Section 3. Data Memory

HIGHLIGHTS

This section of the manual contains the following topics:

3.1 Introduction

The dsPIC30F data width is 16-bits. All internal registers and data space memory are organized as 16-bits wide. The dsPIC30F features two data spaces. The data spaces can be accessed separately (for some DSP instructions) or together as one 64-Kbyte linear address range (for MCU instructions). The data spaces are accessed using two Address Generation Units (AGUs) and separate data paths.

An example data space memory map is shown in Figure 3-1.

Data memory addresses between `0x0000` and `0x07FF` are reserved for the device special function registers (SFRs). The SFRs include control and status bits for the CPU and peripherals on the device.

The RAM begins at address `0x0800` and is split into two blocks, X and Y data space. For data writes, the X and Y data spaces are always accessed as a single, linear data space. For data reads, the X and Y memory spaces can be accessed independently or as a single, linear space. Data reads for MCU class instructions always access the the X and Y data spaces as a single combined data space. Dual source operand DSP instructions, such as the `MAC` instruction, access the X and Y data spaces separately to support simultaneous reads for the two source operands.

MCU instructions can use any W register as an address pointer for a data read or write operation.

During data reads, the DSP class of instructions isolates the Y address space from the total data space. W10 and W11 are used as address pointers for reads from the Y data space. The remaining data space is referred to as X space, but could more accurately be described as “X minus Y” space. W8 and W9 are used as address pointers for data reads from the X data space in DSP class instructions.

Figure 3-2 shows how the data memory map functions for both MCU class and DSP class instructions. Note that it is the W register number and type of instruction that determines how address space is accessed for data reads. In particular, MCU instructions treat the X and Y memory as a single combined data space. The MCU instructions can use any W register as an address pointer for reads and writes. The DSP instructions that can simultaneously pre-fetch two data operands, split the data memory into two spaces. Specific W registers must be used for read address pointers in this case.

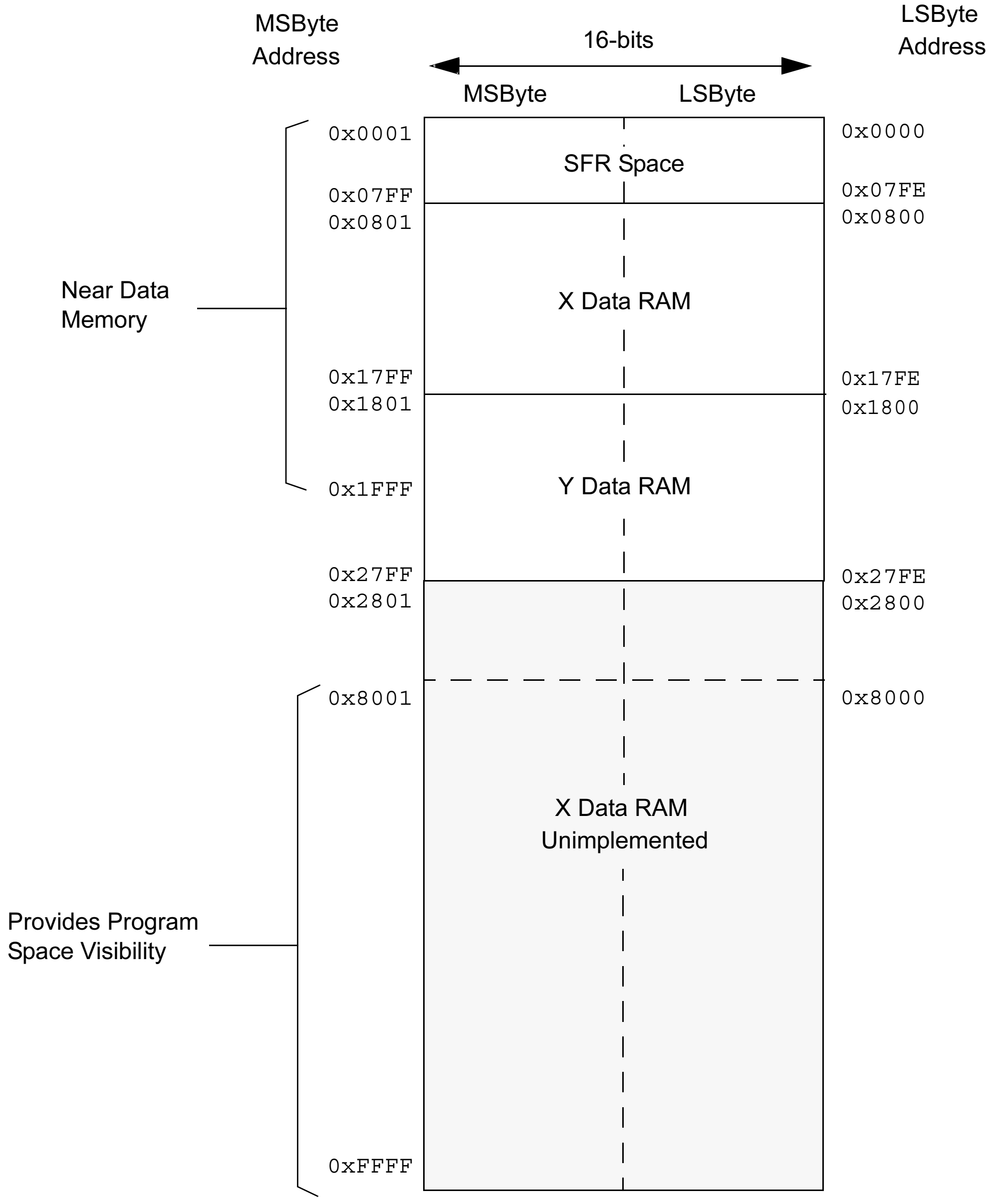
Some DSP instructions have the ability to store the accumulator that is not targeted by the instruction to data memory. This function is called “accumulator write back”. W13 must be used as an address pointer to the combined data memory space for accumulator write back operations.

