMXDUDBA -1	BMXDKPBA	BMXDUDBA -1	BMXDUDBA - BMXDKPBA
	Peripheral	0xBF800000	0xBF8FFFFF	0x1F800000	0x1F8FFFFF	1 MB
User Address Space	**Program Flash**	0x7D000000 + BMXPUPBA	0x7D000000 + PFM Size - 1	0xBD000000 + BMXPUPBA	0xBD000000 + PFM Size - 1	PFM Size - BMXPUPBA
	RAM (Data)	0x7F000000 + BMXDUDBA	0x7F000000 + BMXDUPBA - 1	0xBF000000 + BMXDUDBA	0xBF000000 + BMXDUPBA - 1	BMXDUPBA - BMXDUDBA
	RAM (Prog)	0x7F000000 + BMXDUPBA	0x7F000000 + RAM Size[(3)] - 1	0xBF000000 + BMXDUPBA	0xBF000000 + RAM Size[(3)] - 1	DRM Size - BMXDUPBA

Note 1: Program Flash virtual addresses in the non-cacheable range (KSEG1).
2: Program Flash virtual addresses in the cacheable and prefetchable range (KSEG0).
3: The RAM size varies between PIC32MX device variants.

3.4.2 Program Flash Memory Partitioning

The Program Flash Memory can be partitioned for User and Kernel mode programs as shown in Figure 3-1.

At Reset, the User mode partition does not exist (BMXPUPBA is initialized to `0`). The entire Program Flash Memory is mapped to Kernel mode program space starting at virtual address KSEG1: 0xBD000000 (or KSEG0: 0x9D000000). To set up a partition for the User mode program, initialize BMXPUPBA as follows:

BMXPUPBA = BMXPFMSZ – USER_FLASH_PGM_SZ

The USER_FLASH_PGM_SZ is the partition size of the User mode program. BMXPFMSZ is the bus matrix register that holds the total size of Program Flash Memory.

Example:

Assuming the PIC32MX device has 512 Kbytes of Flash memory, the BMXPFMSZ will contain 0x00080000.

To create a user Flash program partition of 20 Kbytes (0x5000):

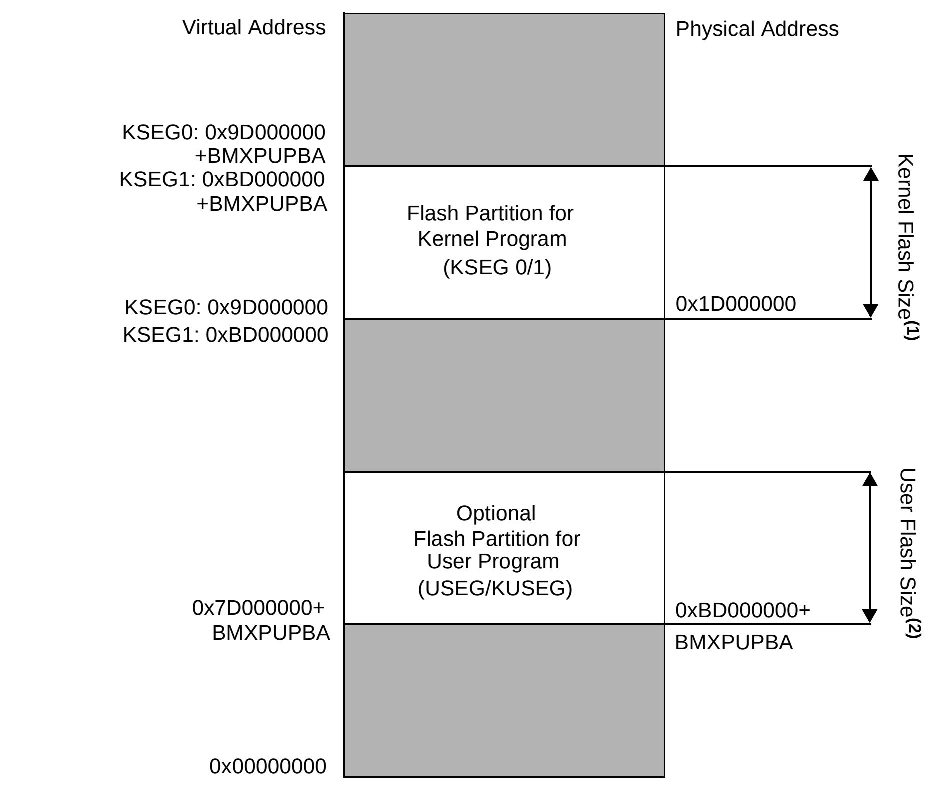
BMXPUPBA = 0x80000 – 0x5000 = 0x7B000

The size of the user Flash will be 20K and the size left for the Kernel Flash will be 512k – 20k = 492K.

The user Flash partition will extend from 0x7D07B000 to 0x7D07FFFF (virtual addresses).

The Kernel mode partition always starts from KSEG1: 0xBD000000 or KSEG0: 0x9D000000. In the above example, the Kernel partition will extend from 0xBD000000 to 0xBD07AFFF (492 Kbytes in size).

Figure 3-3: Flash Partitioning



Note 1: Kernel Flash Size = BMXPUPBA

2: User Flash Size = BMXPFMSZ-BMXPUPBA

3: If BMXPUPBA is ‘0’, then:
K Flash Size = BMXPFMSZ (i.e., all the Flash)
Usr Flash Size = 0

3.4.3 RAM Partitioning

The RAM memory can be divided into 4 partitions. These are:

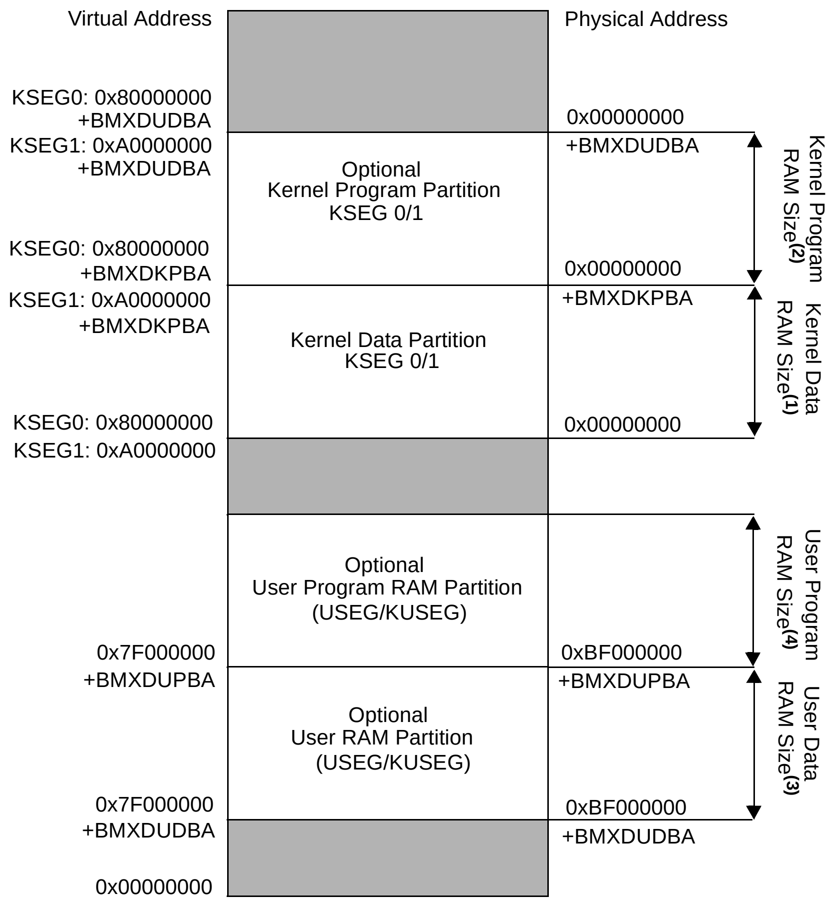
1. Kernel Data
2. Kernel Program
3. User Data
4. User Program

In order to execute from data RAM, a kernel or user program partition must be defined. At Power-on Reset, the entire data RAM is assigned to the kernel data partition. This partition always starts from the base of the data RAM. See Figure 3-4 for details.

Note 1: To properly partition the RAM, you have to program all of the following registers: BMXDKPBA, BMXDUDBA and BMXDUPBA.

2: The size of the available RAM is given by the BMXDRMSZ register.

Figure 3-4: RAM Partitioning


Virtual Address
Physical Address
KSEG0: 0x80000000 +BMXDUDBA
KSEG1: 0xA0000000 +BMXDUDBA
0x00000000 +BMXDUDBA
Optional Kernel Program Partition KSEG 0/1
Kernel Program RAM Size(2)
KSEG0: 0x80000000 +BMXDKPBA
KSEG1: 0xA0000000 +BMXDKPBA
0x00000000 +BMXDKPBA
Kernel Data Partition KSEG 0/1
Kernel Data RAM Size(1)
KSEG0: 0x80000000
KSEG1: 0xA0000000
0x00000000
Optional User Program RAM Partition (USEG/KUSEG)
User Program RAM Size(4)
0x7F000000 +BMXDUPBA
0xBF000000 +BMXDUPBA
Optional User RAM Partition (USEG/KUSEG)
User Data RAM Size(3)
0x7F000000 +BMXDUDBA
0xBF000000 +BMXDUDBA
0x00000000

Note 1: Kernel Data RAM Size = BMXDKPBA

2: Kernel Program RAM Size = BMXDUDBA – BMXDKPBA

3: User Data RAM Size = BMXDUPBA – BMXDUDBA

4: User Program RAM Size = DRM Size – BMXDUPBA

5: If BMXDKPBA, BMXDUDBA or BMXDUPBA is ‘0’, then:
Kernel Data RAM Size = BMXDRMSZ (i.e., all RAM)
Kernel Program RAM Size = 0
User Data RAM Size = 0
User Program RAM Size = 0

3.4.3.1 Kernel Data RAM Partition

The kernel data RAM partition is located at virtual address KSEG0:0x80000000, KSEG1:0xA0000000. It is always active and cannot be disabled.

Please note that if any of the BMXDKPBA, BMXDUDBA or BMXDUPBA register is '0', then the whole RAM is assigned to kernel data RAM (i.e., the size of the kernel data RAM partition is given by the BMXDRMSZ register value; see Figure 3-5). Otherwise, the size of the kernel data RAM partition is given by the value of the BMXDKPBA register. See Figure 3-6.

The kernel data RAM partition exists on Reset and takes up all the available RAM, as the BMXD-KPBA, BMXDUDBA and BMXDUPBA registers default to zero at any Reset.

Figure 3-5: RAM Partitioning When BMXDKPBA, BMXDUDBA or BMXDUPBA = 0

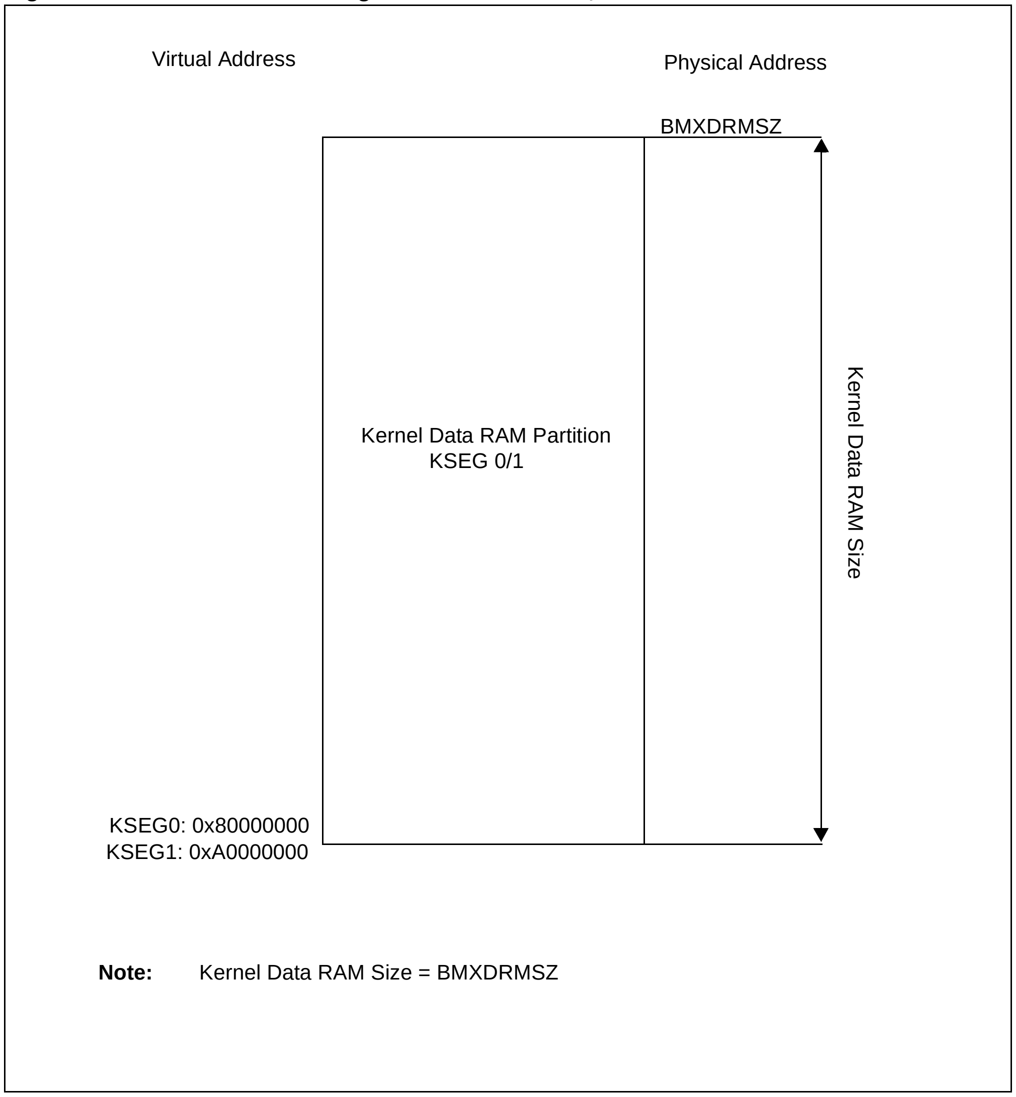
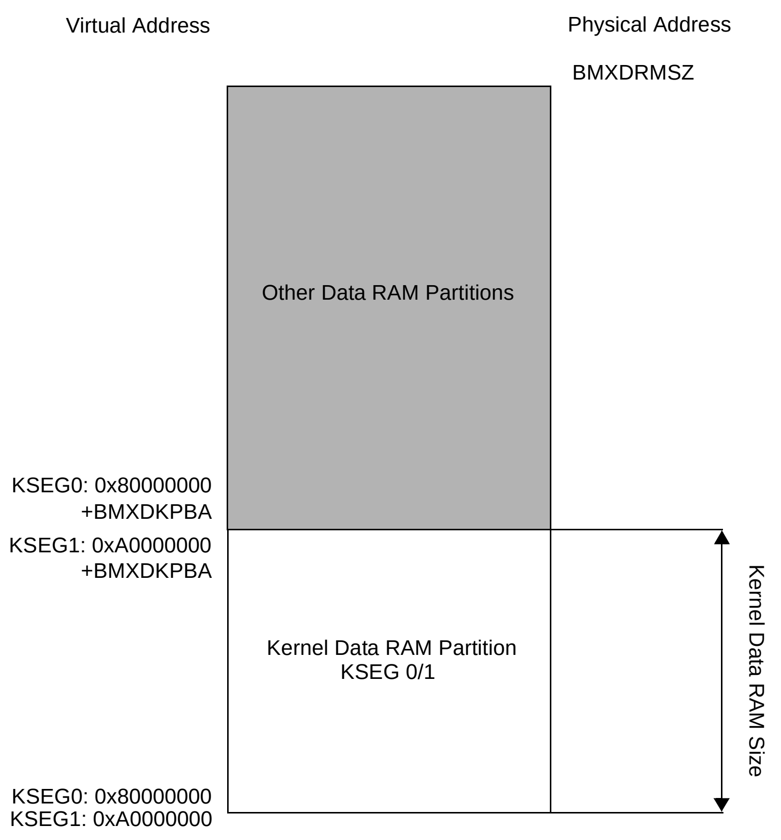