For DSP class instructions, W8 and W9 should point to implemented X memory space for all memory reads. If W8 or W9 points to Y memory space, zeros will be returned. If W8 or W9 points to an unimplemented memory address, an address error trap will be generated.

For DSP class instructions, W10 and W11 should point to implemented Y memory space for all memory reads. If W10 or W11 points to implemented X memory space, all zeros will be returned. If W10 or W11 points to an unimplemented memory address, an address error trap will be generated. For additional information on address error traps, refer to **Section 6. “Reset Interrupts”**.

Note: The data memory map and the partition between the X and Y data spaces is device specific. Refer to the specific dsPIC30F device data sheet for further details.

Figure 3-1: Example Data Memory Map



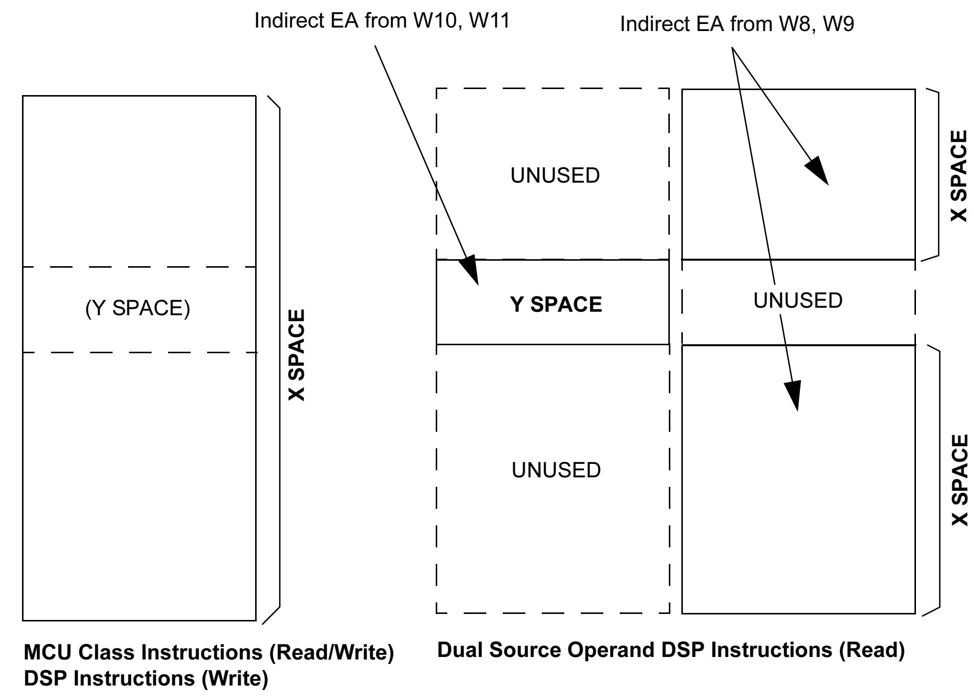
Note 1: The partition between the X and Y data spaces is device specific. Refer to the appropriate device data sheet for further details. The data space boundaries indicated here are used for example purposes only.

2: Near data memory can be accessed directly via file register instructions that encode a 13-bit address into the opcode. At a minimum, the near data memory region overlaps all of the SFR space and a portion of X memory space. All of X memory space and some or all of Y memory space may be included in the near data memory region, depending on the device variant.

3: All data memory can be accessed indirectly via W registers or directly using the `MOV` instruction.

4: Upper half of data memory map can be mapped into a segment of program memory space for program space visibility.

Figure 3-2: Data Spaces for MCU and DSP Instructions


Indirect EA from W10, W11
Indirect EA from W8, W9
(Y SPACE)
X SPACE
UNUSED
Y SPACE
UNUSED
UNUSED
X SPACE
X SPACE
MCU Class Instructions (Read/Write)
DSP Instructions (Write)
Dual Source Operand DSP Instructions (Read)

Note: Data writes for DSP instructions consider the entire data memory as one combined space. DSP instructions that perform an accumulator write back use W13 as an address pointer for writes to the combined data spaces.

3.1.1 Near Data Memory

An 8-Kbyte address space, referred to as near data memory, is reserved in the data memory space between `0x0000` and `0x1FFF`. Near data memory is directly addressable via a 13-bit absolute address field within all file register instructions.

The memory regions included in the near data region will depend on the amount of data memory implemented for each dsPIC30F device variant. At a minimum, the near data region will include all of the SFRs and some of the X data memory. For devices that have smaller amounts of data memory, the near data region may include all of X memory space and possibly some or all of Y memory space. Refer to Figure 3-1 for more details.

Note: The entire 64K data space can be addressed directly using the `MOV` instruction. Refer to the dsPIC30F Programmer's Reference Manual (DS70030) for further details.

3.2 Data Space Address Generator Units (AGUs)

The dsPIC30F contains an X AGU and a Y AGU for generating data memory addresses. Both X and Y AGUs can generate any effective address (EA) within a 64-Kbyte range. However, EAs that are outside the physical memory provided will return all zeros for data reads and data writes to those locations will have no effect. Furthermore, an address error trap will be generated. For more information on address error traps, refer to **Section 6. "Reset Interrupts"**.

3.2.1 X Address Generator Unit

The X AGU is used by all instructions and supports all Addressing modes. The X AGU consists of a read AGU (X RAGU) and a write AGU (X WAGU), which operate independently on separate read and write buses during different phases of the instruction cycle. The X read data bus is the return data path for all instructions that view data space as combined X and Y address space. It is also the X address space data path for the dual operand read instructions (DSP instruction class). The X write data bus is the only write path to the combined X and Y data space for all instructions.

The X RAGU starts its effective address calculation during the prior instruction cycle, using information derived from the just pre-fetched instruction. The X RAGU EA is presented to the address bus at the beginning of the instruction cycle.

The X WAGU starts its effective address calculation at the beginning of the instruction cycle. The EA is presented to the address bus during the write phase of the instruction.