Figure 3-6: Kernel Data RAM Partitioning


Virtual Address
Physical Address
BMXDRMSZ
Other Data RAM Partitions
KSEG0: 0x80000000 +BMXDKPBA
KSEG1: 0xA0000000 +BMXDKPBA
Kernel Data RAM Partition KSEG 0/1
Kernel Data RAM Size
KSEG0: 0x80000000
KSEG1: 0xA0000000

Note 1: Kernel Data RAM Size = BMXDKPBA.

2: None of the registers BMXDKPBA, BMXDUDBA or BMXDUPBA = `0`.

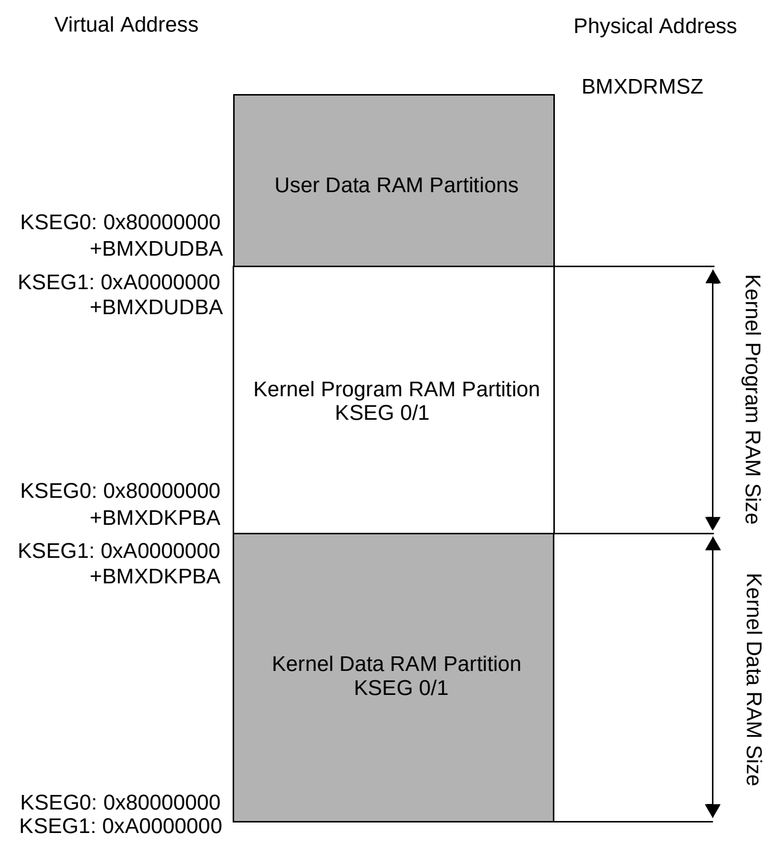
3.4.3.2 Kernel Program RAM Partition

The kernel program RAM partition is required if code needs to be executed from data RAM in Kernel mode.

This partition starts at KSEG0:0x80000000 + BMXDKPBA (KSEG1:0xA0000000 + BMXDKPBA), and its size is given by BMXDUDBA – BMXDKPBA. See Figure 3-7.

The kernel program RAM partition does not exist on Reset, as the BMXDKPBA and BMXDUDBA registers default to zero at Reset.

Figure 3-7: Kernel Program RAM Partitioning


Virtual Address
Physical Address
BMXDRMSZ
User Data RAM Partitions
KSEG0: 0x80000000 +BMXDUDBA
KSEG1: 0xA0000000 +BMXDUDBA
Kernel Program RAM Partition KSEG 0/1
Kernel Program RAM Size
KSEG0: 0x80000000 +BMXDKPBA
KSEG1: 0xA0000000 +BMXDKPBA
Kernel Data RAM Partition KSEG 0/1
Kernel Data RAM Size
KSEG0: 0x80000000
KSEG1: 0xA0000000

Note 1: Kernel Program RAM Size = BMXDUDBA - BMXDKPBA

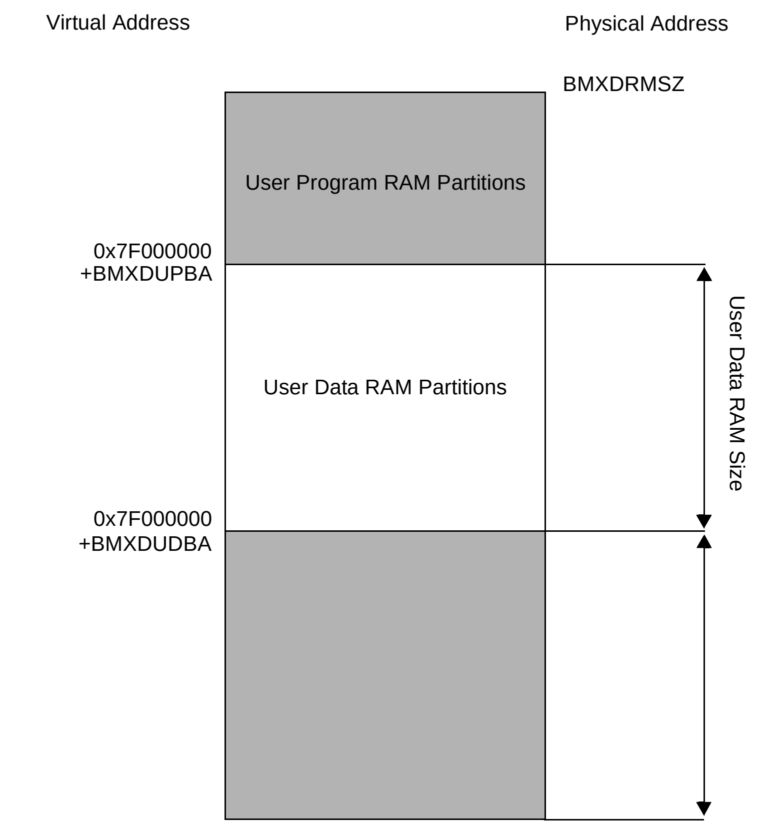
2: None of BMXDKPBA, BMXDUDBA, BMXDUPBA = 0

3.4.3.3 User Data RAM Partition

For User mode applications, a User mode data partition in RAM is required. This partition starts at address 0x7F000000 + BMXDUDBA, and its size is given by BMXDUPBA – BMXDUDBA. See Figure 3-8.

The user data RAM partition does not exist on Reset, as the BMXDUDBA and BMXDUPBA registers default to zero at Reset.

Figure 3-8: User Data RAM Partitioning



Note 1: User Data RAM Size = BMXDUPBA – BMXDUDBA.

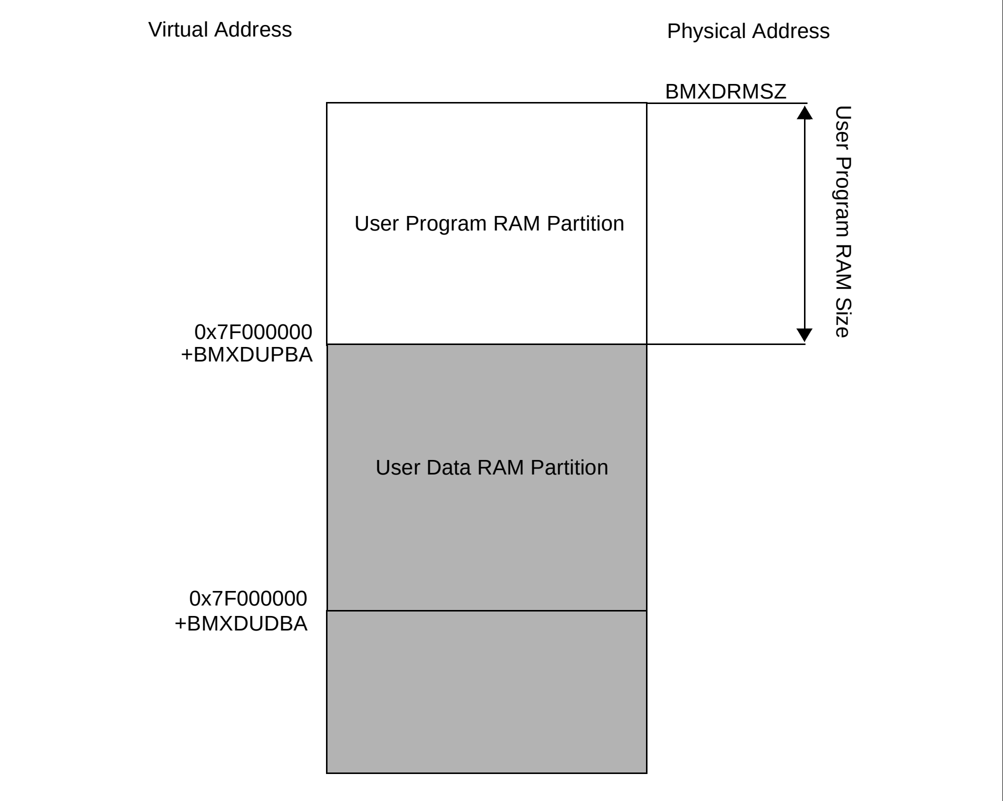
2: None of the registers BMXDKPBA, BMXDUDBA, or BMXDUPBA = `0`.

3.4.3.4 User Program RAM Partition

The user program partition in data RAM is required if code needs to be executed from data RAM in User mode. This partition starts at address 0x7F000000 + BMXDUPBA, and its size is given by BMXDRMSZ – BMXDUPBA. See Figure 3-9.

The User Program RAM partition does not exist on Reset, as the BMXDUPBA register defaults to zero at Reset.

Figure 3-9: User Program RAM Partitioning



Note 1: User Program RAM Size = BMXDRMSZ – BMXDUPBA.

2: None of the registers BMXDKPBA, BMXDUDBA, or BMXDUPBA = `0`.

3.4.3.5 RAM Partitioning Examples

This section provides the following practical examples of RAM partitioning.

1. RAM Partitioned as Kernel Data
2. RAM Partitioned as Kernel Data and Kernel Program
3. RAM Partitioned as Kernel Data and User Data
4. RAM Partitioned as Kernel Data, Kernel Program and User Data
5. RAM Partitioned as Kernel Data, Kernel Program, User Data and User Program

Example 1. RAM Partitioned as Kernel Data

The entire RAM is partitioned as kernel data RAM after a Reset. No other programming is required. Setting the BMXDKPBA, BMXDUDBA, or BMXDUPBA register to ‘`0`’ will partition the entire RAM space to a kernel data partition. See Figure 3-5.

Example 2. RAM Partitioned as Kernel Data and Kernel Program

For this example, assume that the available RAM on the PIC32MX device is 32 KB, of which 8 KB kernel data RAM and 24 KB of kernel program RAM are needed. In this example, the user data RAM and user program RAM will have their sizes set to '0'.