Both the X RAGU and the X WAGU support modulo addressing.

Bit-reversed addressing is supported by the X WAGU only.

3.2.2 Y Address Generator Unit

The Y data memory space has one AGU that supports data reads from the Y data memory space. The Y memory bus is never used for data writes. The function of the Y AGU and Y memory bus is to support concurrent data reads for DSP class instructions.

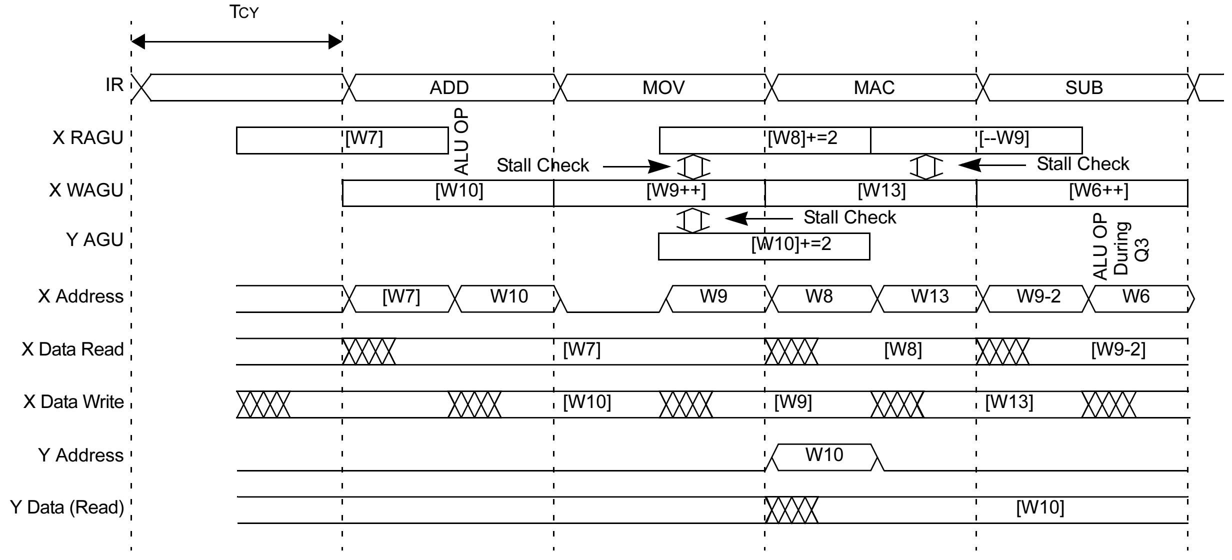
The Y AGU timing is identical to that of the X RAGU, in that its effective address calculation starts prior to the instruction cycle, using information derived from the pre-fetched instruction. The EA is presented to the address bus at the beginning of the instruction cycle.

The Y AGU supports Modulo Addressing and Post-modification Addressing modes for the DSP class of instructions that use it.

Note: The Y AGU does not support data writes. All data writes occur via the X WAGU to the combined X and Y data spaces. The Y AGU is only used during data reads for dual source operand DSP instructions.

Figure 3-3: Data Space Access Timing

```
ADD.W  W0, [W7], [W10]
MOV.W  W10, [W9++]
MAC    W4*W5, A, W4, [W8]+=2, W5, [W10]+=2, [W13]+=2
SUB.W  W4, [--W9], [W6++]
```



3.2.3 Address Generator Units and DSP Class Instructions

The Y AGU and Y memory data path are used in concert with the X RAGU by the DSP class of instructions to provide two concurrent data read paths. For example, the `MAC` instruction can simultaneously pre-fetch two operands to be used in the next multiplication.

The DSP class of instructions dedicates two W register pointers, W8 and W9, to always operate through the X RAGU and address X data space independently from Y data space, plus two W register pointers, W10 and W11, to always operate through the Y AGU and address Y data space independently from X data space. Any data write performed by a DSP class instruction will take place in the combined X and Y data space and the write will occur across the X-bus. Consequently, the write can be to any address irrespective of where the EA is directed.

The Y AGU only supports Post-modification Addressing modes associated with the DSP class of instructions. For more information on Addressing modes, please refer to the dsPIC30F Programmer's Reference Manual. The Y AGU also supports modulo addressing for automated circular buffers. All other (MCU) class instructions can access the Y data address space through the X AGU when it is regarded as part of the composite linear space.

3.2.4 Data Alignment

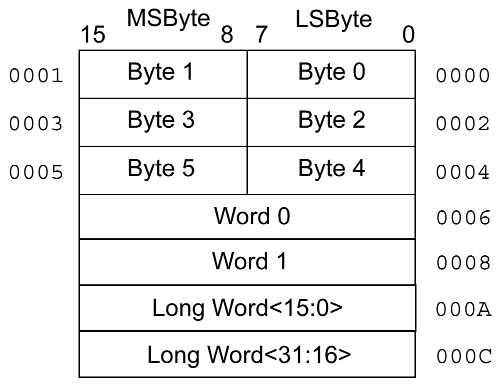
The ISA supports both word and byte operations for all MCU instructions that access data through the X memory AGU. The LSb of a 16-bit data address is ignored for word operations. Word data is aligned in the little-endian format with the LSByte at the even address (LSB = `0`) and the MSByte at the odd address (LSB = `1`).

For byte operations, the LSB of the data address is used to select the byte that is accessed. The addressed byte is placed on the lower 8 bits of the internal data bus.

All effective address calculations are automatically adjusted depending on whether a byte or a word access is performed. For example, an address will be incremented by 2 for a word operation that post-increments the address pointer.

Note: All word accesses must be aligned to an even address (LSB = `0`). Misaligned word data fetches are not supported, so care must be taken when mixing byte and word operations or translating from existing PICmicro code. Should a misaligned word read or write be attempted, an address error trap will occur. A misaligned read operation will complete, but a misaligned write will not take place. The trap will then be taken, allowing the system to examine the machine state prior to execution of the address Fault.