Please note that a kernel data RAM partition is always required. See Figure 3-10 for details.

The values of the registers are as follows:

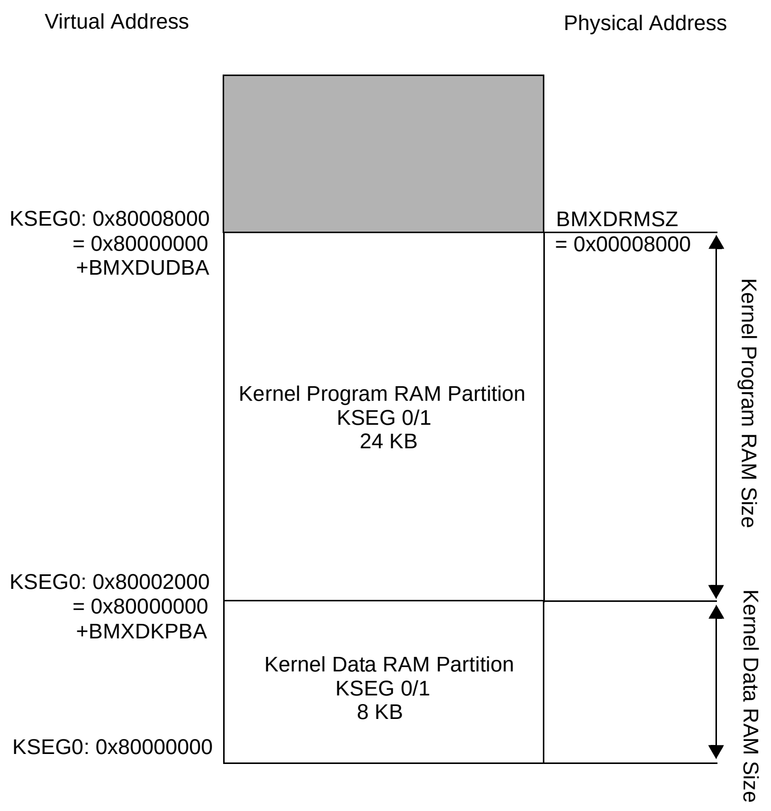
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00002000 (i.e., 8 KB kernel data)

BMXDUDBA = 0x00008000 (i.e., 0x6000 kernel program)

BMXDUPBA = 0x00008000 (i.e., user data size = 0, and user program size = 0)

Figure 3-10: RAM Partitioning for 8 KB Kernel Data and 16 KB Kernel Program



BMXDKPBA = 0x2000
BMXDUDBA = 0x8000
BMXDUPBA = 0x8000

Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA000000.

Example 3. RAM Partitioned as Kernel Data and User Data

For this example, assume that the available RAM on the PIC32MX device is 32 KB, of which 16 KB of kernel data RAM and 16 KB of user data RAM are needed. In this example, the kernel program RAM and user program RAM will have their sizes set to '0'. See Figure 3-11 for details.

The values of the registers are as follows:

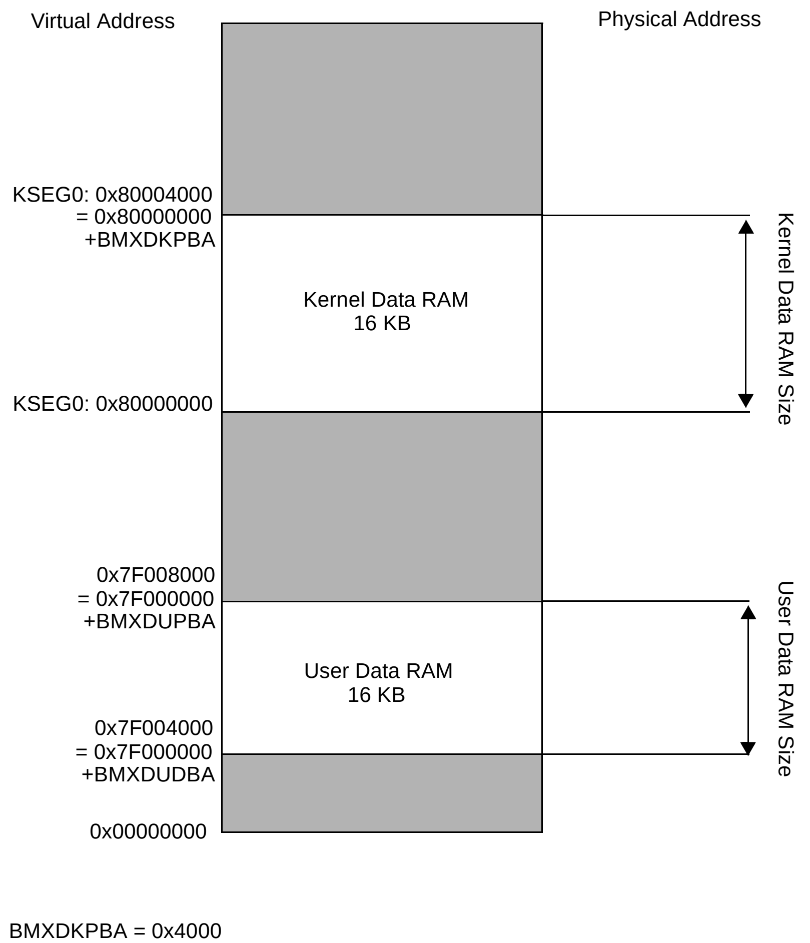
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00004000 (i.e., 16 KB kernel data)

BMXDUDBA = 0x00004000 (i.e., 0 kernel program)

BMXDUPBA = 0x00008000 (i.e., user data size = 16 KB, and user program size = 0)

Figure 3-11: RAM Partitioning for 16 KB Kernel Data and 16 KB User Data



BMXDKPBA = 0x4000
BMXDUDBA = 0x4000
BMXDUPBA = 0x8000

Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA0000000.

Example 4. RAM Partitioned as Kernel Data, Kernel Program and User Data

For this example, assume that the available RAM on the PIC32MX device is 32 KB, and 4 KB of kernel data RAM, 6 KB of kernel program and 22 KB of user data RAM are needed. In this example, the user program RAM will have its size set to '0'. See Figure 3-12 for details.

The values of the registers are as follows:

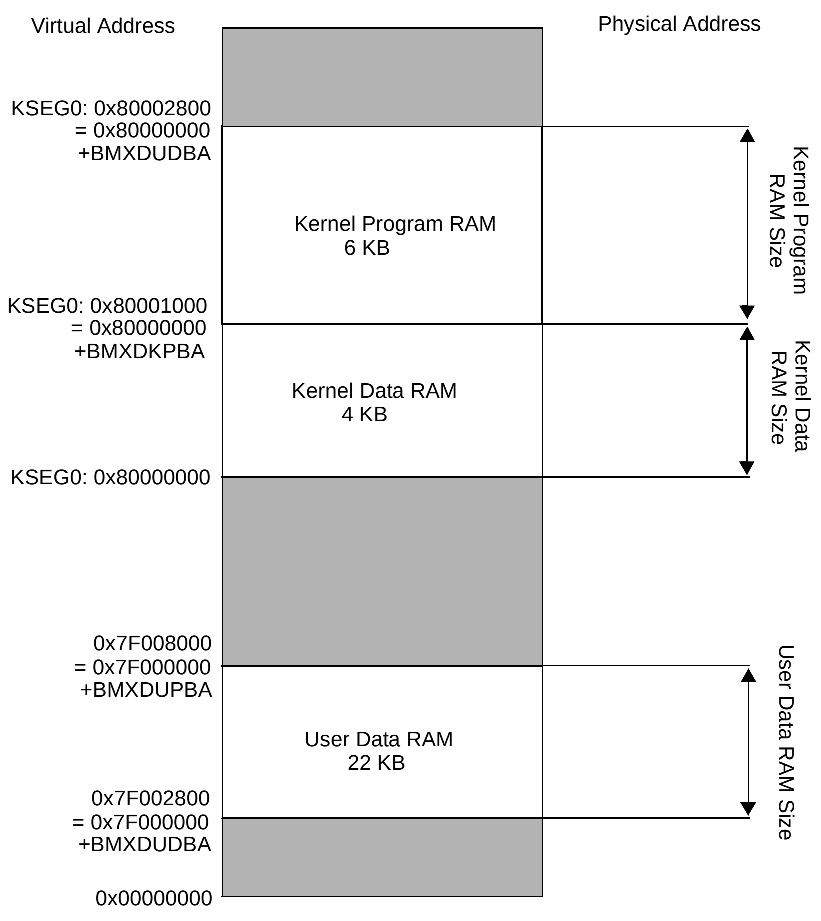
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00001000 (i.e., 4 KB kernel data)

BMXDUDBA = 0x00002800 (i.e., 6 KB kernel program)

BMXDUPBA = 0x00008000 (i.e., user data size = 22 KB, and user program size = 0)

Figure 3-12: RAM Partitioning for 4 KB K-Data, 6 KB K-Program and 22 KB U-Data



BMXDKPBA = 0x1000
BMXDUDBA = 0x2800
BMXDUPBA = 0x8000

Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA0000000.

Example 5. RAM Partitioned as Kernel Data, Kernel Program, User Data and User Program

For this example, assume that the available RAM on the PIC32MX device is 32 KB, and 6 KB of kernel data RAM, 5 KB of kernel program RAM, 12 KB of user data RAM and 9 KB of user program RAM are needed. See Figure 3-13 for details.

The values of the registers are as follows:

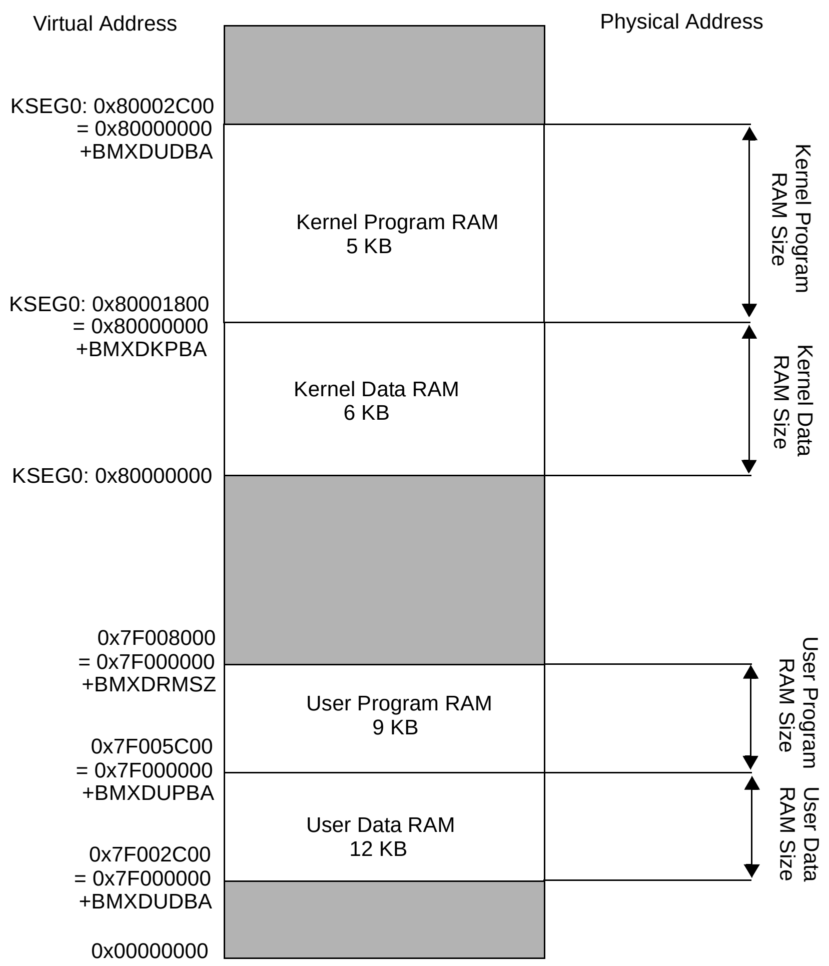
BMXDRMSZ = 0x00008000 (read-only value)

BMXDKPBA = 0x00001800 (i.e., 6 KB kernel data)

BMXDUDBA = 0x00002C00 (i.e., 5 KB kernel program)

BMXDUPBA = 0x00005C00 (i.e., user data size = 12 KB, and user program size = 9 KB)

Figure 3-13: RAM Partitioning for 6 KB K-Data, 5 KB K-Program, 12 KB U-Data and 9 KB U-Program



BMXDKPBA = 0x1800
BMXDUDBA = 0x2c00
BMXDUPBA = 0x5c00

Note: Only KSEG0 addresses are shown. For KSEG1 addresses, start at 0xA0000000.

3.5 BUS MATRIX

The processor supports two modes of operation, Kernel mode and User mode. The Bus Matrix controls the allocation of memory for each of these modes. It also controls the type of access, program or data, for a given region of address space.

The Bus Matrix connects master devices, generically called initiators, to slave devices, generically called targets. The PIC32MX product family can have up to five initiators and three targets (e.g., Flash, RAM, ...) on the main bus structure.

Of the five possible initiators, the CPU Instruction Bus (CPU IS), CPU Data Bus (CPU DS), In-Circuit Debug (ICD) and DMA Controller (DMA) are the default set of initiators and are always present. The PIC32MX also includes an Initiator Expansion Interface (IXI) to support additional initiators for future expansion.