Figure 3-4: Data Alignment



3.3 Modulo Addressing

Modulo, or circular addressing provides an automated means to support circular data buffers using hardware. The objective is to remove the need for software to perform data address boundary checks when executing tightly looped code as is typical in many DSP algorithms.

Any W register, except W15, can be selected as the pointer to the modulo buffer. The modulo hardware performs boundary checks on the address held in the selected W register and automatically adjusts the pointer value at the buffer boundaries, when required.

dsPIC30F modulo addressing can operate in either data or program space (since the data pointer mechanism is essentially the same for both). One circular buffer can be supported in each of the X (which also provides the pointers into Program space) and Y data spaces.

The modulo data buffer length can be any size up to 32K words. The modulo buffer logic supports buffers using word or byte sized data. However, the modulo logic only performs address boundary checks at word address boundaries, so the length of a byte modulo buffer must be even. In addition, byte-sized modulo buffers cannot be implemented using the Y AGU because byte access is not supported via the Y memory data bus.

3.3.1 Modulo Start and End Address Selection

Four address registers are available for specifying the modulo buffer start and end addresses:

- XMODSRT: X AGU Modulo Start Address Register
- XMODEND: X AGU Modulo End Address Register
- YMODSRT: Y AGU Modulo Start Address Register
- YMODEND: Y AGU Modulo End Address Register

The start address for a modulo buffer must be located at an even byte address boundary. The LSB of the XMODSRT and YMODSRT registers is fixed at '0' to ensure the correct modulo start address. The end address for a modulo buffer must be located at an odd byte address boundary. The LSB of the XMODEND and YMODEND registers is fixed to '1' to ensure the correct modulo end address.

The start and end address selected for each modulo buffer have certain restrictions, depending on whether an incrementing or decrementing buffer is to be implemented. For an incrementing buffer, a W register pointer is incremented through the buffer address range. When the end address of the incrementing buffer is reached, the W register pointer is reset to point to the start of the buffer. For a decrementing buffer, a W register pointer is decremented through the buffer address range. When the start address of a decrementing buffer is reached, the W register pointer is reset to point to the end of the buffer.

Note:	The user must decide whether an incrementing or decrementing modulo buffer is required for the application. There are certain address restrictions that depend on whether an incrementing or decrementing modulo buffer is to be implemented.

3.3.1.1 Modulo Start Address

The data buffer start address is arbitrary, but must be at a 'zero' power of two boundary for incrementing modulo buffers. The modulo start address can be any value for decrementing modulo buffers and is calculated using the chosen buffer end address and buffer length.

For example, if the buffer length for an incrementing buffer is chosen to be 50 words (100 bytes), then the buffer start byte address must contain 7 Least Significant zeros. Valid start addresses may, therefore, be `0xNN00` and `0xNN80`, where 'N' is any hexadecimal value.

3.3.1.2 Modulo End Address

The data buffer end address is arbitrary but must be at a 'ones' boundary for decrementing buffers. The modulo end address can be any value for an incrementing buffer and is calculated using the chosen buffer start address and buffer length.

For example, if the buffer size (modulus value) is chosen to be 50 words (100 bytes), then the buffer end byte address for decrementing modulo buffer must contain 7 Least Significant ones. Valid end addresses may, therefore, be `0xNNFF` and `0xNN7F`, where 'x' is any hexadecimal value.

Note:	If the required modulo buffer length is an even power of 2, modulo start and end addresses can be chosen that satisfy the requirements for incrementing and decrementing buffers.

3.3.1.3 Modulo Address Calculation

The end address for an incrementing modulo buffer must be calculated from the chosen start address and the chosen buffer length in bytes. Equation 3-1 may be used to calculate the end address.

Equation 3-1: Modulo End Address for Incrementing Buffer

$$\text{End Address} = \text{Start Address} + \text{Buffer Length} - 1$$

The start address for a decrementing modulo buffer is calculated from the chosen end address and the buffer length, as shown in Equation 3-2.

Equation 3-2: Modulo Start Address for Decrementing Buffer

$$\text{Start Address} = \text{End Address} - \text{Buffer Length} + 1$$

3.3.1.4 Data Dependencies Associated with Modulo Addressing SFRs

A write operation to the Modulo Addressing Control register, MODCON, should not be immediately followed by an indirect read operation using any W register. The code segment shown in Example 3-1 will thus lead to unexpected results.

Note 1: Using a `POP` instruction to pop the contents of the top-of-stack (TOS) location into MODCON, also constitutes a write to MODCON. The instruction immediately following a write to MODCON cannot be any instruction performing an indirect read operation.

2: The user should note that some instructions perform an indirect read operation, implicitly. These are: `POP`, `RETURN`, `RETFIE`, `RETLW` and `ULNK`.

Example 3-1: Incorrect MODCON Initialization

```
MOV   #0x8FF4, w0    ;Initialize MODCON
MOV   w0, MODCON
MOV   [w1], w2       ;Incorrect EA generated here
```

To work around this problem of initialization, use any Addressing mode other than indirect reads in the instruction that immediately follows the initialization of MODCON. A simple work around to the problem is achieved by adding a `NOP` after initializing MODCON, as shown in Example 3-2.

Example 3-2: Correct MODCON Initialization

```
MOV   #0x8FF4, w0    ;Initialize MODCON
MOV   w0, MODCON
NOP                  ;See Note below
MOV   [w1], w2       ;Correct EA generated here
```

An additional condition exists for indirect read operations performed immediately after writing to the modulo address SFRs:

- XMODSRT
- XMODEND
- YMODSRT
- YMODEND

If modulo addressing has already been enabled in MODCON, then a write to the X (or Y) modulo address SFRs should not be immediately followed by an indirect read, using the W register designated for modulo buffer access from X-data space (or Y-data space). The code segment in Example 3-3 shows how initializing the modulo SFRs associated with the X-data space, could lead to unexpected results. A similar example can be made for initialization in Y-data space.

Example 3-3: Incorrect Modulo Addressing Setup

```
MOV   #0x8FF4,   w0      ;Modulo addressing enabled
MOV   w0, MODCON         ;in X-data space using w4
                         ;for buffer access
MOV   #0x1200,   w4      ;XMODSRT is initialized
MOV   w4, XMODSRT
MOV   #0x12FF,   w0      ;XMODEND is initialized
MOV   w0, XMODEND
MOV   [w4++],    w5      ;Incorrect EA generated
```

To work around this issue, insert a `NOP`, or perform any operation other than an indirect read that uses the W register designated for modulo buffer access, after initializing the modulo address SFRs. This is demonstrated in Example 3-4. Another alternative would be to enable modulo addressing in MODCON after initializing the modulo start and end address SFRs.

Example 3-4: Correct Modulo Addressing Setup

```
MOV   #0x8FF4,   w0      ;Modulo addressing enabled
MOV   w0, MODCON         ;in X-data space using w4
                         ;for buffer access
MOV   #0x1200,   w4      ;XMODSRT is initialized
MOV   w4, XMODSRT
MOV   #0x12FF,   w0      ;XMODEND is initialized
MOV   w0, XMODEND
NOP                      ;See Note below
MOV   [w4++],    w5      ;Correct EA generated here
```

Note: Alternatively, execute other instructions that do not perform indirect read operations, using the W register designated for modulo buffer access.

3.3.2 W Address Register Selection

The X address space pointer W register (XWM) to which modulo addressing is to be applied, is stored in MODCON<3:0> (see Register 3-1). The XMODSRT, XMODEND, and the XWM register selection are shared between the X RAGU and X WAGU. Modulo addressing is enabled for X data space when XWM is set to any value other than 15 and the XMODEN bit is set (MODCON<15>). W15 cannot be used as the pointer for modulo addressing because it is the dedicated software stack pointer.

The Y address space pointer W register (YWM) to which modulo addressing is to be applied, is stored in MODCON<7:4> (see Register 3-2). Modulo addressing is enabled for Y data space when YWM is set to any value other than 15 and the YMODEN bit is set (MODCON<14>).

Note: A write to the MODCON register should not be followed by an instruction that performs an indirect read operation using a W register. Unexpected results may occur. Some instructions perform an implicit indirect read. These are:`POP`, `RETURN`, `RETFIE`, `RETLW` and `ULNK`.

3.3.3 Modulo Addressing Applicability

Modulo addressing can be applied to the effective address (EA) calculation associated with the selected W register. It is important to realize that the address boundary tests look for addresses equal to or greater than the upper address boundary for incrementing buffers and equal to or less than the lower address boundary for decrementing buffers. Address changes may, therefore, jump over boundaries and still be adjusted correctly. Remember that the automatic adjustment of the W register pointer by the modulo hardware is uni-directional. That is, the W register pointer may not be adjusted correctly by the modulo hardware when the W register pointer for an incrementing buffer is decremented and vice versa. The exception to this rule is when the buffer length is an even power of 2 and the start and end addresses can be chosen to meet the -boundary requirements for both incrementing and decrementing modulo buffers.

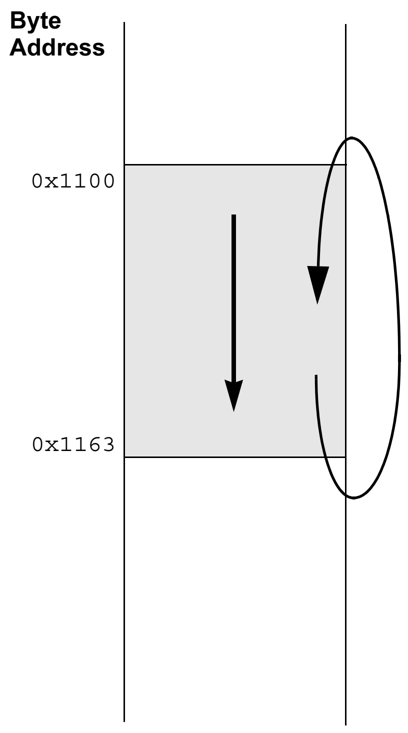
A new EA can exceed the modulo buffer boundary by up to the length of the buffer and still be successfully corrected. This is important to remember when the Register Indexed (`[Wb + Wn]`) and Literal Offset (`[Wn + lit10]`) Addressing modes are used. The user should remember that the Register Indexed and Literal Offset Addressing modes do not change the value held in the W register. Only the indirect with Pre- and Post-modification Addressing modes (`[Wn++]`, `[Wn--]`, `[++Wn]`, `[--Wn]`) will modify the W register address value.

3.3.4 Modulo Addressing Initialization for Incrementing Modulo Buffer

The following steps describe the setup procedure for an incrementing circular buffer. The steps are similar whether the X AGU or Y AGU is used.

1. Determine the buffer length in 16-bit data words. Multiply this value by 2 to get the length of the buffer in bytes.
2. Select a buffer starting address that is located at a binary ‘zeros’ boundary based on the desired length of the buffer. Remember that the buffer length in words must be multiplied by 2 to obtain the byte address range. For example, a buffer with a length of 100 words (200 bytes) could use `0xXX00` as the starting address.
3. Calculate the buffer end address using the buffer length chosen in Step 1 and the buffer start address chosen in Step 2. The buffer end address is calculated using Equation 3-1.
4. Load the XMODSRT (YMODSRT) register with the buffer start address chosen in Step 2.
5. Load the XMODEND (YMODEND) register with the buffer end address calculated in Step 3.
6. Write to the XWM<3:0> (YWM<3:0>) bits in the MODCON register to select the W register that will be used to access the circular buffer.
7. Set the XMODEN (YMODEN) bit in the MODCON register to enable the circular buffer.
8. Load the selected W register with address that points to the buffer.
9. The W register address will be adjusted automatically at the end of the buffer when an indirect access with pre/post increment is performed (see Figure 3-5).

Figure 3-5: Incrementing Buffer Modulo Addressing Operation Example



```
MOV    #0x1100,W0
MOV    W0,XMODSRT     ;set modulo start address
MOV    #0x1163,W0
MOV    W0,XMODEND     ;set modulo end address
MOV    #0x8001,W0
MOV    W0,MODCON      ;enable W1, X AGU for modulo
MOV    #0x0000,W0     ;W0 holds buffer fill value
MOV    #0x1100,W1     ;point W1 to buffer
DO     #49,FILL       ;fill the 50 buffer locations
FILL:
MOV    W0,[W1++]      ;fill the next location

;W1 = 0x1100 when DO loop completes
```

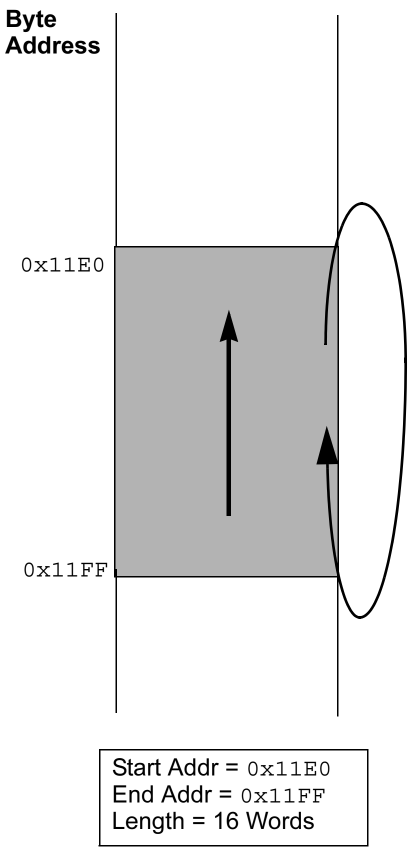
Start Addr = `0x1100`
End Addr = `0x1163`
Length = 50 Words

3.3.5 Modulo Addressing Initialization for Decrementing Modulo Buffer

The following steps describe the setup procedure for a decrementing circular buffer. The steps are similar whether the X AGU or Y AGU is used.

1. Determine the buffer length in 16-bit data words. Multiply this value by 2 to get the length of the buffer in bytes.
2. Select a buffer end address that is located at a binary ‘ones’ boundary, based on the desired length of the buffer. Remember that the buffer length in words must be multiplied by 2 to obtain the byte address range. For example, a buffer with a length of 128 words (256 bytes) could use `0xXXFF` as the end address.
3. Calculate the buffer start address using the buffer length chosen in Step 1 and the end address chosen in Step 2. The buffer start address is calculated using Equation 3-2.
4. Load the XMODSRT (YMODSRT) register with the buffer start address chosen in Step 3.
5. Load the XMODEND (YMODEND) register with the buffer end address chosen in Step 2.
6. Write to the XWM<3:0> (YWM<3:0>) bits in the MODCON register to select the W register that will be used to access the circular buffer.
7. Set the XMODEN (YMODEN) bit in the MODCON register to enable the circular buffer.
8. Load the selected W register with address that points to the buffer.
9. The W register address will be adjusted automatically at the end of the buffer when an indirect access with pre/post-decrement is performed (see Figure 3-6).

Figure 3-6: Decrementing Buffer Modulo Addressing Operation Example



```
MOV     #0x11E0,W0
MOV     W0,XMODSRT     ;set modulo start address
MOV     #0x11FF,W0
MOV     W0,XMODEND     ;set modulo end address
MOV     #0x8001,W0
MOV     W0,MODCON      ;enable W1, X AGU for modulo
MOV     #0x000F,W0     ;W0 holds buffer fill value
MOV     #0x11FE,W1     ;point W1 to buffer
DO      #15,FILL       ;fill the 16 buffer locations
MOV     W0,[W1--]      ;fill the next location

FILL:
DEC     W0,W0          ;decrement the fill value

; W1 = 0x11FE when DO loop completes
```

Start Addr = `0x11E0`
End Addr = `0x11FF`
Length = 16 Words

3.4 Bit-Reversed Addressing

3.4.1 Introduction to Bit-Reversed Addressing

Bit-reversed addressing simplifies data re-ordering for radix-2 FFT algorithms. It is supported through the X WAGU only. Bit-reversed addressing is accomplished by effectively creating a 'mirror image' of an address pointer by swapping the bit locations around the center point of the binary value, as shown in Figure 3-7. An example bit-reversed sequence for a 4-bit address field is shown in Table 3-1.

Figure 3-7: Bit-Reversed Address Example

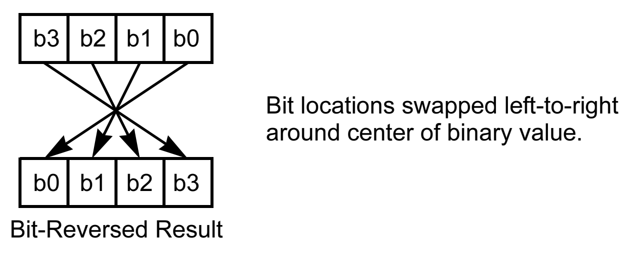

b3
b2
b1
b0
b0
b1
b2
b3
Bit-Reversed Result
Bit locations swapped left-to-right around center of binary value.

Table 3-1: Bit-Reversed Address Sequence (16-Entry)

Normal Address					Bit-Reversed Address				
A3	A2	A1	A0	decimal	A3	A2	A1	A0	decimal
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	8
0	0	1	0	2	0	1	0	0	4
0	0	1	1	3	1	1	0	0	12
0	1	0	0	4	0	0	1	0	2
0	1	0	1	5	1	0	1	0	10
0	1	1	0	6	0	1	1	0	6
0	1	1	1	7	1	1	1	0	14
1	0	0	0	8	0	0	0	1	1
1	0	0	1	9	1	0	0	1	9
1	0	1	0	10	0	1	0	1	5
1	0	1	1	11	1	1	0	1	13
1	1	0	0	12	0	0	1	1	3
1	1	0	1	13	1	0	1	1	11
1	1	1	0	14	0	1	1	1	7
1	1	1	1	15	1	1	1	1	15

3.4.2 Bit-Reversed Addressing Operation

Bit-reversed addressing is only supported by the X WAGU and is controlled by the MODCON and XBREV special function registers. Bit-reversed addressing is invoked as follows:

1. Bit-reversed addressing is assigned to one of the W registers using the BWM control bits (MODCON<11:8>).
2. Bit-reversed addressing is enabled by setting the BREN control bit (XBREV<15>).
3. The X AGU bit-reverse modifier is set via the XB control bits (XBREV<14:0>).

When enabled, the bit-reversed addressing hardware will generate bit-reversed addresses, only when the register indirect with Pre- or Post-increment Addressing modes are used (`[Wn++]`, `[++Wn]`). Furthermore, bit-reverse addresses are only generated for Word mode instructions. It will not function for all other Addressing modes or Byte mode instructions (normal addresses will be generated).

Note: A write to the MODCON register should not be followed by an instruction that performs an indirect read operation using a W register. Unexpected results may occur. Some instructions perform an implicit indirect read. These are: `POP`, `RETURN`, `RETFIE`, `RETLW` and `ULNK`.

3.4.2.1 Modulo Addressing and Bit-Reversed Addressing

Modulo addressing and bit-reversed addressing can be enabled simultaneously using the same W register, but bit-reversed addressing operation will always take precedence for data writes when enabled. As an example, the following setup conditions would assign the same W register to modulo and bit-reversed addressing:

- X modulo addressing is enabled (XMODEN = `1`)
- Bit-reverse addressing is enabled (BREN = `1`)
- W1 assigned to modulo addressing (XWM<3:0> = `0001`)
- W1 assigned to bit-reversed addressing (BWM<3:0> = `0001`)

For data reads that use W1 as the pointer, modulo address boundary checking will occur. For data writes using W1 as the destination pointer, the bit-reverse hardware will correct W1 for data re-ordering.

3.4.2.2 Data Dependencies Associated with XBREV

If bit-reversed addressing has already been enabled by setting the BREN (XBREV<15>) bit, then a write to the XBREV register should not be followed by an indirect read operation using the W register, designated as the bit reversed address pointer.

3.4.3 Bit-Reverse Modifier Value

The value loaded into the XBREV register is a constant that indirectly defines the size of the bit-reversed data buffer. The XB modifier values used with common bit-reversed buffers are summarized in Table 3-2.

Table 3-2: Bit-Reversed Address Modifier Values

Buffer Size (Words)	XB Bit-Reversed Address Modifier Value
32768	`0x4000`
16384	`0x2000`
8192	`0x1000`
4096	`0x0800`
2048	`0x0400`
1024	`0x0200`
512	`0x0100`
256	`0x0080`
128	`0x0040`
64	`0x0020`
32	`0x0010`
16	`0x0008`
8	`0x0004`
4	`0x0002`
2	`0x0001`

Note: Only the the bit-reversed modifier values shown will produce valid bit-reversed address sequences.

The bit-reverse hardware modifies the W register address by performing a ‘reverse-carry’ addition of the W contents and the XB modifier constant. A reverse-carry addition is performed by adding the bits from left-to-right instead of right-to-left. If a carry-out occurs in a bit location, the carry out bit is added to the next bit location to the right. Example 3-5 demonstrates the reverse-carry addition and subsequent W register values using `0x0008` as the XB modifier value. Note that the XB modifier is shifted one bit location to the left to generate word address values.

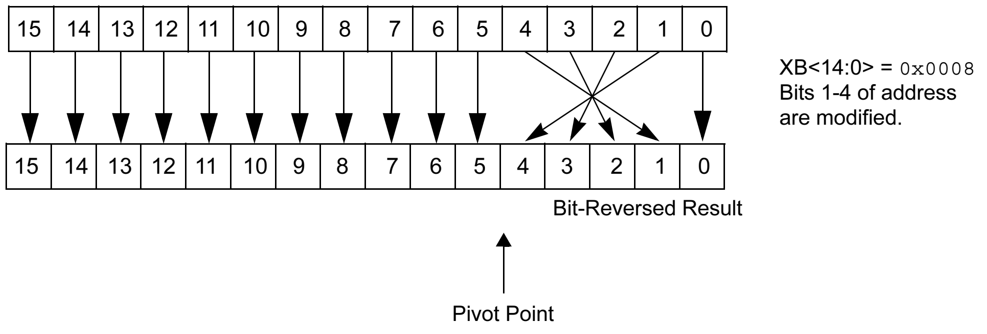
Example 3-5: XB Address Calculation

```
0000 0000 0000 0000     Wn points to word 0
             +1 0000     Wn = Wn + XB
_____________________

0000 0000 0001 0000     Wn points to word 8
             +1 0000     Wn = Wn + XB
_____________________

0000 0000 0000 1000     Wn points to word 4
             +1 0000     Wn = Wn + XB
_____________________

0000 0000 0001 1000    Wn points to word 12
             +1 0000     Wn = Wn + XB
_____________________

0000 0000 0000 0100     Wn points to word 2
             +1 0000     Wn = Wn + XB
_____________________

0000 0000 0001 0100    Wn points to word 10
```

When XB<14:0> = `0x0008`, the bit-reversed buffer size will be 16 words. Bits 1-4 of the W register will be subject to bit-reversed address correction, but bits 5-15 (outside the pivot point) will not be modified by the bit-reverse hardware. Bit 0 is not modified because the bit-reverse hardware only operates on word addresses.

The XB modifier controls the 'pivot point' for the bit-reverse address modification. Bits outside of the pivot point will not be subject to bit-reversed address corrections.

Figure 3-8: Bit-Reversed Address Modification for 16-Word Buffer



3.4.4 Bit-Reversed Addressing Code Example

The following code example reads a series of 16 data words and writes the data to a new location in bit-reversed order. W0 is the read address pointer and W1 is the write address pointer subject to bit-reverse modification.

```
; Set XB for 16-word buffer, enable bit reverse addressing
    MOV     #0x8008,W0
    MOV     W0,XBREV
; Setup MODCON to use W1 for bit reverse addressing
    MOV     #0x01FF,W0
    MOV     W0,MODCON
; W0 points to input data buffer
    MOV     #Input_Buf,W0
; W1 points to bit reversed data
    MOV     #Bit_Rev_Buf,W1
; Re-order the data from Input_Buf into Bit_Rev_Buf
    REPEAT  #15
    MOV     [W0++],[W1++]
```

3.5 Control Register Descriptions

The following registers are used to control modulo and bit-reversed addressing:

- MODCON: Modulo Addressing Control Register
- XMODSRT: X AGU Modulo Start Address Register
- XMODEND: X AGU Modulo End Address Register
- YMODSRT: Y AGU Modulo Start Address Register
- YMODEND: Y AGU Modulo End Address Register
- XBREV: X AGU Bit-Reverse Addressing Control Register

A detailed description of each register is provided on subsequent pages.

Register 3-1: MODCON: Modulo and Bit-Reversed Addressing Control Register

Upper Byte:							
R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
XMODEN	YMODEN	—	—	BWM<3:0>			
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
YWM<3:0>				XWM<3:0>			
bit 7							bit 0

bit 15 **XMODEN:** X RAGU and X WAGU Modulus Addressing Enable bit
`1` = X AGU modulus addressing enabled
`0` = X AGU modulus addressing disabled

bit 14 **YMODEN:** Y AGU Modulus Addressing Enable bit
`1` = Y AGU modulus addressing enabled
`0` = Y AGU modulus addressing disabled

bit 13-12 **Unimplemented:** Read as ‘0’

bit 11-8 **BWM<3:0>:** X WAGU Register Select for Bit-Reversed Addressing bits
`1111` = Bit-reversed addressing disabled
`1110` = W14 selected for bit-reversed addressing
`1101` = W13 selected for bit-reversed addressing
•
•
`0000` = W0 selected for bit-reversed addressing

bit 7-4 **YWM<3:0>:** Y AGU W Register Select for Modulo Addressing bits
`1111` = Modulo addressing disabled
`1010` = W10 selected for modulo addressing
`1011` = W11 selected for modulo addressing

Note: All other settings of the YWM<3:0> control bits are reserved and should not be used.

bit 3-0 **XWM<3:0>:** X RAGU and X WAGU W Register Select for Modulo Addressing bits
`1111` = Modulo addressing disabled
`1110` = W14 selected for modulo addressing
•
•
`0000` = W0 selected for modulo addressing

Note: A write to the MODCON register should not be followed by an instruction that performs an indirect read operation using a W register. Unexpected results may occur. Some instructions perform an implicit indirect read. These are: `POP`, `RETURN`, `RETFIE`, `RETLW` and `ULNK`.

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

3

Data Memory

Register 3-2: XMODSRT: X AGU Modulo Addressing Start Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
XS<15:8>							
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0
XS<7:1>							0
bit 7							bit 0

bit 15-1 **XS<15:1>:** X RAGU and X WAGU Modulo Addressing Start Address bits

bit 0 **Unimplemented:** Read as ‘0’

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 3-3: XMODEND: X AGU Modulo Addressing End Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
XE<15:8>							
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1
XE<7:1>							1
bit 7							bit 0

bit 15-1 **XE<15:1>:** X RAGU and X WAGU Modulo Addressing End Address bits

bit 0 **Unimplemented:** Read as ‘1’

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 3-4: YMODSRT: Y AGU Modulo Addressing Start Register

Upper Byte:

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

Lower Byte:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0
YS<7:1>							0
bit 7							bit 0

bit 15-1	**YS<15:1>:** Y AGU Modulo Addressing Start Address bits
bit 0	**Unimplemented:** Read as ‘0’

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 3-5: YMODEND: Y AGU Modulo Addressing End Register

Upper Byte:

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

Lower Byte:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1
YE<7:1>							1
bit 7							bit 0

bit 15-1	**YE<15:1>:** Y AGU Modulo Addressing End Address bits
bit 0	**Unimplemented:** Read as ‘1’

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 3-6: XBREV: X Write AGU Bit-Reversal Addressing Control Register

Upper Byte:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BREN	XB<14:8>						
bit 15							bit 8

Lower Byte:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
XB<7:0>							
bit 7							bit 0

bit 15 **BREN:** Bit-Reversed Addressing (X AGU) Enable bit
`1` = Bit-reversed addressing enabled
`0` = Bit-reversed addressing disabled

bit 14-0 **XB<14:0>:** X AGU Bit-Reversed Modifier bits
`0x4000` = 32768 word buffer
`0x2000` = 16384 word buffer
`0x1000` = 8192 word buffer
`0x0800` = 4096 word buffer
`0x0400` = 2048 word buffer
`0x0200` = 1024 word buffer
`0x0100` = 512 word buffer
`0x0080` = 256 word buffer
`0x0040` = 128 word buffer
`0x0020` = 64 word buffer
`0x0010` = 32 word buffer
`0x0008` = 16 word buffer
`0x0004` = 8 word buffer
`0x0002` = 4 word buffer
`0x0001` = 2 word buffer

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

3.6 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. The current application notes related to the Data Memory module are:

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

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

3.7 Revision History

Revision A

This is the initial released revision of this document.

Revision B

This revision incorporates additional technical content for the dsPIC30F Data Memory module.

Revision C

This revision incorporates all known errata at the time of this document update.