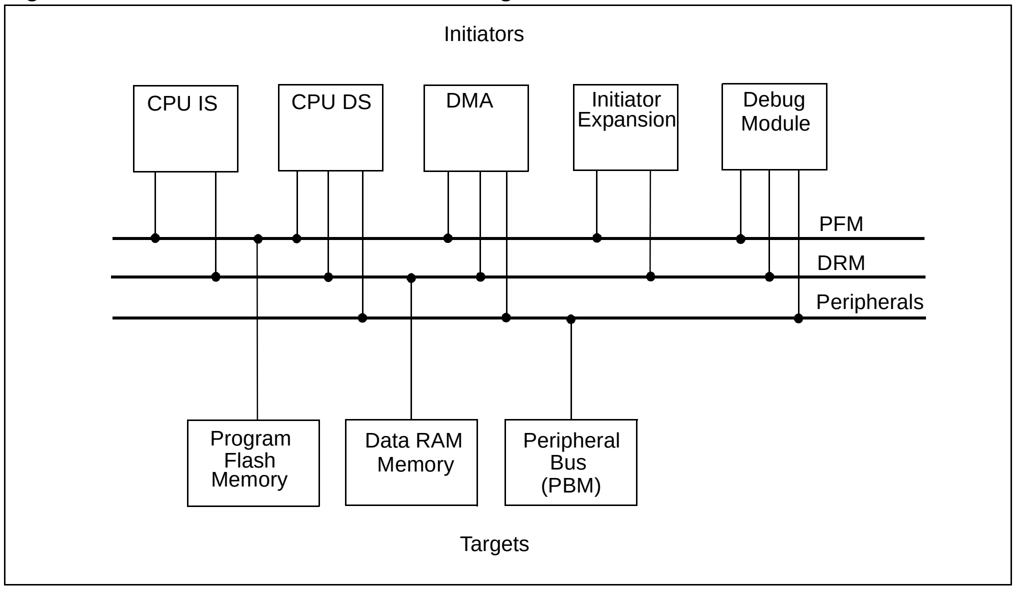
The Bus Matrix decodes a general range of addresses that map to a target. The target (memory or peripherals) may provide additional addresses depending on its functionality.

Table 3-3 shows which initiators can access which targets.

Table 3-3: Initiator Access Map

		Target		
		Flash	**RAM**	**Peripheral Bus**
Initiator	**CPU IS**	Y	Y	N
	CPU DS	Y	Y	Y
	DMA	Y	Y	Y
	IXI	Y	Y	N
	ICD	Y	Y	Y

Figure 3-14: Bus Matrix Initiators and Targets



3.5.1 Initiator Arbitration Modes

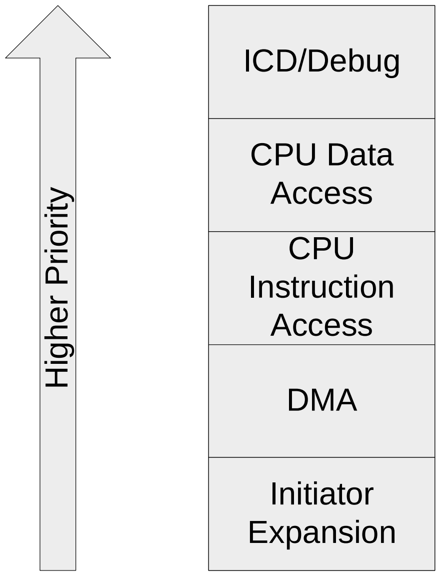
Since there can be more than one initiator attempting to access the same target, an arbitration scheme must be used to control access to the target. The arbitration modes assign priority levels to all the initiators. The initiator with the higher priority level will always win target access over a lower priority initiator.

3.5.1.1 Arbitration Mode 0

The fixed priority scheme in Arbitrition Mode 0 is shown in Figure 3-15. The CPU data and instruction access are given higher priority than DMA access. This mode can starve the DMA, so chose this mode when DMA is not being used.

As shown in Figure 3-15, each initiator is assigned a fixed priority level. Programming the register field BMXARB (BMXCON<2:0>) to ‘0’ selects Mode 0 operation.

Figure 3-15: Priority Assignment in Arbitration Mode 0

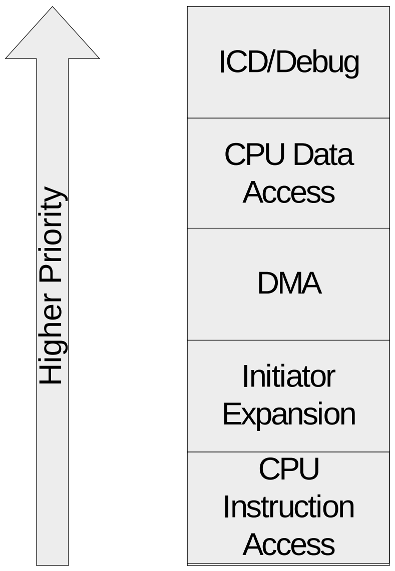


3.5.1.2 Arbitration Mode 1

Arbitrition Mode 1 is a fixed priority scheme like Mode 0; however, the CPU IS is always the lowest priority. Figure 3-16 shows the priority scheme in Mode 1. Mode 1 arbitration is the default mode.

Programming the register field BMXARB (BMXCON<2:0>) to ‘1’ selects Mode 1 operation.

Figure 3-16: Priority Assignment in Arbitration Mode 1

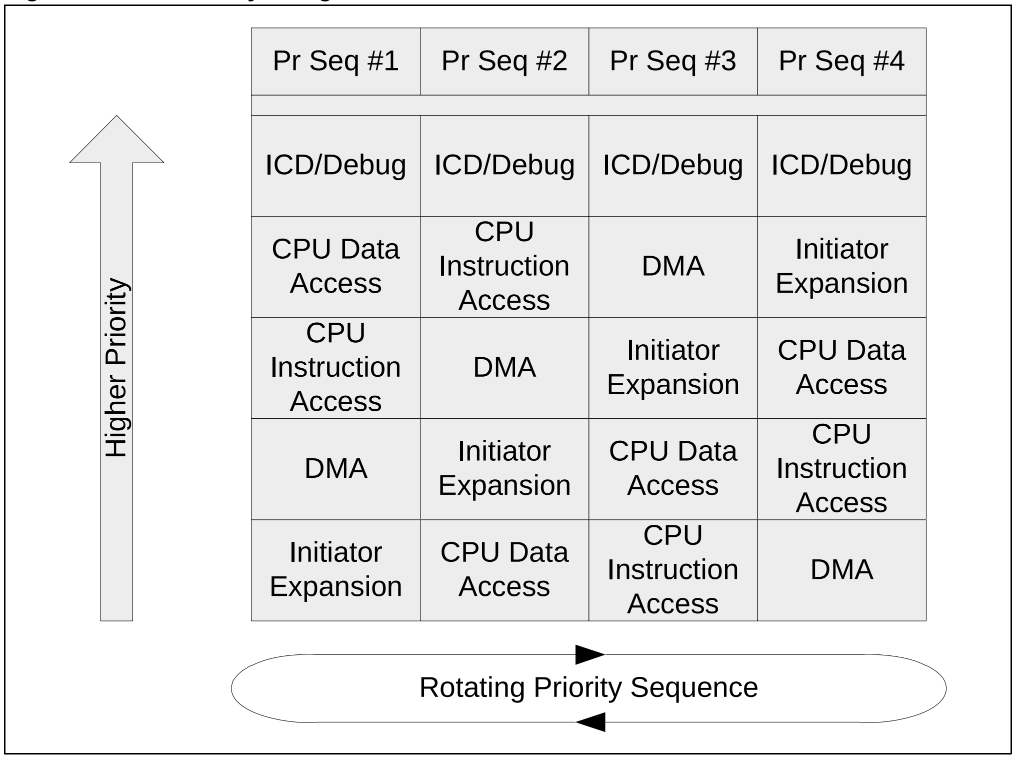


3.5.1.3 Arbitration Mode 2

Mode 2 arbitration supports rotating priority assignments to all initiators. Instead of a fixed priority assignment, each initiator is assigned the highest priority in a rotating fashion. In this mode, the rotating priority is applied with the following exceptions:

1. CPU data is always selected over CPU instruction.
2. ICD is always the highest priority.
3. When the CPU is processing an exception (EXL = 1) or an error (ERL = 1), the arbiter temporarily reverts to Mode 0.

Figure 3-17: Priority Assignments in Arbitration Mode 2



Pr Seq #1	Pr Seq #2	Pr Seq #3	Pr Seq #4
ICD/Debug	ICD/Debug	ICD/Debug	ICD/Debug
CPU Data Access	CPU Instruction Access	DMA	Initiator Expansion
CPU Instruction Access	DMA	Initiator Expansion	CPU Data Access
DMA	Initiator Expansion	CPU Data Access	CPU Instruction Access
Initiator Expansion	CPU Data Access	CPU Instruction Access	DMA

Note that priority sequence #2 is not selected in the rotating priority scheme if there is a pending CPU data access. In this case, once the data access is complete, sequence #2 is selected.

Programming the register field BMXARB (BMXCON<2:0>) to ‘2’ selects Mode 2 operation.

3.5.2 Bus Error Exceptions

The Bus Matrix generates a bus error exception on:

- Any attempt to access unimplemented memory
- Any attempt to access an illegal target
- Any attempt to write to program Flash memory

Bus Error Exceptions may be temporarily disabled by clearing the BMXERRxxx bits in the BMXCON register. This is not recommended.

The Bus Matrix disables bus error exceptions for accesses from CPU IS and CPU DS while in DEBUG mode.

3.5.3 Break Exact Breakpoint Support

The PIC32MX supports break exact breakpoints by inserting one Wait state to data RAM access. This method allows the CPU to stop execution just before the breakpoint address instruction. This is useful in case of breakpointed store instructions. When the Wait state is not used, the break will still occur at the store instruction, however, the DRM location is updated with the store value. If the Wait state is enabled the DRM is not updated with the store value.

3.6 I/O PIN CONTROL

There are no pins associated with this module.

3.7 OPERATION IN POWER-SAVING AND DEBUG MODES

Note:	In this manual, a distinction is made between a power mode as it is used in a specific module, and a power mode as it is used by the device, e.g., Sleep mode of the Comparator and SLEEP mode of the CPU. To indicate which type of power mode is intended, uppercase and lowercase letters (Sleep, Idle, Debug) signify a module power mode, and all uppercase letters (SLEEP, IDLE, DEBUG) signify a device power mode.

3.7.1 Memory Operation on Power-up or Brown-out Reset:

- The contents of data RAM are undefined.
- The BMXxxxBA registers are reset to '`0`'.
- CPU is switched to Kernel mode.

3.7.2 Memory Operation on Reset:

- The data RAM contents are retained. If the device is code-protected, the RAM contents are cleared.
- The BMX base address registers (BMXxxxBA) are set to '`0`'.
- CPU is switched to Kernel mode.

3.7.3 Memory Operation on Wake-up from SLEEP or IDLE Mode:

- The RAM contents are retained.
- The BMX base address register (BMXxxxBA) contents are not changed.
- CPU mode is unchanged.

3.8 CODE EXAMPLES

Example 3-1: Create a User Mode Partition of 12K in Program Flash

```
BMXPUPBA = BMXPFMSZ - (12*1024);  // User Mode Flash 12K,
                                  // Kernel Mode Flash 500K (512K-12K)
```

Example 3-2: Create a Kernel Mode Data RAM Partition of 16K; Rest of RAM for Kernel Program

```
BMXDKPBA = 16*1024;
BMXDUDBA = BMXDRMSZ;
BMXDUPBA = BMXDRMSZ;
```

Example 3-3 can be used to create the following partitions in RAM:

Kernel mode data = 12K

Kernel mode program = 6K

User mode data = 8K

User mode program = 6K

Example 3-3: Create RAM Partitions

```
BMXDKPBA = 12*1024;               // Kernel Data Partition of 12K.
                                  // Start offset of Kernel Program Partition
BMXDUDBA = BMXDKPBA + (6*1024);   // Kernel Program Partition of 6K
                                  // Start offset of User Data Partition
BMXDUPBA = BMXDUDBA + (8*1024);   // User Data Partition of 8K
                                  // Start offset of User Program Partition.
                                  // This partition will go up to the size of
                                  // RAM (32K). So the partition size will be
                                  // 6K (32K - 8K - 6K - 12K)
```

3.9 DESIGN TIPS

Question 1: ***At Reset, which mode is the CPU running in?***

Answer: The CPU starts in Kernel mode. The entire RAM is mapped to kernel data segments in KSEG0 and KSEG1. Flash memory is mapped to kernel program segments in KSEG0 and KSEG1. Also ERL = 1, which should be reset to zero (normally in the C start-up code).

Question 2 ***Do I need to initialize the BMX registers?***

Answer: Generally, no. You can leave the BMX registers at their default values, which allows maximum RAM and Flash memories for Kernel mode applications. If you want to run code from RAM or set up User mode partitions, you will need to configure the BMX registers.

Question 3 ***What is the CPU Reset vector address?***

Answer: The CPU Reset address is 0xBFC00000.

Question 4 ***What is a Bus-Error Exception?***

Answer: The bus-error exceptions are generated when the CPU tries to access unimplemented addresses. Also, when the CPU tries to execute a program from RAM without defining a RAM program partition, a bus-error exception is generated.

3.10 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 PIC32MX device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Memory Organization of the PIC32MX family include the following:

Title	**Application Note #**
No related application notes at this time.	N/A

Note:	Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC32MX family of devices.

3.11 REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (October 2007)

Updated document to remove Confidential status.

Revision C (April 2008)

Revised status to Preliminary; Revised U-0 to r-x.

Revision D (June 2008)

Change Reserved bits from “Maintain as” to “Write”.

NOTES: