AN726

PIC17CXXX to PIC18CXXX Migration

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

The specification of the PIC18CXXX Architecture was done with several goals. One of the most important of these goals was code compatibility with existing PICmicro® families. This goal eases the migration from one product family to the PIC18CXXX family.

For customers that are designing a new application that is based on an existing PICmicro device, but require added functionality (memory space, performance, peripheral features, ...), having source code compatibility is very useful (eases the development).

This application note looks at what may need to be addressed when migrating an application from a PIC17CXXX device to a PIC18CXXX device. It will not address the details of layout issues due to the different pinouts between these two families.

So looking at the issues for a code conversion, the following few points need to be inspected:

- Module Differences
- Memory Map Differences
  - Program Memory Map
  - Data Memory Map
- Instruction Execution Differences
- Architectural Modifications (such as Table Read and Table Write implementation)

Like any conversion project, the ease of the conversion is influenced by the way the initial project was implemented, such as using the register names and bit names from the data sheet (supplied in the Microchip include file). This along with other good programming techniques (symbolic code, documentation, ...) do a lot to ease the effort in a conversion project.

MODULE DIFFERENCES

First, one needs to understand what are the differences between the modules. Then the code can be evaluated to see if there are any changes required due to these differences. Some modules are functionally compatible and should only require minor changes due to the differences of the program and data memory maps of the devices. Other modules have differences due to the decision to keep module compatibility with the PICmicro Mid-Range family.

The following PIC17CXXX modules are upward compatible to the PIC18CXXX module. This means that the status and control bits are in the same registers at the same bit position. The PIC18CXXX module may have some additional control bits for the added features, but as long as the PIC17CXXX unimplemented bits were written as'0', the modules will operate in the same modes. PIC17CXXX modules that should not require source code modification to function on the PIC18CXXX family include:

- MSSP
- USART
- Hardware 8 x 8 Multiply

PIC17CXXX modules that will require some source code modification to function on the PIC18CXXX family include:

- 10-Bit A/D
- Timer0

PIC17CXXX modules that will require extensive source code modification to function on the PIC18CXXX family include:

- Timer 1
- Timer 2
- Timer 3
- Capture
- PWM
- In-Circuit Serial Programming (ICSP™)

A/D module

The PIC18CXXX 10-bit A/D module was specified to be compatible with the PIC16CXXX 10-bit A/D module. This means that there are some differences in the location of the status/control bits in the ADCON0 and ADCON1 registers. Table 1 shows which A/D control registers the bits reside in, and the comments indicate if the bit position changed or if it is in a different register.

Migration Impact

Code written for the PIC17C7XX 10-bit A/D module will require changes due to the remapping of the bit locations, as well as the differences for the program and data memory maps of the devices. The functionality of the module did not change, so the timing requirements should not need any modifications.

Note:	Please refer to the device data sheet for timing specifications to ensure applicability.

CCP Special Event Trigger

The CCP Special Event Trigger allows the compare action to start an A/D conversion. This feature is not present on the PIC17C7XX family and is an enhancement that does not affect code migration to the PIC18CXXX family.

TABLE 1: 10-BIT A/D BIT COMPATIBILITY

Bit	PIC17CXXX Register	PIC18CXX2 Register	Comments
ADON	ADCON0	ADCON0	— (1)
GO/$\overline{\text{DONE}}$	ADCON0	ADCON0	— (1)
CHS3	ADCON0	N.A.	PIC18CXX2 has up to 8 analog input channels. PIC17C7XX has up to 16 analog input channels.
CHS2	ADCON0	ADCON0	New bit position
CHS1	ADCON0	ADCON0	New bit position
CHS0	ADCON0	ADCON0	New bit position
ADCS2	N.A.	ADCON1	PIC18CXX2 has 3 new A/D Conversion Clock selections: Fosc/2, Fosc/4, and Fosc/16
ADCS1	ADCON1	ADCON0	Moved to different register
ADCS0	ADCON1	ADCON0	Moved to different register
ADFM	ADCON1	ADCON1	New bit position
PCFG3	ADCON1	ADCON1	— (1)
PCFG2	ADCON1	ADCON1	— (1)
PCFG1	ADCON1	ADCON1	— (1)
PCFG0	ADCON1	ADCON1	— (1)

Note 1: No change required

USART module

The PIC17CXXX has a USART module, while the PIC18CXXX has an Addressable USART (AUSART) module. The AUSART module is based on the PIC16CXXX family AUSART module, which has the high baud rate feature. All bits for the PIC17CXXX USART module have the same register names and the same bit position as the PIC18CXXX AUSART module. The AUSART module has two additional bits, the High Baud Rate Select (BRGH) bit and the Address Detect Enable (ADDEN) bit.

Table 2 shows the Addressable USART Register compatibility.

Migration Impact

Code written for the PIC17CXXX USART module should only require changes due to the differences for the program and data memory maps of the devices, but not due to the functionality of the module. The default state of the BRGH and ADDEN bits after a Power-on Reset allows compatibility with the PIC17CXXX USART module. Ensure that your code did not modify the state of these bits from the default state of '0'.

TABLE 2: ADDRESSABLE USART COMPATIBILITY

Bit	PIC17CXXX Register	PIC18CXX2 Register	Comments
CSRC	TXSTA	TXSTA	— (1)
TX9	TXSTA	TXSTA	— (1)
TXEN	TXSTA	TXSTA	— (1)
SYNC	TXSTA	TXSTA	— (1)
BRGH	N.A.	TXSTA	New bit
TRMT	TXSTA	TXSTA	— (1)
TX9D	TXSTA	TXSTA	— (1)
SPEN	RCSTA	RCSTA	— (1)
RX9	RCSTA	RCSTA	— (1)
SREN	RCSTA	RCSTA	— (1)
CREN	RCSTA	RCSTA	— (1)
ADDEN	N.A.	RCSTA	New bit
FERR	RCSTA	RCSTA	— (1)
OERR	RCSTA	RCSTA	— (1)
RX9D	RCSTA	RCSTA	— (1)

Note 1: No change required

Timer0 module

This module was specified to allow an operational compatibility to both the PIC16CXXX and PIC17CXXX families. Compatibility is specified by some new control bits. Table 3 shows the Timer0 Register compatibility. Figure 1 shows the PIC17CXXX Timer0 Block Diagram, while Figure 2 shows PIC18CXXX Timer0 Block Diagram when in 16-bit timer mode (T08BIT is cleared).

In the PIC17CXXX, the Timer0 module has the unique characteristic of having its own interrupt vector address. In PIC18CXXX devices, the Timer0 interrupt is included with all the other peripheral interrupts. Code conversions will need to take this into account.

Migration Impact

In the PIC18CXXX, the T08BIT selects if the Timer0 module will operate as an 8-bit timer or a 16-bit timer. To make this compatible with the PIC17CXXX implementation, the T08BIT must be cleared by software to select the 16-bit timer mode (the default state is set). When in the 16-bit timer mode, the 16-bit reads are now buffered. The TMR0H register is a buffered register that is loaded/written with an access to the TMR0L register. This allows removal of any software routines that were used to ensure a proper 16-bit read.

The PIC18CXXX PreScaler Assignment (PSA) bit selects if the prescaler is to be used. The default is prescaler not used, giving the same default prescale assignment as the PIC17CXXX Timer0. If the PSA bit is cleared, the prescaler is used. With the default state of the T0PS2:T0PS0 bits, the prescale assignment is 1:256. To assign this value to the PIC17CXXX, T0PS3 would need to be set and the T0PS2:T0PS0 bits would be don't care. For the PIC18CXXX, when the prescaler is selected all T0PS2:T0PS0 bits have meaning.

The PIC17CXXX Timer0 Interrupt vector address is no longer a dedicated location in the PIC18CXXX. The interrupt service routine is now required to test the TMR0IF bit as a potential interrupt source. Interrupt latency can be addressed by partitioning the interrupt sources between the high and low priority interrupt vector addresses. This technique is application dependent.

TABLE 3: TIMER0 REGISTER COMPATIBILITY

Bit	PIC17CXXX Register	PIC18CXX2 Register	Comments
TMR0ON	N.A.	T0CON	New bit to start Timer0 incrementing
T08BIT	N.A.	T0CON	New bit to configure timer in 16-bit mode
T0CS	T0STA	T0CON	New register
T0SE	T0STA	T0CON	New register and bit position
PSA	N.A.	T0CON	New register and bit position
T0PS3	T0STA	N.A.	— (1)
T0PS2	T0STA	T0CON	New register and bit position
T0PS1	T0STA	T0CON	New register and bit position
T0PS0	T0STA	T0CON	New register and bit position
INTEDG	T0STA	N.A.	— (1)

Note 1: This bit name is not applicable to the PIC18CXXX family.

FIGURE 1: PIC17CXXX TIMER0 MODULE BLOCK DIAGRAM

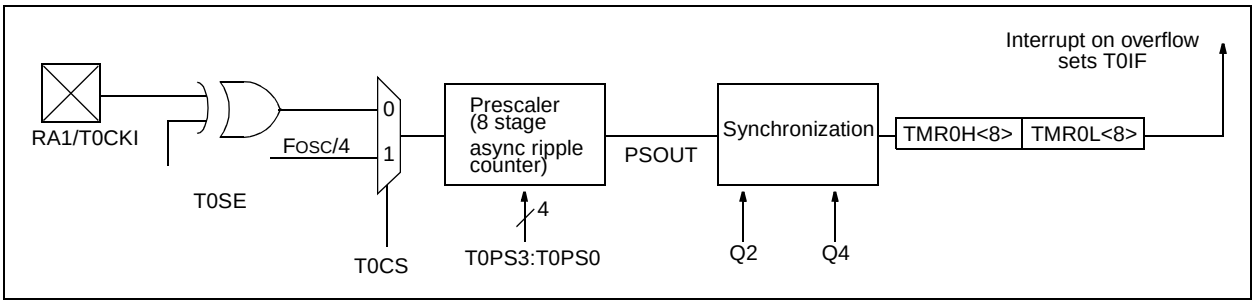
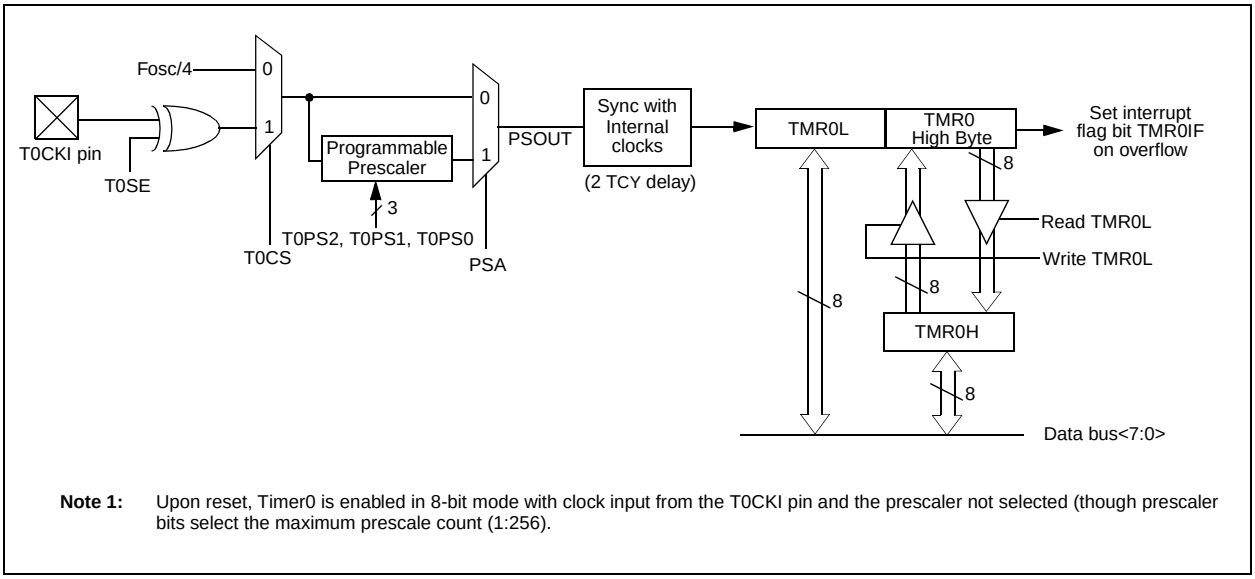


FIGURE 2: PIC18CXXX TIMER0 MODULE BLOCK DIAGRAM



Note 1: Upon reset, Timer0 is enabled in 8-bit mode with clock input from the T0CKI pin and the prescaler not selected (though prescaler bits select the maximum prescale count (1:256).

Timer1 module

The implementation of the PIC17CXXX Timer1 module is completely different from that on the PIC18CXXX. The module used on the PIC18CXXX family is the same implementation as the Timer1 module found on the PIC16CXXX with some enhancements. This module now allows a true implementation of a Real Time Clock circuit.

Figure 3 shows the Timer1 block diagrams for the PIC17CXXX. Operation in both the 8-bit and 16-bit modes are shown.

Figure 4 shows the Timer1 block diagram for the PIC18CXXX.

Migration Impact

This module requires a source code rewrite.

FIGURE 3: PIC17CXXX TIMER1 BLOCK DIAGRAMS

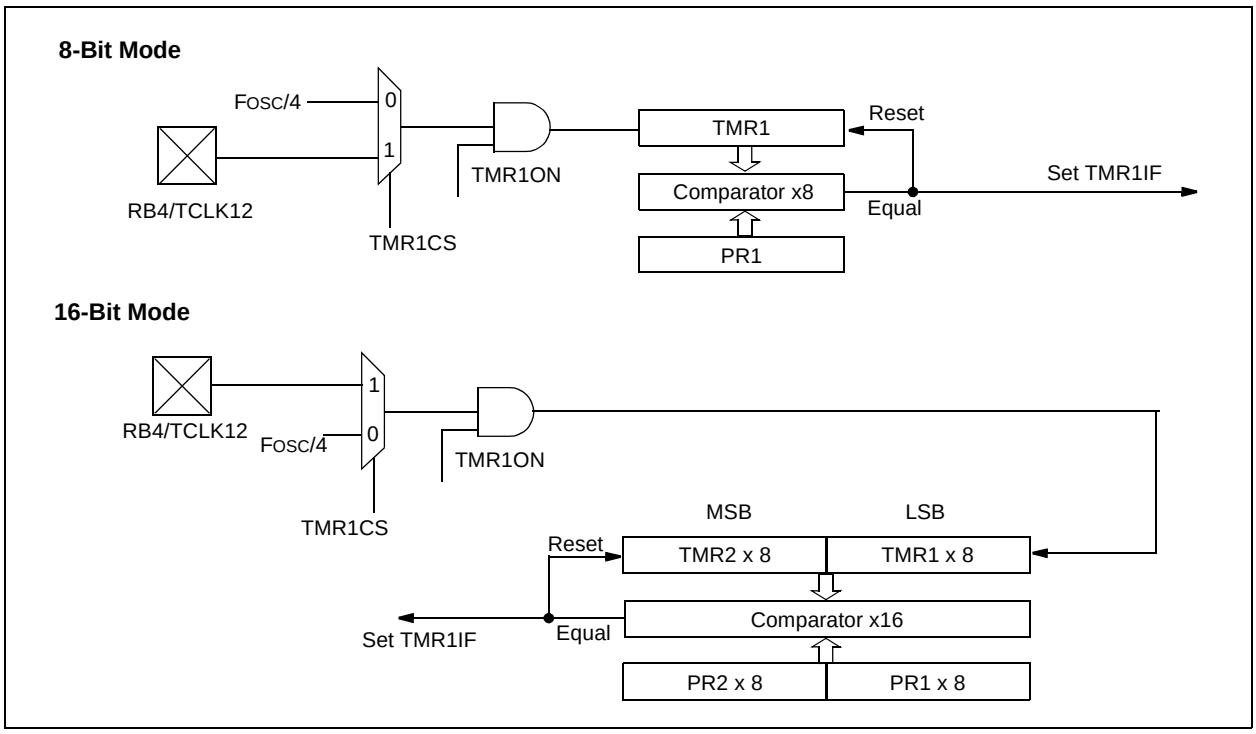
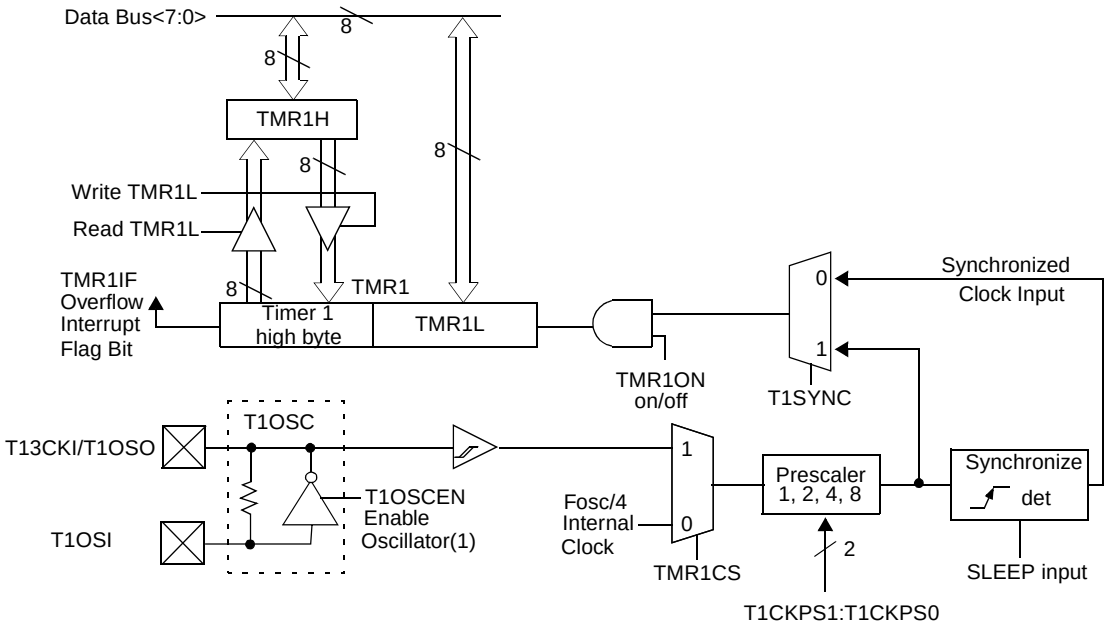


FIGURE 4: PIC18CXXX TIMER1 MODULE BLOCK DIAGRAM



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

Timer2 module

The implementation of the PIC17CXXX Timer2 module is completely different from that on the PIC18CXXX. The module used on the PIC18CXXX family is the same implementation as the Timer2 module found on the PIC16CXXX.

Figure 5 shows the Timer2 block diagrams for the PIC17CXXX. Operation in both the 8-bit and 16-bit modes are shown.

Figure 6 shows the Timer2 block diagram for the PIC18CXXX.

Migration Impact

This module requires a source code rewrite.

FIGURE 5: PIC17CXXX TIMER2 BLOCK DIAGRAMS

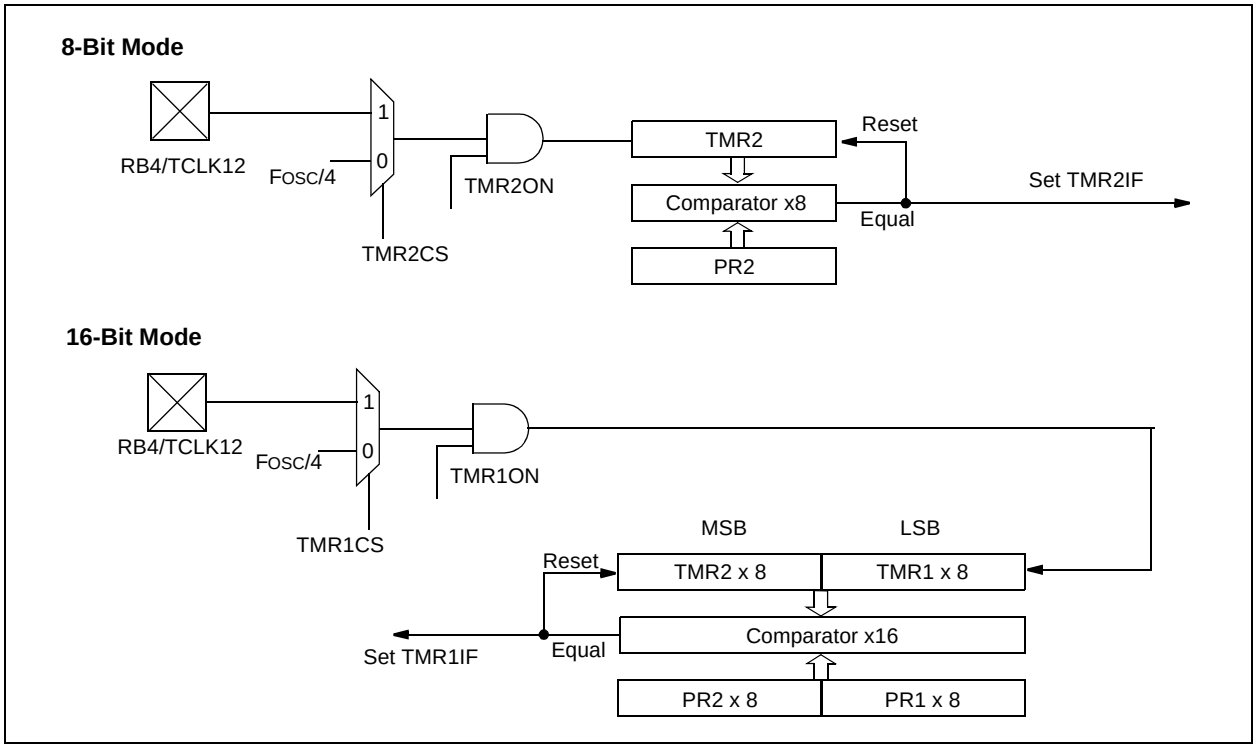
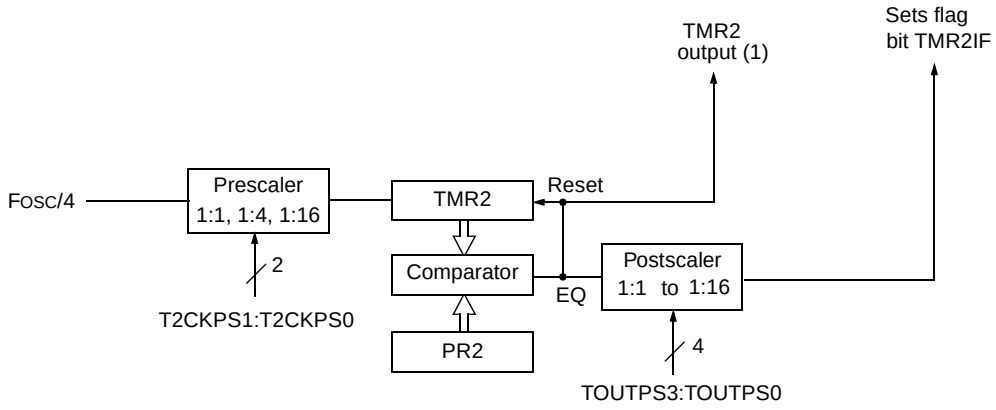


FIGURE 6: PIC18CXXX TIMER2 MODULE BLOCK DIAGRAM



Note 1: TMR2 register output can be software selected by the MSSP Module as a baud clock.

Timer3 module

The implementation of the PIC17CXXX Timer3 module is completely different from that on the PIC18CXXX. The module used on the PIC18CXXX family is the same implementation as the Timer1 module found on the PIC16CXXX, with some enhancements. This module now allows a true implementation of a Real Time Clock circuit.

Figure 7 is the block diagram of the PIC17CXXX Timer3 with three capture registers and one period register. Figure 8 is the block diagram of the PIC17CXXX Timer3 with four capture registers. As can be seen from these diagrams, the Timer3 module is tightly linked with the capture feature of the PIC17CXXX. In the PIC18CXXX, the capture feature is a software programmable mode of the CCP module.

Figure 9 is a block diagram of the PIC18CXXX Timer3 module.

Migration Impact

This module requires a source code rewrite.

FIGURE 7: PIC17CXXX TIMER3 WITH THREE CAPTURE AND ONE PERIOD REGISTER BLOCK DIAGRAM

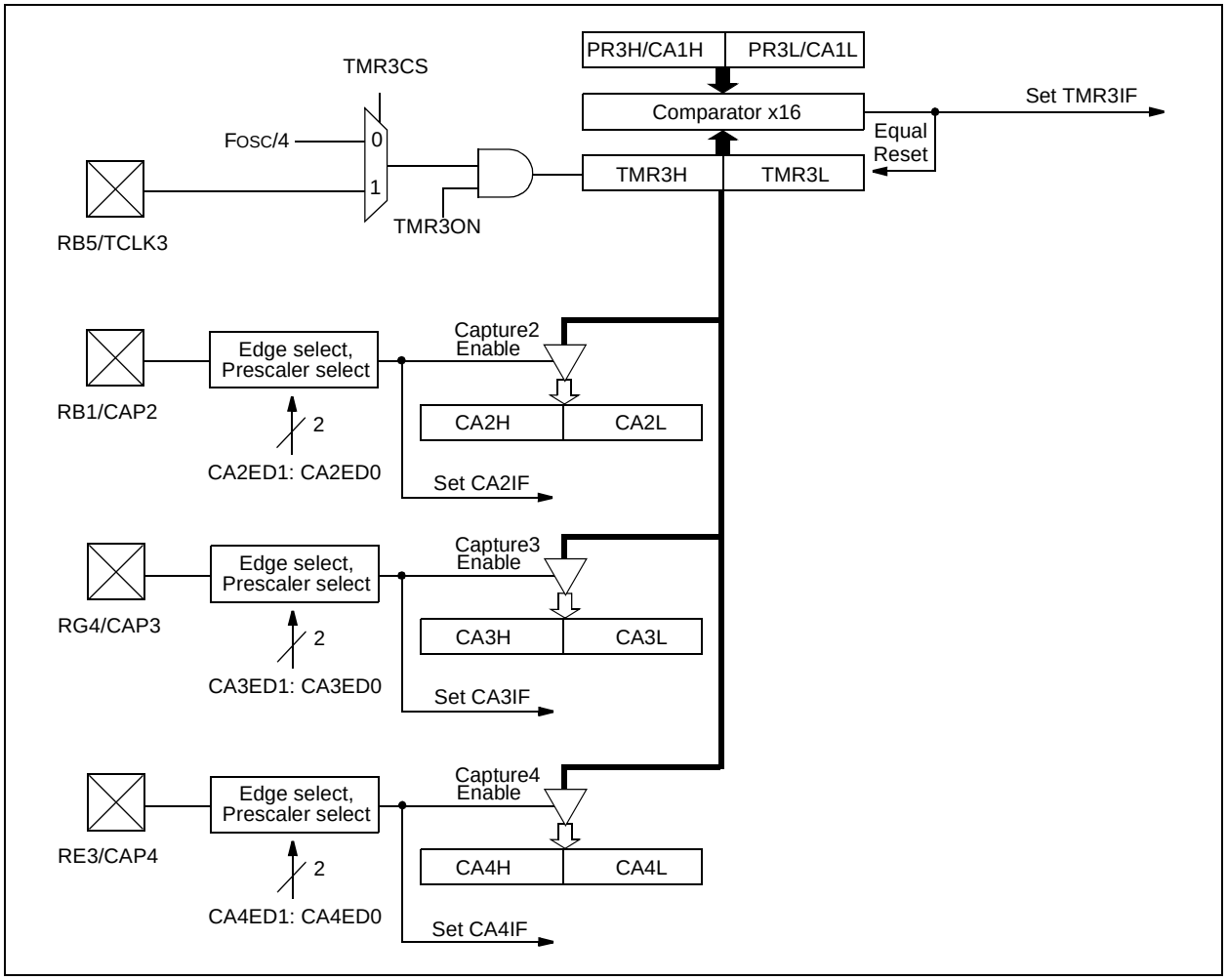


FIGURE 8: PIC17CXXX TIMER3 WITH FOUR CAPTURES BLOCK DIAGRAM

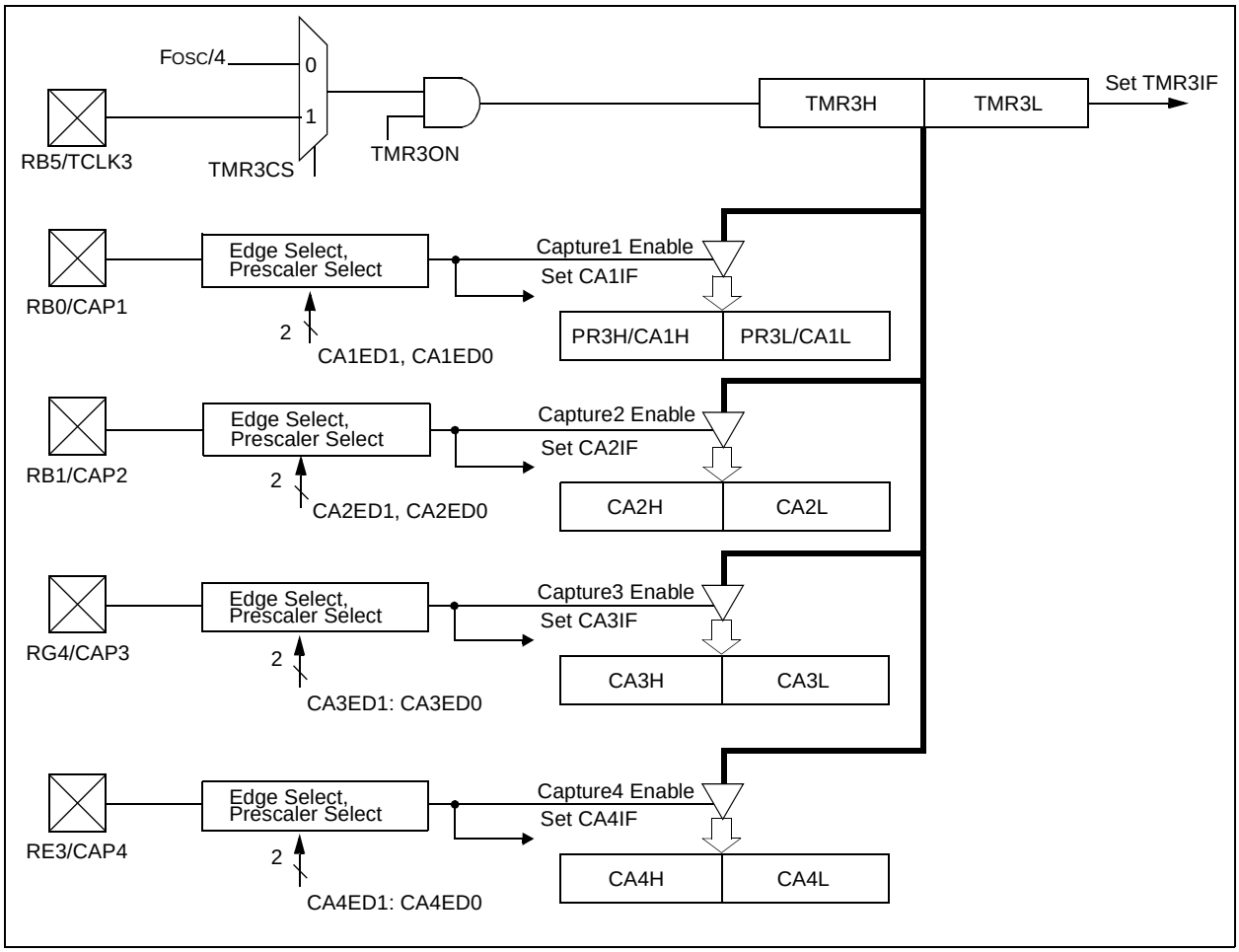
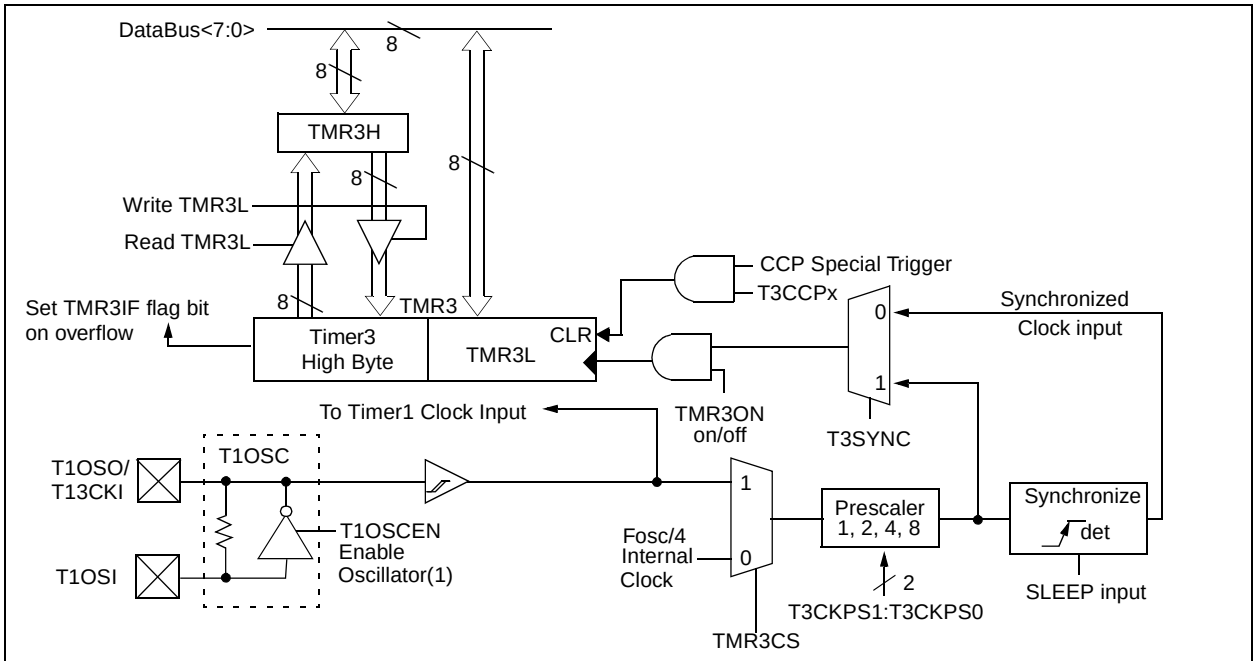


FIGURE 9: PIC18CXXX TIMER3 MODULE BLOCK DIAGRAM



Note 1: When the T1OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

Capture/Compare/PWM modules

The PIC18CXXX family uses CCP modules. This is compatible with PIC16CXXX family devices. The PIC17CXXX allows more features to be used concurrently (3 PWM outputs and 4 capture inputs), while the PIC18CXX2 devices have 2 CCP modules. Table 4 shows the timer resources that are usable for the Time Based Operation feature selected.

TABLE 4: TIMER RESOURCES FOR TIME BASED OPERATION FEATURES

Time Based Feature	PIC17CXXX	PIC18CXXX
Capture	Timer3	Timer1 or Timer3
Compare	N.A.	Timer1 or Timer3
PWM	Timer1 or Timer2	Timer2

PWM Operation

In the PIC17CXXX, the PWM time base can be set to either Timer1 or Timer2. These timers both have the capability to have their clock source derived from the external pin TCLK12. The PIC18CXXX PWM must always use Timer2 as the time base with the clock source from the internal device clock. Table 5 shows the registers used to specify the PWM duty cycle between the two families.

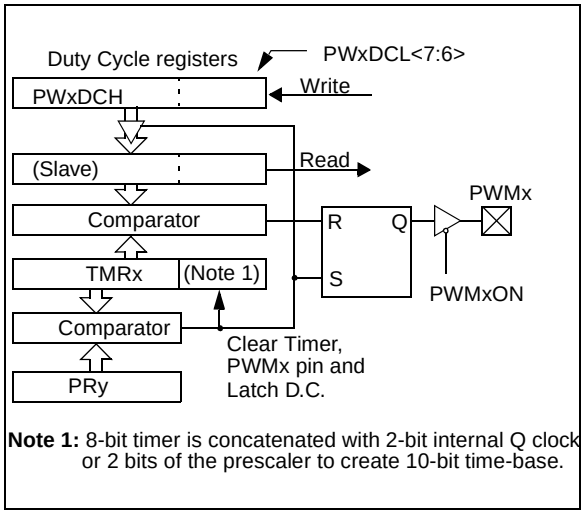
Figure 10 is a block diagram of the PIC17CXXX PWM. Timer1 or Timer2 may be used as the time base for the PWM outputs.

Figure 11 is a block diagram of the PIC18CXXX PWM. Timer2 is the time base for all PWM outputs.

TABLE 5: DUTY CYCLE REGISTERS

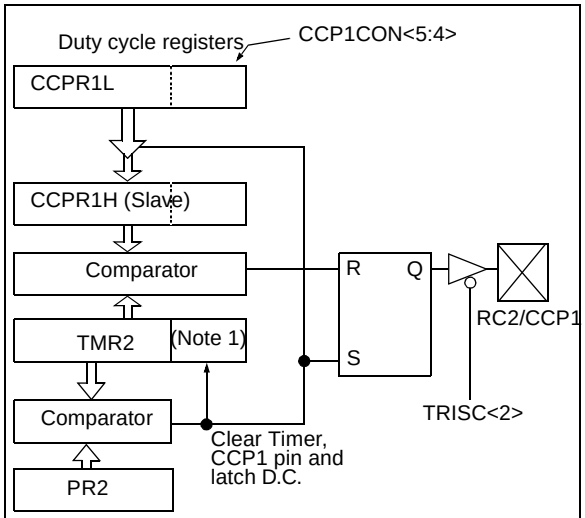
Device	PWM Duty Cycle Bits	
	DC9:DC2	DC1:DC0
PIC17CXXX	PWxDCH	PWxDCL<7:6>
PIC18CXXX	CCPRxL	CCPxCON<5:4>

FIGURE 10: PIC17CXXX PWM BLOCK DIAGRAM



Note 1: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time-base.

FIGURE 11: PIC18CXXX PWM BLOCK DIAGRAM



Note: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit timebase.

MIGRATION IMPACT

Migrating code from the PIC17CXXX family to the PIC18CXXX family will require a rewrite of the source code to function. Since the CCP module is software programmable to operate in any of the three modes, the total number of PWM outputs may not match what is provided by the PIC17CXXX.

Capture Operation

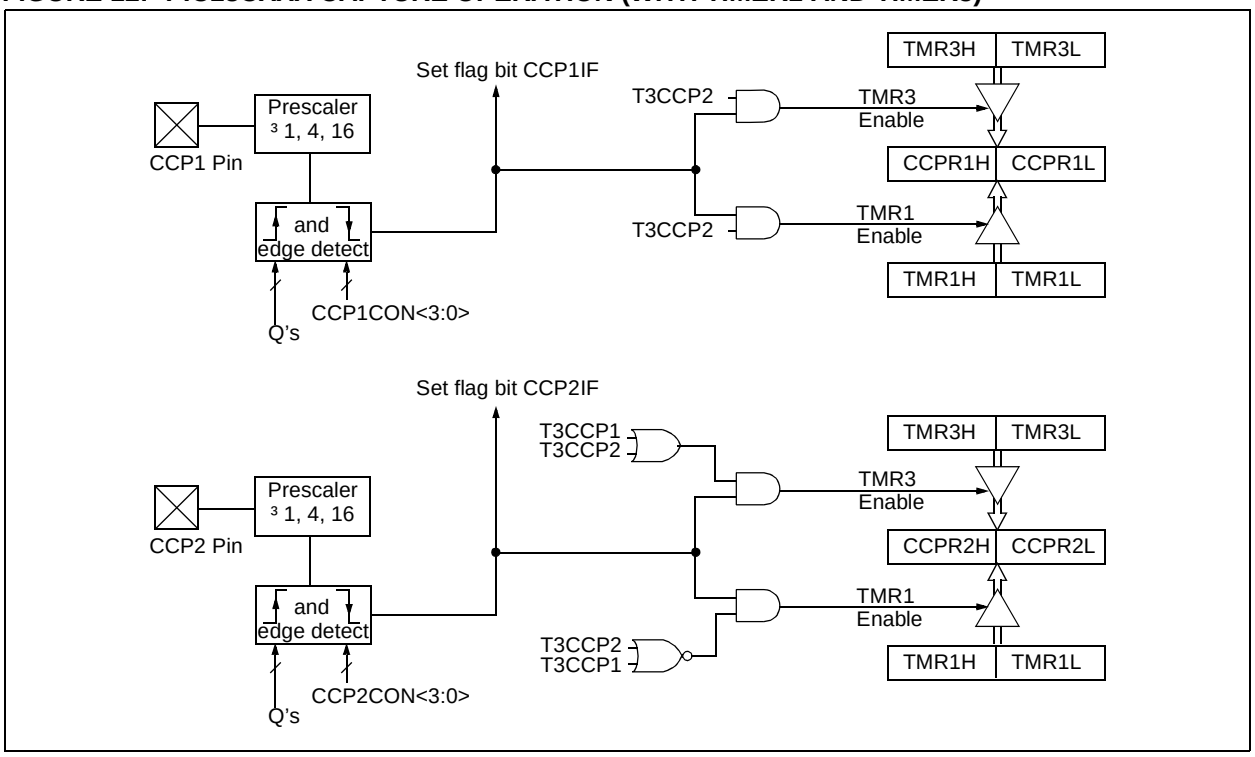
In the PIC17CXXX family, the capture feature is tightly linked with the Timer3 module. Figure 7 and Figure 8 show the capture block diagrams.

Figure 12 is the PIC18CXXX Capture Operation Block Diagram. In the PIC18CXXX, the capture feature is a software programmable mode of the CCP module.

MIGRATION IMPACT

Migrating code from the PIC17CXXX family to the PIC18CXXX family will require a rewrite of the source code to function. Since the CCP module is software programmable to operate in any of the three modes, the total number of capture inputs may not match what is provided by the PIC17CXXX.

FIGURE 12: PIC18CXXX CAPTURE OPERATION (WITH TIMER1 AND TIMER3)



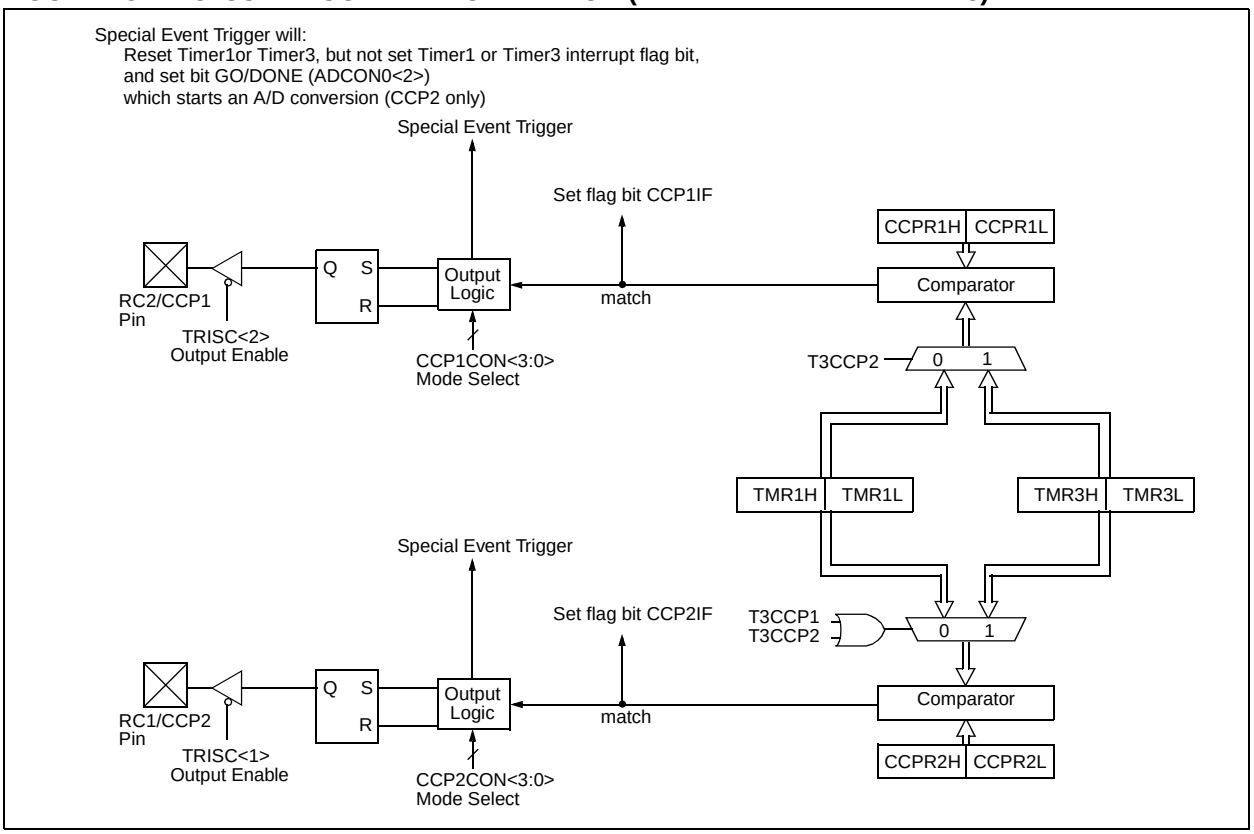
Compare Operation

The PIC17CXXX family does not support a compare operation. This is an enhancement for the PIC18CXXX family. Figure 13 shows the operation of the PIC18CXXX compare mode. Two compare values can be initialized, and they can be used to compare against either Timer1 or Timer3.

MIGRATION IMPACT

This feature of the CCP module is an enhancement to the PIC17CXXX devices.

FIGURE 13: PIC18CXXX COMPARE OPERATION (WITH TIMER1 AND TIMER3)



Master SSP Module

The PIC17CXXX MSSP module is upwardly compatible with the PIC18CXXX MSSP module. The PIC18CXXX MSSP module also includes two modes that are present in the PIC16CXXX SSP module. These are the modes:

1. I^2C slave mode, 7-bit address with start and stop bit interrupts enabled
2. I^2C slave mode, 10-bit address with start and stop bit interrupts enabled

These modes were retained for ease of code migration from PIC16CXXX devices to the PIC18CXXX family.

Migration Impact

Code written for the PIC17CXXX MSSP module should only require changes due to the differences in the program and data memory maps of the devices, but not due to the functionality of the module.

External Interrupts

For the PIC17CXXX, the INT interrupt had its own vector address. In the PIC18CXXX, it is part of the peripheral interrupts vector address. This means that the INT interrupt code will need to be moved into the general peripheral interrupt service routine (ISR), and this routine will need to add a check for the INT interrupt source.

The PIC18CXXX family has some enhancements for the external interrupts. First, there are now three external interrupt pins, as opposed to one pin in the PIC17CXXX family. Second, enhancements to the architecture of the interrupt logic allows additional capability (High/Low priority). These enhancements are discussed in the section "Architectural Enhancements" .

Migration Impact

The PIC17CXXX external interrupt requires minor modifications to be used with the PIC18CXXX devices.

PortB Interrupt-On-Change

The PORTB interrupt-on-change feature of the PIC17CXXX family has all PORTB pins with the interrupt on change feature. This feature was multiplexed with other peripheral features such as Captures, PWMs, Timer clock inputs, and SPI pins. The PORTB interrupt on change feature of the PIC18CXXX family matches that of our Mid-Range family. That is, there is only an interrupt on change on the upper four port pins of PORTB. There are no other peripheral feature multiplexed onto these pins.

Migration Impact

On the PIC18CXXX family, only RB7:RB4 have the interrupt on change feature. These pins do not have any peripheral feature multiplexed on them.

PORTB Weak Pull-up Enable

The control bit to enable the weak pull-ups on PORTB have been moved from PORTB<7> (PIC17CXXX) to INTCON2<7> (PIC18CXXX).

Migration Impact

Code changes are only required due to the differences of the data memory maps.

Hardware 8 x 8 Multiply

The operation of the 8 x 8 hardware multiply is identical between the two families.

Migration Impact

Changes may only be required due to the differences of the data memory maps.

Brown-out Reset (BOR)

The Brown-out Reset (BOR) logic has been enhanced in the PIC18CXXX family. The BOR trip point is now programmable at time of device programming. One of four trip points can be selected. The BOR trip points are shown in Table 6.

Migration Impact

Since none of these trip points are specified at the same voltage level as the trip point for the PIC17CXXX family, some modifications may need to be done with the application. These modifications may be software, hardware, or both.

TABLE 6: BOR TRIP POINT COMPARISON

Family	BOR Trip Point Option				
	4.5 V (min)	4.2 V (min)	4.0 V (typ)	2.7 V (min)	2.5 V (min)
PIC18CXXX	Yes	Yes	—	Yes	Yes
PIC17CXXX	—	—	Yes	—	—

On-Chip Oscillator Circuit

The oscillator circuit has been modified to allow new enhanced features, such as a Phase Lock Loop (PLL) option) and clock switching to the Timer1 oscillator. Clock switching allows the optimization of the applications power consumption, by only operating at high frequency (high power) when the application software requires that performance. It also allows the operation at a lower frequency (low power) when application software is not performance critical.

The oscillator options of the PIC18CXXX family allow an extended frequency range compared to the PIC17CXXX device. The oscillator mode that was selected for the PIC17CXXX device may need to be changed to operate the PIC18CXXX at the desired frequency.

Table 7 shows a comparison of the oscillator selection modes between the PIC17CXXX and the PIC18CXXX devices. There are some modes where an additional I/O pin becomes available to the device. These are the RCIO and ECIO modes.

Migration Impact

Since the oscillator circuitry is different between the two families, any external components that are required need to be re-evaluated to ensure operation in the application.

Note:	Oscillator operation should be verified to ensure that it starts and performs as expected. Adjusting the loading capacitor values and/or the oscillator mode may be required.

$\overline{\text{MCLR}}$

The $\overline{\text{MCLR}}$ operation is different between the two families. The $\overline{\text{MCLR}}$ operation of the PIC18CXXX family is identical to the Mid-Range family. Please inspect electrical specification parameter # 30 to understand the implications in your system.

Migration Impact

Ensure that the differences in the electrical specifications are met by the application circuit.

Power-On Reset (POR)

The Power-On Reset (POR) operation is different between the two families. The POR operation of the PIC18CXXX family is identical to the Mid-Range family (for the same modes).

Migration Impact

Ensure that the differences in the Power-On Reset timings are addressed by the hardware and software of the application.

In-Circuit Serial Programming (ICSP)

The ICSP operation is different between the two families. This relates to both the hardware interface as well as the software protocol and timings.

Migration Impact

The new implementation method will need to be accounted for in the design conversion.

TABLE 7: OSCILLATOR MODE SELECTION COMPARISON

Frequency Range	Oscillator Type	Oscillator Mode Selection		Comment
		PIC17CXXX	PIC18CXXX	
DC - 4 MHz	RC	RC	RC or RCIO	—
DC - 200 kHz	Crystal/Resonator	LF	LP	—
200 KHz - 2 MHz	Crystal/Resonator	LF	XT	—
2 MHz - 4 MHz	Crystal/Resonator	XT	XT	—
4 MHz - 16 MHz	Crystal/Resonator	XT	HS	—
16 MHz - 25 MHz	Crystal/Resonator	XT	HS or HS + PLL (1)	—
25 MHz - 33 MHz	Crystal/Resonator	XT	HS + PLL (2)	—
33 MHz - 40 MHz	Crystal/Resonator	N.A.	HS + PLL (3)	—
DC - 33 MHz	External Clock	EC	EC or ECIO	—
33 - 40 MHz	External Clock	N.A.	EC or ECIO	—

Note 1: The external crystal would have a frequency of 4 MHz - 6.25 MHz.

Note 2: The external crystal would have a frequency of 6.25 MHz - 8.25 MHz.

Note 3: The external crystal would have a frequency of 8.25 MHz - 10 MHz.

MEMORY MAP DIFFERENCES

The memory map affects instructions that are required for program flow and addressing program and data memory. The memory maps between the PIC17CXXX and PIC18CXXX families are similar, but still require discussion for the upward migration of application code.

These are broken down into two discussions, one for the Program Memory map and the other for the Data Memory map.

Program Memory

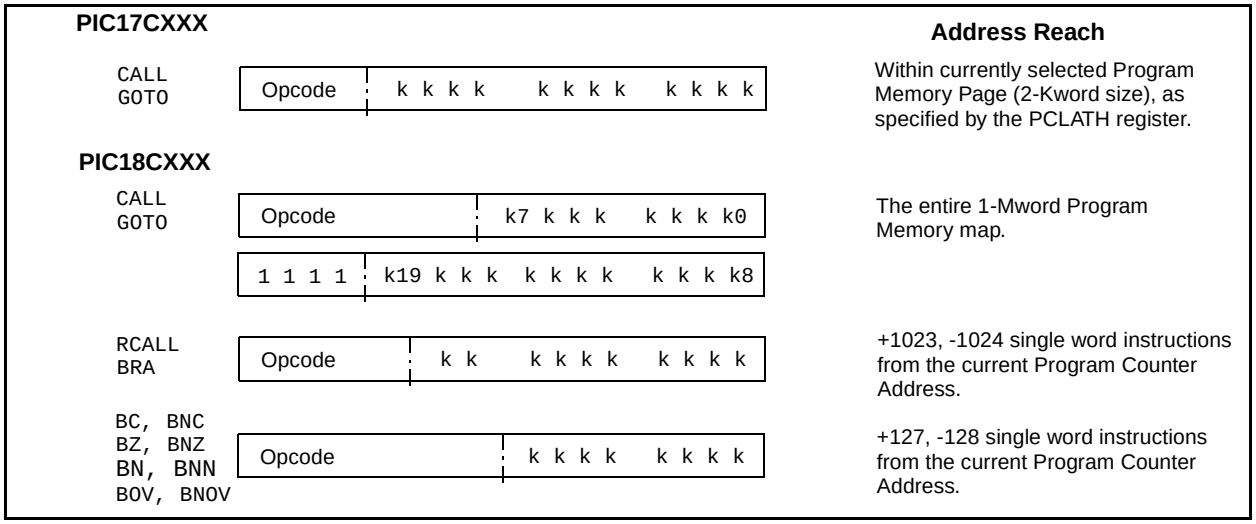
The PIC17CXXX family can address 64-Kwords of program memory (128-KBytes). This memory space is broken up into 8 program memory pages of 8-Kwords. The architecture required the modification of the PCLATH register for any CALL or GOTO instruction that has a destination in a different page than is currently selected by the PCLATH register.

The PIC18CXXX family can address 2-MBytes of program memory (1-Mword). The use of program memory pages has been eliminated. Now the CALL and GOTO instructions are 2-word instructions and can address any location in the program memory space. In some instances the destination address is close to the CALL or GOTO instruction. In these cases, optimized instructions are available; the relative call and unconditional branch instructions (called the RCALL and BRA instructions), which are one word instructions. Condition branch instructions are also available, which will branch to a new program memory location based on an offset from the current program counter value. These conditional branch instructions are useful for the generation of optimized code from a C compiler. Figure 14 shows the program flow instructions for PIC17CXXX and PIC18CXXX families.

Example 1 shows a code sequence for branching to a code segment depending on the status of the zero bit. For the PIC17CXXX family, the GOTO instruction will cause the execution to branch to the program memory page dependent on the value loaded in the PCLATCH register. In the PIC18CXXX family, the GOTO instruction is two words and can address any location in the program memory. To ensure robustness of the system, the 2nd word of a two word instruction (when executed as an instruction) is executed as a NOP. This allows the same source code to work for both families, though in the PIC18CXXX family an extra instruction cycle will be required to reach the code at the Not_Zero symbol.

Example 2 shows an alternate implementation done with the PIC18CXXX instruction set. With this instruction, the location of the software routine labeled Zero would need to be within 128 words before the BZ instruction or 127 words after the BZ instruction.

FIGURE 14: PROGRAM MEMORY FLOW INSTRUCTIONS



EXAMPLE 1: PIC17CXXX OR PIC18CXXX CODE EXAMPLE

```
          BTFSC STATUS, Z      ; Is result Zero
          GOTO  Zero           ; YES, goto the code for a result of Zero
Not_Zero  :                    ; NO, result was not Zero
```

EXAMPLE 2: ALTERNATE PIC18CXXX CODE EXAMPLE

```
          BZ    Zero           ; If result Zero, goto the code for a result of Zero
 Not_Zero :                    ; NO, result was not Zero
```

The ability to branch on the condition of a status bit value allows more efficient code to be generated. Table 8 shows how these are implemented between the PIC18CXXX family and the PIC17CXXX family. In the PIC18CXXX family, the branch is relative from the program counter location and has a reach of -128 words or +127 words. In the PIC17CXXX family two possible methods are shown. Method 1 is the positive logic method which may have been used. Method 2 (shaded) is the negative logic method which would get to the carry routine one instruction cycle quicker. Method 1 is what translates to the corresponding PIC18CXXX instruction.

Each method requires the use of a GOTO instruction. The GOTO instruction allows access to any location in the selected page of program memory (as specified by the value in the PCLATH register). The number of cycles indicates the number of cycles to get to the desired routine for the true case (as defined by the PIC18CXXX conditional branch instruction) and the cycles in parentheses () indicates the number of cycles for the false case. As can be seen by the Table 8 comparison, the number of cycles and memory requirements is better for the PIC18CXXX instructions.

Migration Impact

Minimal changes should be required for PIC17CXXX source code. Any operations on PCLATH (paging) are ignored, since the PIC18CXXX families CALL and GOTO instructions contain the entire address.

Look-up tables will require some sort of modification. Tables implemented using the RETLW instruction need to be modified due to the Program Counter now being a byte counter (see explanation in "Program Counter"). Tables implemented using the PIC17CXXX Table Reads will need to be modified to address the new implementation in the PIC18CXXX (see explanation in "Table Reads and Table Writes").

Code optimization can be achieved by removing instructions that modify the PCLATH register before the CALL and GOTO instructions. If the desired program memory location is within ± 1K instruction words, then the use of the BRA and RCALL instructions will maintain the use of one instruction word, instead of the new requirement for two words.

Additional optimization can be achieved by utilizing new instructions, such as Branch on condition instructions. These Branch on condition instructions must have the program memory address of the branch code to be within -128 to +127 instruction words from the branch instruction.

TABLE 8: BRANCH ON STATUS BIT COMPARISON

Alternate PIC18CXXX Instruction			PIC17CXXX Instruction Sequence (Note 1)								
Method		Cycles/ Words	Method 1			Cycle/ Words	Method 2			Cycle/ Words
BC	Carry	2 (1) /1		BTFSC	STATUS, C	3 (2) /2		BTFSS	STATUS, C	2 (3) /2
				GOTO	Carry			GOTO	NoCarry	
			NoCarry	:			Carry	:		
BNC	NoCarry	2 (1) /1		BTFSS	STATUS, C	3 (2) /2		BTFSC	STATUS, C	2 (3) /2
				GOTO	NoCarry			GOTO	Carry	
			Carry	:			NoCarry	:		
BN	Neg	2 (1) /1		BTFSC	STATUS, N	3 (2) /2		BTFSS	STATUS, N	2 (3) /2
				GOTO	Neg			GOTO	NotNeg	
			NotNeg	:			Neg	:		
BNN	NotNeg	2 (1) /1		BTFSS	STATUS, N	3 (2) /2		BTFSC	STATUS, N	2 (3) /2
				GOTO	NotNeg			GOTO	Neg	
			Neg	:			NotNeg	:		
BOV	Ovflw	2 (1) /1		BTFSC	STATUS, OV	3 (2) /2		BTFSS	STATUS, OV	2 (3) /2
				GOTO	Ovflw			GOTO	NoOvflw	
			NoOvflw	:			Ovflw	:		
BNOV	NoOvflw	2 (1) /1		BTFSS	STATUS, OV	3 (2) /2		BTFSC	STATUS, OV	2 (3) /2
				GOTO	NoOvflw			GOTO	Ovflw	
			Ovflw	:			NoOvflw	:		
BZ	Zero	2 (1) /1		BTFSC	STATUS, Z	3 (2) /2		BTFSS	STATUS, Z	2 (3) /2
				GOTO	Zero			GOTO	NotZero	
			NotZero	:			Zero	:		
BNZ	NotZero	2 (1) /1		BTFSS	STATUS, Z	3 (2) /2		BTFSC	STATUS, Z	2 (3) /2
				GOTO	NotZero			GOTO	Zero	
			Zero	:			NotZero	:		

Note 1: This method may also be used by the PIC18CXXX family. This is source code compatible, but the GOTO instruction is now a two word instruction. When the second word of the GOTO instruction is executed as if it was a single word instruction (when the skip occurs), the second word is executed as a no operation (NOP instruction).

Data Memory

The Data Memory Map of the PIC17CXXX devices is shown in Figure 15 with the PIC18CXXX data memory map shown in Figure 16. In both architectures, the bank size is 256 bytes. Software code migration does not require the low nibble of the Bank Select Register (BSR) to be modified. The PIC17CXXX devices also bank the Special Function Registers (SFRs). This is not required with the PIC18CXXX architecture. All instructions which are used to modify the BSR<7:4> bit may be removed from the user code.

Figure 17 shows the mapping of data memory from a PIC17CXXX device to a PIC18CXXX device. The mapping translates without effort given that the GPR RAM addresses were specified with the full address, and not the relative address within the selected bank. With a full 10-bit address, the assembler will map the addresses correctly. The SFR, though not at the same addresses will be properly mapped to the correct location in bank 15 due to the supplied header file. No software coding modifications are required to address the SFR registers, since the SFR registers are in the Access bank, and can be addressed regardless of the selected bank (value of the BSR register).

The low nibble of the BSR (BSR<3:0>) specifies the RAM bank to access. This is the same for both the PIC17CXXX and PIC18CXXX. Any operations on the high nibble of the BSR (BSR<7:4>) can be ignored by the PIC18CXXX, since these bits are not implemented.

In the PIC17CXXX, the GPRs in the memory range 1Ah to 1Fh are in shared RAM. When mapped to the PIC18CXXX, these addresses are in the access bank and therefore are also shared RAM.

FIGURE 15: PIC17CXXX DATA MEMORY MAP

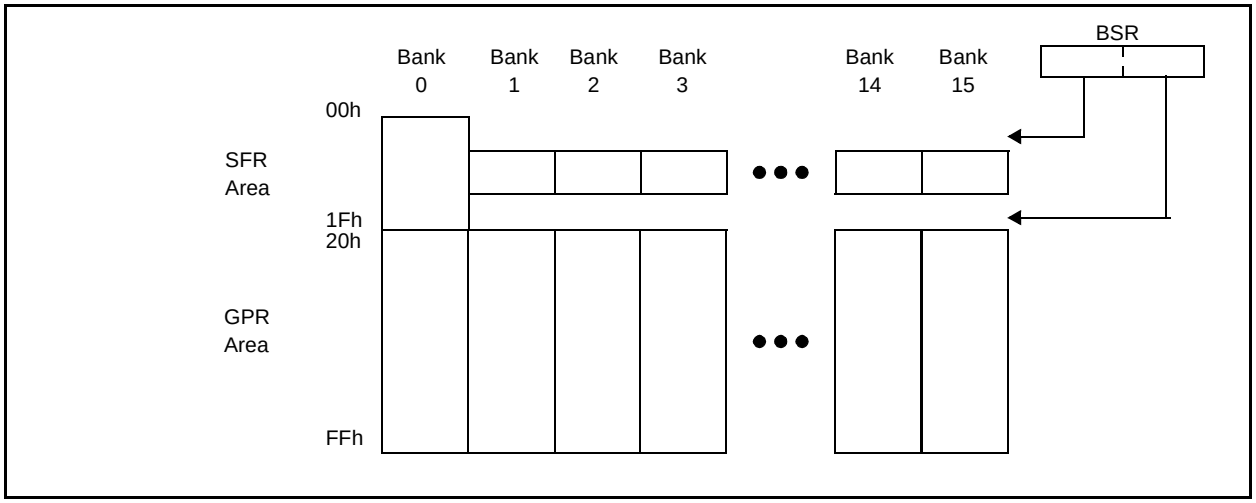


FIGURE 16: PIC18CXXX DATA MEMORY MAP

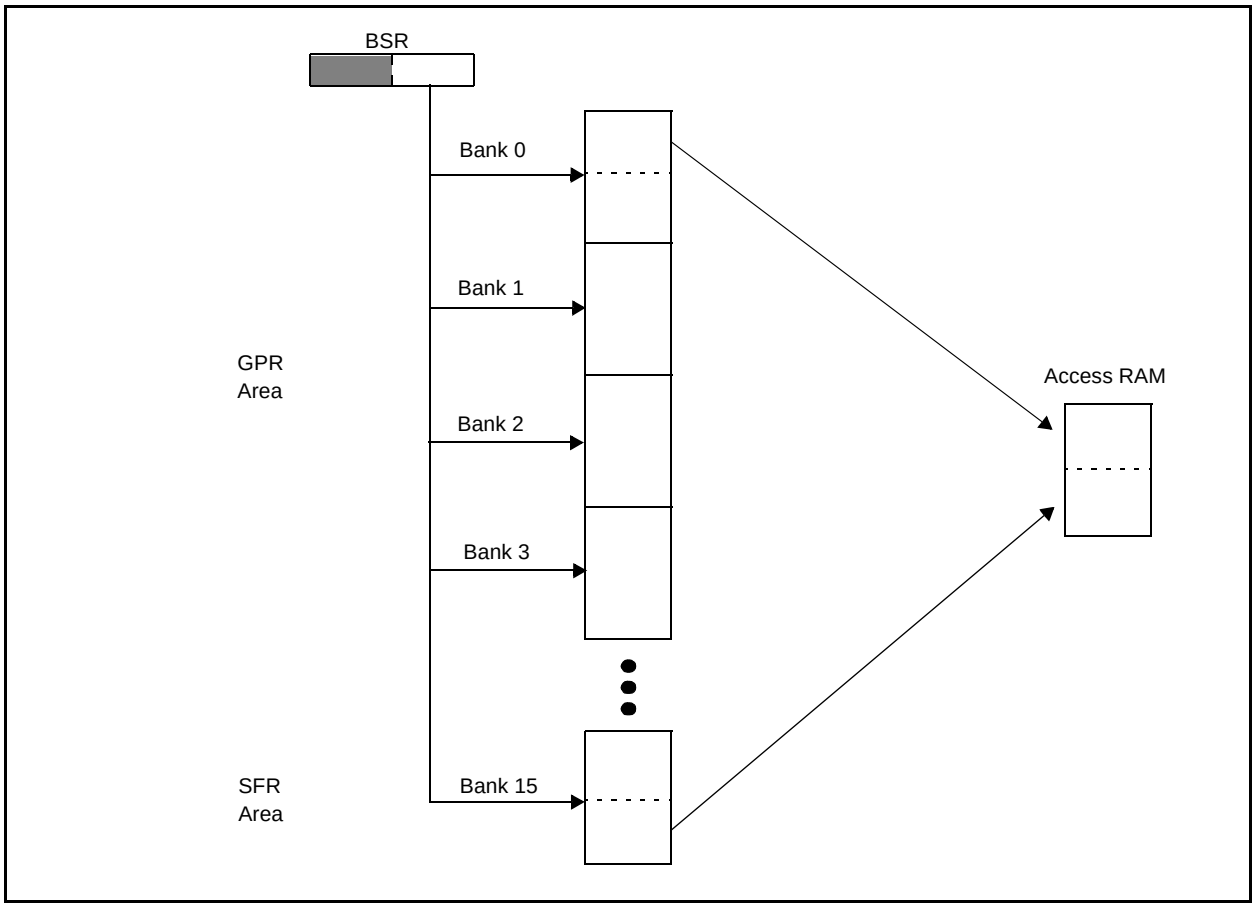
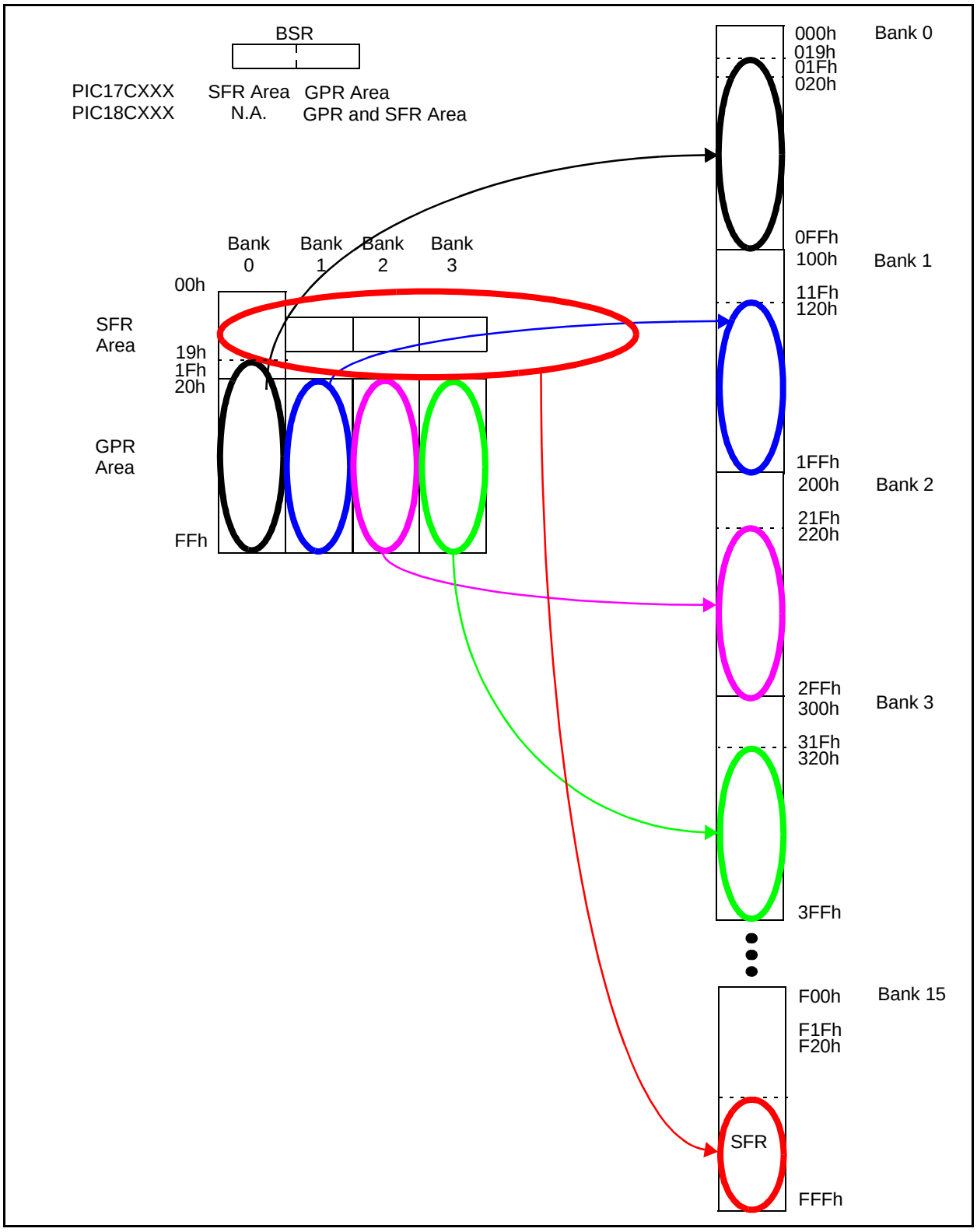


FIGURE 17: MAPPING OF DATA MEMORY FROM PIC17CXXX TO PIC18CXXX



There are occasions where the application software requires moving data from one register to another. This transfer may be a single byte or a block of data.

For the PIC17CXXX family there are instructions that move the contents from the Peripheral (P) area (first 32 locations in data memory) to the File (F) area (anywhere in the specified 256 locations), or from the File area to the Peripheral area. The actual RAM address is specified by the Bank Select Register (BSR) value, since both the SFR registers and GPR registers are banked.

With the PIC18CXXX, the instruction moves the contents from one register to another anywhere in the 4 KByte data memory space without any banking requirements.

Example 1 shows the sequence of instructions to move a value from one RAM location to another in the PIC17CXXX family. Example 2 shows the instruction to move a value from one RAM location to another in the PIC18CXXX family.

Migration Impact

If the PIC17CXXX source code uses register definitions that specifies full 12-bit addresses for ALL register file locations, then no work is required for the remapping of the data memory. The BSR low nibble (BSR<3:0>) will be updated in the same fashion. Optimization can be done by removing any instructions that are used to modify the high nibble of the BSR register (BSR<7:4>), since these bits are not implemented on the PIC18CXXX. There is one instruction that is only used to do this modification. This is the MOVLR instruction.

EXAMPLE 1: PIC17CXXX MEMORY-TO-MEMORY MOVES

Case 1: Single Byte Transfer

```
            banksel     MYREG1          ; Switch to the bank for MYREG1
                                        ;   (may not be required)
            MOVFP MYREG1, WREG          ; Move contents of MYREG1 to the
                                        ; WREG register
            banksel     MYREG2          ; Switch to the bank for MYREG2
                                        ;   (may not be required)
            MOVPF WREG, MYREG2          ; Move contents of the WREG
                                        ; register to MYREG2
```

Case 2: Block Transfer

```
            MOVLW BYTE_CNT              ; Load the Byte Count value
            MOVWF CNTR                  ; into register CNTR (same bank as MYREG1)
LP1         banksel     MYREG1          ; Switch to the bank for MYREG1
                                        ;   (may not be required)
            MOVFP MYREG1, WREG          ; Move contents of MYREG1 to the
                                        ;   WREG register
            banksel     MYREG2          ; Switch to the bank for MYREG2
                                        ;   (may not be required)
            MOVPF WREG, MYREG2          ; Move contents of the WREG
                                        ;   register to MYREG2
            DECFSZ  CNTR                ; All bytes moved?
            BRA    LP1                  ; NO, move next byte
Continue    :                           ; YES, Continue
```

EXAMPLE 2: PIC18CXXX MEMORY-TO-MEMORY MOVES

Case 1: Single Byte Transfer

```
            MOVFF MYREG1, MYREG2        ; Move contents of MYREG1 to MYREG2
```

Case 2: Block Transfer

```
            MOVLW BYTE_CNT              ; Load the Byte Count value
            MOVWF CNTR                  ; into register CNTR
LP1         MOVFF POSTINC0, POSTINC1    ; Move contents of MYREG1 to MYREG2
            DECFSZ  CNTR                ; All bytes moved?
            BRA    LP1                  ; NO, move next byte
Continue    :                           ; YES, Continue
```

INSTRUCTION SET

With the merging of the PIC16CXXX and PIC17CXXX instruction sets to create the PIC18CXXX instruction set and the enhancements to the architecture, some instructions had to be modified. Table 9 shows the PIC17CXXX instructions that have been modified. Some instructions operate on a new status bit which indicates if the resultant value is negative (N). Five instructions now affect the status of the zero (Z) bit. These instructions are:

- CLRF
- RRCF
- RRNCF
- RLCF
- RLNCF

Three instructions have changed the mnemonics, but the arguments do not need to be modified. For these three instructions a simple search and replace can be used. These instructions are:

- MOVPF
- MOVFP
- NEGW

Table 9 shows what these instructions should be replaced with.

The method of operation for four instructions (Table Reads and Table Writes) has changed. The application code surrounding the operation of this feature needs to be revisited and the code modified accordingly. These instructions are:

- TABLRD
- TLRD
- TABLWT
- TLWT

Lastly, two instructions have been removed. These instructions are:

- MOVLR
- LCALL

The MOVLR instruction is no longer required since there are no separate banks for the Special Function Registers. The LCALL instruction is changed to the PIC18CXXX CALL instruction, since it can access any location in the program memory map. The application code that preconditioned the PCLATH register can be removed, since it is no longer needed for calling the desired routine.

The PIC17CXXX family only has five instructions where the operation on the status bits changed. These are the clear file and rotate instructions (CLRF, RLCF, RLNCF, RRCF, and RRNCF).

Table 9 shows the PIC17CXXX instructions that are different in the PIC18CXXX architecture. These differences may be related to the status bits that are affected. An instruction is now handled by a more generic instruction, or the operation of the instruction has been modified to better fit with the new architecture. Rows that are shaded are new instructions to the PIC18CXXX architecture that are replacing PIC17CXXX instructions. These are shown to indicate the status bits affected.

Migration Impact

Ensure that the instructions that affect the status bits differently do not cause algorithm issues and that other instructions are appropriately converted and implemented.

TABLE 9: INSTRUCTION SET COMPARISON

Instruction	Status Bits Affected		Comment
	PIC17CXXX	PIC18CXXX	
ADDLW k	C,DC,OV, Z	C,DC,OV, Z, **N**	—
ADDWF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
ADDWFC f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
ANDLW k	Z	Z, **N**	—
ANDWF f, d	Z	Z, **N**	—
CLRF f, s	none	**Z**	Instruction now affects Zero (Z) bit
DECF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
INCF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
IORLW k	Z	Z, **N**	—
IORWF f, d	Z	Z, **N**	—
LCALL k	none	N.A.	Use CALL n, s instruction
MOVFP f, p	none	N.A.	Use MOVFF fs, fd instruction
MOVFF fs, fd	N.A.	none	Replaced MOVFP and MOVPF instructions
MOVLR k	none	N.A.	Not required for PIC18CXXX devices
MOVPF p, f	Z	N.A.	Use MOVFF fs, fd instruction
NEGW f, s	C,DC,OV, Z	N.A.	Use NEGF f instruction
NEGF f	N.A.	C,DC,OV, Z, **N**	Replaced NEGW f, s instruction
RLCF f, d	C	C, **Z, N**	Instruction now affects Zero (Z) bit
RLNCF f, d	none	**Z, N**	Instruction now affects Zero (Z) bit
RRCF f, d	C	C, **Z, N**	Instruction now affects Zero (Z) bit
RRNCF f, d	none	**Z, N**	Instruction now affects Zero (Z) bit
SUBLW k	C,DC,OV, Z	C,DC,OV, Z, **N**	—
SUBWF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
SUBWFB f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
TABLRD t, i, f	none	N.A.	Use TBLRD instructions
TABLWT t, i, f	none	N.A.	Use TBLWT instructions
TLRD t, f	none	N.A.	Use TBLRD instructions
TLWT t, f	none	N.A.	Use TBLWT instructions
TBLRD* TBLRD*+ TBLRD*- TBLRD+*	N.A.	none	Replaced TABLRD and TLRD instructions
TBLWT* TBLWT*+ TBLWT*- TBLWT+*	N.A.	none	Replaced TABLWT and TLWT instructions
XORLW k	Z	Z, **N**	—
XORWF f, d	Z	Z, **N**	—

Note 1: The N bit is new for the PIC18CXXX family of devices.

ARCHITECTURAL ENHANCEMENTS

Some of the architectural enhancements that are implemented in the PIC18CXXX family include:

- Program Counter
- Table Read / Table Write
- Interrupts
- Stack
- Indirect Addressing

Program Counter

The program counter of the PIC18CXXX Architecture works on a byte address, as opposed to a word address for the PIC17CXXX family. This means that the addresses of routines will be different. When using symbolic coding, the assembler will take care of generating the correct address, but any routine that directly modifies the program counter needs to take this difference into account. One of the most common code functions where this occurs is in table lookup routines that use the RETLW instruction.

Example 1 shows a typical table look-up for the PIC17CXXX family. Example 2 shows the table look-up for the PIC18CXXX Architecture. Since the Offset needs to be multiplied by two to get the byte address, the reach (size) of the look-up table is now half. Access to the PCLATU and PCLATH registers or the ability to do Table Reads allows larger tables to be stored in memory.

Occasionally the use of the '$' symbol is used in the source code to indicate the program address of the current instruction. The '$' syntax still operates as before, but since the program counter now specifies byte addresses any offset to the '$' parameter need to be doubled. Example 3 shows these modifications.

Migration Impact

Any modification of the PCL register will require the source code to be inspected to ensure that the desired address will be accessed. This is due to the Program Counter being a byte count into program memory and not the program memory word count. This is commonly found in simple Table Lookup routine. Remember that reading PCL updates the contents of PCLATH and PCLATU (from PCH and PCU), and writing to PCL loads PCH and PCU with the contents of PCLATH and PCLATU.

EXAMPLE 1: PIC17CXXX TABLE LOOK-UP USING THE RETLW INSTRUCTIONS

```
            MOVFP   Offset, WREG      ; Load WREG with offset to Table
            CALL    Table_LU          ; Call the lookup table
            :
            :

Table_LU    ADDWF   PCL               ; Add Offset to PCL
            RETLW   'A'               ; Returns value in WREG
            RETLW   'B'               ; Returns value in WREG
            :                         ;
```

EXAMPLE 2: PIC18CXXX TABLE LOOK-UP USING THE RETLW INSTRUCTIONS

```
            MOVFF   Offset, WREG      ;
            CALL    Table_LU          ; Call the lookup table
            :
            :

Table_LU    BCF     Offset, 7         ; Clear MSb, for rotate to LSb
            RLNCF   Offset, PCL       ; Offset * 2 added to PCL
            RETLW   'A'               ; Returns value in WREG
            RETLW   'B'               ; Returns value in WREG
            :                         ;
```

EXAMPLE 3: USE OF THE '$' PARAMETER

```
           GOTO    $ - 6             ; Replaces  GOTO   $ - 3
           GOTO    $ - 0x2E          ; Replaces  GOTO   $ + 0x17
```

Table Reads and Table Writes

Table Read and Table Write operations have been changed. In the PIC17CXXX architecture, the table pointer register points to the program memory word address. In the PIC18CXXX architecture the table pointer register points to the program memory byte address. This means that the PIC18CXXX is now only operating with 8-bits of data. This allows there to be only one instruction for a Table Read and one instruction for a Table Write. The PIC17CXXX architecture requires two instructions for each, due to operating with 16-bits of data.

Example 4 shows the PIC17CXXX code segment for reading a fixed number of words (WORD_COUNT) into sequential RAM locations using indirect addressing. Example 5 shows the PIC18CXXX code segment for reading a fixed number of bytes (BYTE_COUNT) into sequential RAM locations using indirect addressing.

Migration Impact

The code sections where Table reads and Table writes were implemented would need to be rewritten to address the differences in the implementations.

EXAMPLE 4: PIC17CXXX TABLE READ

```
          MOVLW     WORD_COUNT          ; Load the Word Count value
          MOVWF     CNTR                ;   into CNTR
;
          MOVLW     HIGH(TBL_ADDR)      ; Load the Table Address
          MOVWF     TBLPTRH             ;
          MOVLW     LOW(TBL_ADDR)       ;
          MOVWF     TBLPTRL             ;
          TABLRD    0, 1, DUMMY         ; Dummy read,
                                        ;   Updates TABLATH
                                        ;   Increments TBLPTR
LOOP1     TLRD      1, INDF0            ; Read HI byte in TABLATH
          TABLRD    0, 1, INDF0         ; Read LO byte in TABLATL,
                                        ;   update TABLATH:TABLATL,
                                        ;   and increment TBLPTR
          DECFSZ    CNTR                ; Read Word Count locations
          GOTO      LOOP1               ; Read next word
```

EXAMPLE 5: PIC18CXXX TABLE READ

```
          MOVLW     BYTE_COUNT          ; Load the Byte Count value
          MOVWF     CNTR                ;   into CNTR
;
;;        MOVLW     UPPER(TBL_ADDR)     ; Load the Table Address
;;        MOVWF     TBLPTRU             ;  (on POR TBLPTRU = 0, so
;;                                      ;  loading TBLPTRU is not
;;                                      ;  required for conversions)
          MOVLW     HIGH(TBL_ADDR)      ; Load the Table Address
          MOVWF     TBLPTRH             ;
          MOVLW     LOW(TBL_ADDR)       ;
          MOVWF     TBLPTRL             ;
LOOP1     TBLRD*+                       ; Read value into TABLAT,
                                        ;   Increment TBLPTR
          MOVFF     TABLAT, POSTINC0    ; Copy byte to RAM @ FSR0
                                        ;   Increment FSR0
          DECFSZ    CNTR                ; Read Byte Count locations
          GOTO      LOOP1               ; Read next Byte
```

Interrupts

The interrupt structure of the two families is significantly different. Figure 18 shows a simplified block diagram for the interrupt structures of the two families.

In the PIC17CXXX family, there are four interrupt vector addresses with a priority that is fixed by hardware. In the PIC18CXXX family, there are two interrupt vector addresses. One vector address for High Priority interrupts and one vector address for Low Priority interrupts. The priority of the peripheral interrupt is software programmable.

Table 10 compares the interrupt vector addresses for both the PIC17CXXX and PIC18CXXX families.

Migration Impact

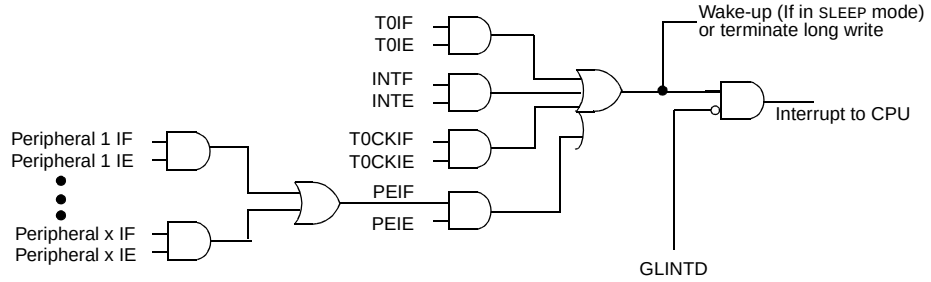
The code section for interrupt handling would need to be rewritten to address the differences in the implementations. If the PIC17CXXX separate interrupts are desired for a reduction of the interrupt latency, the PIC18CXXX HighPriority/Low Priority vectors may be able to address this.

TABLE 10: INTERRUPT VECTOR ADDRESSES

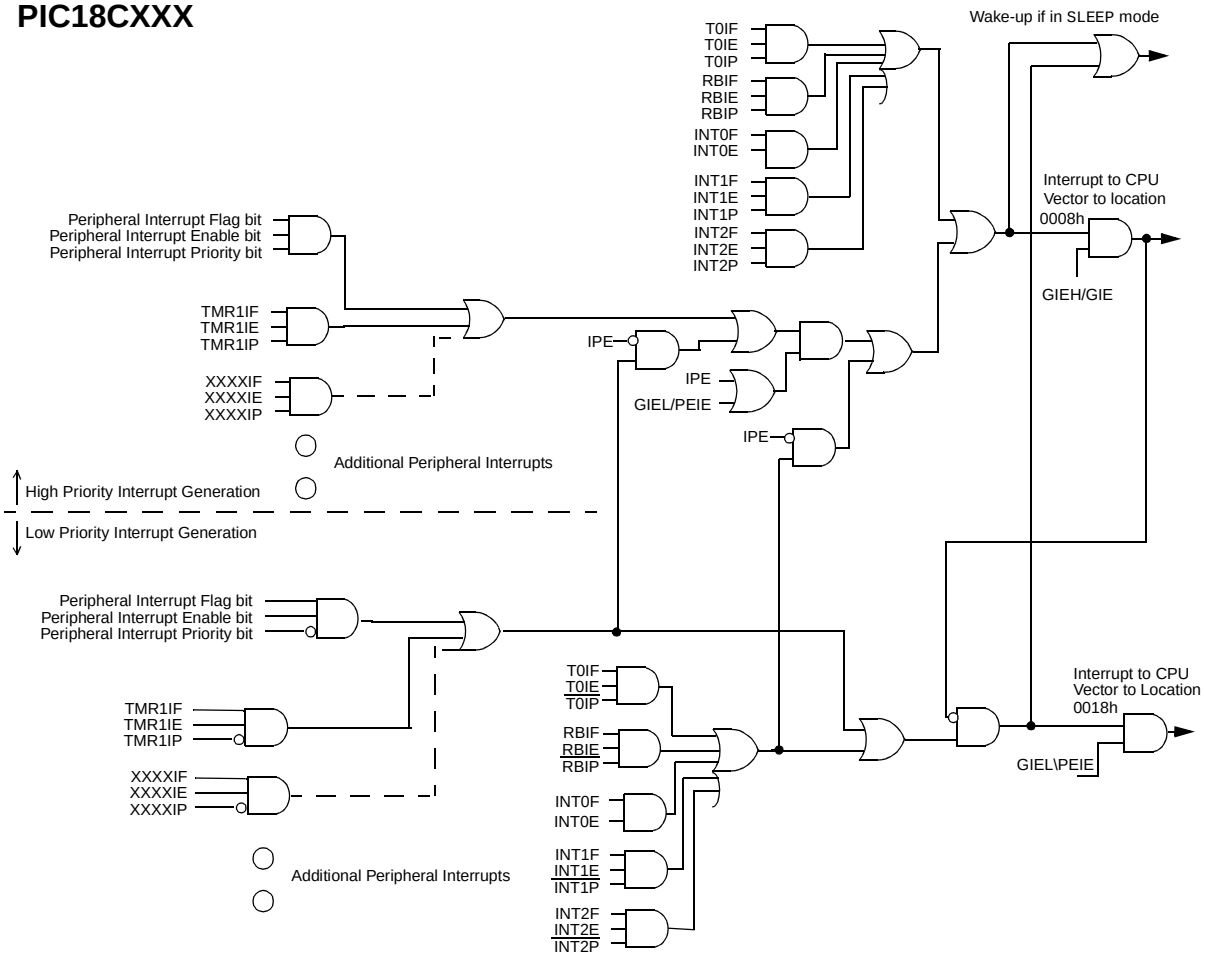
Location	PIC17CXXX Address		PIC18CXXX Address		Comment
	Word	Byte	Word	Byte	
Reset Vector Address	0000h	N.A.	0000h	0000h	—
INT pin Interrupt Vector Address	0004h	N.A.	—	—	PIC17CXXX must move this code to either the high or low priority interrupt vector address
High Priority Interrupt Vector Address	—	—	0004h	0008h	—
Low Priority Interrupt Vector Address	—	—	000Ch	0018h	—
Timer0 Interrupt Vector Address	0010h	N.A.	—	—	PIC17CXXX must move this code to either the high or low priority interrupt vector address
T0CKI pin Interrupt Vector Address	0018h	N.A.	—	—	PIC17CXXX must move this code to either the high or low priority interrupt vector address
Peripheral Interrupt Vector Address	0020h	N.A.	—	—	For code migration without software enhancements, the code at this address should now be ORG'd to the PIC18CXXX High Priority Interrupt Vector Address (0x018)

FIGURE 18: INTERRUPT STRUCTURE BLOCK DIAGRAMS

PIC17CXXX



PIC18CXXX



Stack

The stack of the PIC17CXXX family is 16 levels deep. When the 17th item is loaded onto the stack, the content of stack level 1 is overwritten (circular buffer). In the PIC18CXXX family, the stack is 31 levels deep. When the 32nd item is loaded onto the stack, the content of stack level 31 is overwritten (stack pointer becomes stuck at 31).

The PIC17CXXX has a stack available bit (STKAV), which indicates if the stack pointer is pointing to the top of stack, or if the stack has rolled over. The PIC18CXXX has 2 bits, which are used to specify if the stack is full (STKFUL) or if underflow (STKUNF) condition has occurred. A configuration bit (STVREN) specifies if these flags generate a device reset.

In the PIC18CXXX, the stack pointer is now memory mapped. This is useful in some applications, such as Real Time Operating Systems (RTOS). Utmost care should be taken if modifying the stack pointer and contents of the stack.

An enhancement of the PIC18CXXX is the implementation of the Fast Register Stack. The Fast Register Stack saves the contents of the WREG, STATUS, and BSR registers. This stack is one level deep for each register. This is useful for saving the status of these registers when you do a subroutine call (if interrupts are disabled), or for interrupts where nesting is not a possibility (do not use with low priority interrupts).

Figure 19 shows the operation of the PIC17CXXX stack, while Figure 20 shows the operation of the PIC18CXXX stack.

Migration Impact

When migrating code from the PIC17CXXX to the PIC18CXXX, one should only need to modify the application software in regards to stack overflows and underflows. If no stack overflow/underflow checking was implemented, then there are no code migration issues due to the hardware stack.

FIGURE 19: PIC17CXXX STACK OPERATION

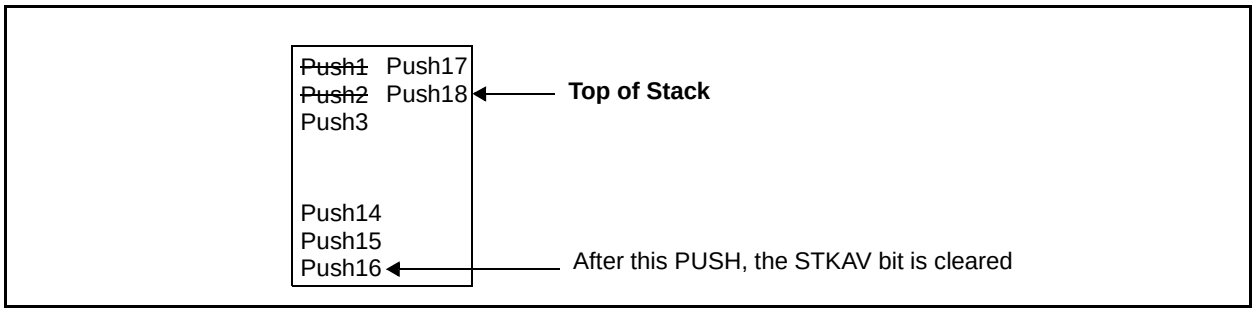
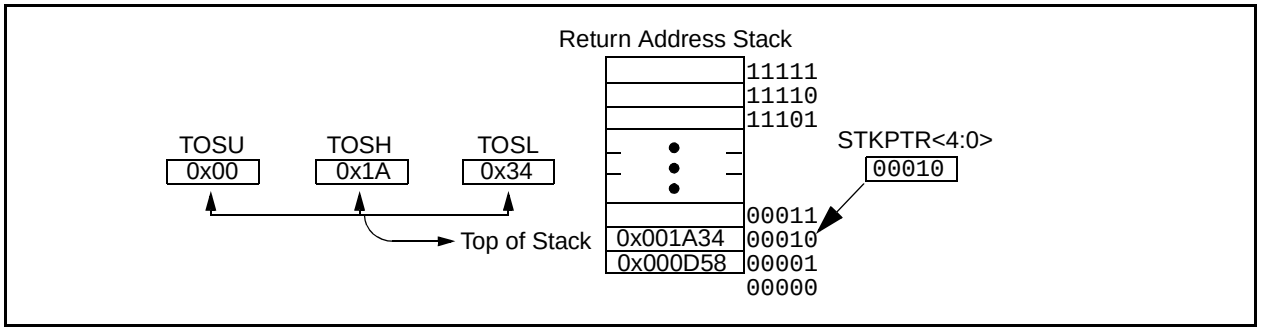


FIGURE 20: PIC18CXXX STACK OPERATION



Indirect Addressing

The PIC17CXXX family has 2 indirect addressing pointers (registers) called FSR0 and FSR1. Each pointer uses an 8-bit register. This allows the indirect addressing to occur anywhere in the selected banks of data memory (SFR bank and GPR bank).

The PIC18CXXX family has 3 indirect addressing pointers (registers) called FSR0, FSR1 and FSR2. Each pointer uses an 12-bit register. This allows the indirect addressing to occur anywhere in the data memory map.

Table 11 shows a comparison of the Indirect Addressing capabilities and operation.

Migration Impact

From the PIC17CXXX code, ensure that the current value of the BSR<3:0> is loaded into the high byte of the FSR register in the PIC18CXXX. This will ensure that the data memory access is in the correct bank.

Also, if any of the FSR automatic increment/decrement features are used (through the manipulation of the PIC17CXXX ALUSTA control bits), the appropriate indirect addressing register needs to be selected in the PIC18CXXX code.

Any indirect accesses to the PIC17CXXX SFRs would require that the PIC18CXXX FSRxH register be loaded with 0x0F. This makes the indirect addresses occur in bank 15.

TABLE 11: INDIRECT ADDRESSING COMPARISON

Feature	PIC17CXXX	PIC18CXXX	Comment
Number of FSR registers	2	3	
FSRx Register Size	8-bits	12-bits	
BSR specifies Bank(s)	Yes	No	PIC17CXXX specifies both SFR and GPR banks
FSR Memory Reach	256 Bytes	4096 Bytes	PIC18CXXX can access entire memory range, PIC17CXXX can access only in selected banks (SFR and GPR banks).
Instruction to load value into FSRx register	No	Yes	LFSR instruction is a 2 word 2 cycle instruction
FSR Pre-increment support	No	Yes	PIC18CXXX operation determined by register addressed (register PREINCx)
FSR Post-increment support	Yes	Yes	PIC18CXXX operation determined by register addressed (register POSTINCx). PIC17CXXX operation determined by control bits (FS3:FS2 and FS1:FS0 in register ALUSTA)
FSR Post-decrement support	Yes	Yes	PIC18CXXX operation determined by register addressed (register POSTDECx) PIC17CXXX operation determined by control bits (FS3:FS2 and FS1:FS0 in register ALUSTA)
FSR with Offset support	No	Yes	PIC18CXXX operation determined by register addressed (register PLUSWx)
ALUSTA control bits (FS3:FS2 and FS1:FS0) for Indirect Addressing Operation	Yes	No	PIC18CXXX operation determined by register addressed

LAYOUT

The pinout of the devices will need to be compared on an individual basis. The first PIC18CXXX devices (PIC18CXX2) are design to be footprint/functional compatible with some of the 28- and 40-pin Mid-Range devices. These devices are therefore not footprint compatible with the existing PIC17CXXX devices. This means that a revision of the board layout will be required.

Future PIC18CXXX devices may be footprint compatible, but a pin-by-pin comparison is required to ensure footprint/functional compatibility with the desired PIC17CXXX device. This functional compatibility does not ensure a compatibility with regards to the electrical characteristics of the device (such as I/O pin V_{IL}/V_{IH} characteristics or signal timings).

Migration Impact

A new layout is currently required for all migrations from the PIC17CXXX devices to the PIC18CXXX devices. Future PIC18CXXX devices may be specified that are footprint compatible with PIC17CXXX devices

CODING TECHNIQUES

The conversion process is aided when the initial code was written symbolically. That is, register names, bit names, and address labels are used in the source code as opposed to the hard coded values.

Example 1 shows the technique for using symbols for register and bit definitions, while Example 2 shows labels being used to specify addresses.

EXAMPLE 1: CODE TECHNIQUE #1

```
                BSF     3,2             ; Bad Programmer

                BSF     STATUS, Z       ; Good Programmer
```

EXAMPLE 2: CODE TECHNIQUE #2

```
                GOTO    0x0934          ; Bad Programmer

                GOTO    MY_Routine      ; Good Programmer
                :
                :
MY_Routine      :                       ; This is at address 0x0934
```

EXAMPLE CODE CONVERSION

Appendix A is a code conversion from a code segment found in Application Note AN547, Serial Port Utilities. The code segment was source file SERINT.ASM. The source file includes indications in the comments for each source code line that was changed. This is shown by a comment as follows:

```
;*****.
```

This was done to easily indicate each line that required a change, and specify the change that was implemented to make the source code compatible with the PIC18CXXX assembler.

CONCLUSION

Understanding the issues in a code conversion from one device to another is very important for assuring a smooth conversion process. This document hopefully has given you insight into where to inspect your code during the conversion process.

One of the main architectural goals of the PIC18CXXX family is that of source code compatibility foremost with the PICmicro Mid-Range Architecture and then with the High-End Architecture. The different implementation of peripheral and architectural features are the biggest hurdle. Since some of these peripheral modules and architectural features are implemented differently between the two families, this directly affects the ease of the conversion process. Conversions may require minimal effort in most applications, but there will be features (such as time based functions) that may require a full source code rewrite. Depending on the application of these functions, this rewrite may be relatively minor or fairly involved.

APPENDIX A: EXAMPLE CODE CONVERSION

EXAMPLE 1: CONVERTED SOURCE CODE EXAMPLE

```
;        TITLE   'Serial Interface Routines
;        PROCESSOR     18C452
;
;This is a short program to demonstrate how to transmit and receive
;serial data using the PIC18C452.
;
;A message will be transmitted and routed right back to the processor
;and read. The read information will be saved in an internal buffer.
;
;       Program:          18C_SER.ASM
;       Revision Date:
;                         7-02-99   Conversion to PIC18Cxx2 code.
;                                      Converted from PIC17C42 SERINT.ASM 1-22-97
;                                   as found in AN547 (DS00547C)
;
;
        LIST     P = 18C452

#include    <p18c452.inc>           ;***** Changed the include file
;
;***  These Registers may be remapped to allow the other application software
;***  to take advantage of the access RAM in Bank 0.
;
TX_BUFFER       equ     0x80
RX_BUFFER       equ     0xB0
RXPTR           equ     0x20
TXPTR           equ     0x21
SERFLAG         equ     0x22
RTINUM          equ     0x23
;
;  Status Bits used with user registers
;
TXDONE          equ     0
RXDONE          equ     1
HILOB           equ     2
;
;
```

```
        ORG     0
        goto    start
;
;   Changes ORG directives to point to new High and Low priority interrupts
;   Removed origines to TMRO, T0CKI, and Peripheral interrupts.
;
;        ORG     0x0010          ;vector for rtcc interrupt
                                 ;***** No Longer an separate TMRO Interrupt Vector Address
;rtcc_int                        ;not used here
;
        ORG     0x0008          ;vector for peripheral interrupt
                                 ;***** Vector Address changed from 0x0020
perf_int
        goto    service_perf    ;service the interrupts
;
        ORG     0x0030
;
;initialize the serial port: baud rate interrupts etc.
init_serial
        clrf    SERFLAG         ;clear all flags
                                 ;***** REMOVED ', F'
;        movlb   0               ;***** REMOVE
        movlw   0x07            ;select 9600 baud
        MOVWF   SPBRG           ;***** Change MOVFP tp MOVWF
        movlw   0x90            ;set up serial pins
        MOVWF   RCSTA           ;***** Change MOVFP tp MOVWF
        clrf    TXSTA           ;setup transmit status
                                 ;***** REMOVED ', F'
;        movlb   1               ;***** REMOVE
        clrf    PIR1            ;clear all interrupts
                                 ;***** REMOVED ', F', Changed PIR -> PIR1
        clrf    PIE1            ;clear all enables
                                 ;***** REMOVED ', F', Changed PIE -> PIE1
        bsf     PIE1,RCIE       ;enable receive interrupt
                                 ;*****    Changed PIE -> PIE1
        movlw   RX_BUFFER       ;set pointer to rx buffer
        MOVWF   RXPTR           ;***** Change MOVPF tp MOVWF
        clrf    INTCON          ;clear all interrupts
                                 ;***** REMOVED ', F', INTSTA -> INTCON
        bsf     INTCON,PEIE     ;enable peripheral ints
                                 ;***** INTSTA -> INTCON
        retfie
;
;start transmission of first two bytes
start_xmit
;        movlb   0               ;***** REMOVE
        bsf     TXSTA,TXEN      ;enable transmit
;        tablrd  1,1,W           ;load latch           ;***** REPLACED
;        tlrd    1,TXREG         ;load high byte       ;***** REPLACED
         TBLRD*+                 ;***** Due to New Implementation of Table Read function
         MOVFF   TABLAT, TXREG   ;***** Due to New Implementation of Table Read function
;        movlb   1               ;***** REMOVE
empty_chk
        btfss   PIR1,TXIF       ;TXBUF empty?
                                 ;*****    Changed PIR -> PIR1
        goto    empty_chk       ;no then keep checking
;        movlb   0               ;***** REMOVE
;        tablrd  0,1,TXREG       ;load lo byte         ;***** REPLACED
         TBLRD*+                 ;***** Due to New Implementation of Table Read function
         MOVFF   TABLAT, TXREG   ;***** Due to New Implementation of Table Read function
;        movlb   1               ;***** REMOVE
        bsf     PIE1,TXIE       ;enable transmit interrupts
                                 ;*****    Changed PIE -> PIE1
        bsf     SERFLAG,HILOB   ;set up next for high byte
        return
;
```

```
;
        PAGE
;
service_perf
;check for transmit or receive interrupts only
        btfsc   PIR1,RCIF       ;RX buffer full?
                                ;*****    Changed PIR -> PIR1
        goto    service_recv    ;yes then service
        btfss   PIR1,TXIF       ;TX buffer empty?
                                ;*****    Changed PIR -> PIR1
        goto    exit_perf       ;no, ignore other int.
service_xmt
        btfsc   SERFLAG,TXDONE  ;all done?
        goto    exit_perf       ;yes then quit
        btfsc   SERFLAG,HILOB   ;if clr, do low byte
        goto    rd_hi           ;else read high byte
;        tablrd  0,1,W           ;read lo                       ;***** REPLACE
        TBLRD*+                 ;***** Due to New Implementation of Table Read function
        goto    sx_cont         ;continue
rd_hi
;        tlrd    1,W             ;read high byte                ;***** REPLACE
        TBLRD*+                 ;***** Due to New Implementation of Table Read function
sx_cont
        btg     SERFLAG,HILOB   ;toggle flag
;        movlb   0               ;***** REMOVE
;        MOVPF   TXREG           ;***** REMOVE, TXREG loaded by next instruction
        MOVFF   TABLAT, TXREG   ;***** Due to New Implementation of Table Read function
        tstfsz  W               ;last byte?
        goto    exit_perf       ;no then cont
end_xmt                         ;else end transmit
;        movlb   1               ;***** REMOVE
        bcf     PIE1,TXIE       ;disable tx interrupt
                                ;******     Changed PIE -> PIE1
        bsf     SERFLAG,TXDONE  ;set done flag
exit_perf
;        bcf     INTSTA,PEIF     ;***** REMOVE,   clear peripheral int
                                ;*****    This instruction was never needed
        retfie
;
service_recv
        btfsc   SERFLAG,RXDONE  ;RX complete?
        goto    exit_perf       ;exit int
        MOVFF   RXPTR, FSR0L    ;***** Change MOVFP to MOVFF and FSR0 to FSR0L
;        movlb   0               ;***** REMOVE
        MOVFF   RCREG,INDF0     ;***** Change MOVFP to MOVFF
        clrf    WREG            ;clr W
                                ;***** REMOVED ', F'
        cpfsgt  INDF0           ;value = 0?
        goto    end_recv        ;yes then end
        incf    FSR0L, F        ;inc pointer
                               ;***** REMOVED ', F', Changed FSR0 to FSR0L and specified desti-
nation
        MOVFF   FSR0L, RXPTR    ;***** Change MOVFP to MOVFF and FSR0 to FSR0L
        goto    exit_perf       ;return from int
end_recv
        bsf     SERFLAG,RXDONE  ;set flag
        clrf    INTCON          ;clear all int
                                ;***** REMOVED ', F', INTSTA -> INTCON
;        movlb   1               ;***** REMOVE
        bcf     PIE1,RCIE       ;disable rx interrupts
                                ;*****     Changed PIE -> PIE1
        goto    exit_perf       ;return
        PAGE
```

```
;
start
        clrf    FSR1L           ;assign FSR1 as S.P.
                                ;***** REMOVED ', F' and Changed FSR0 to FSR0L
        decf    FSR1L, F        ;       /
                                ;***** REMOVED ', F', Changed FSR0 to FSR0L and
                                ;*****      specified destination
        movlw   0x20            ;clear ram space
        MOVWF   FSR0L           ;***** Change MOVFP to MOVWF and FSR0 to FSR0L
start1
        clrf    INDF0              ;clear ram
                                   ;***** REMOVED ', F'
        incfsz  FSR0L, F           ;inc and skip if done
                                   ;***** REMOVED ', F', Changed FSR0 to FSR0L and
                                   ;*****      specified destination
        goto    start1
        call    init_serial        ;initialize serial port
        movlw   LOW   MESSAGE      ;load table pointer
        MOVWF   TBLPTRL            ;***** Change MOVPF tp MOVWF
        movlw   HIGH  MESSAGE      ;     /
        MOVWF   TBLPTRH            ;***** Change MOVPF tp MOVWF
        CLRF    TBLPTRU            ;***** ADDED this instruction due to larger memory
                                   ;*****     space of PIC18Cxxx Architecture
        call    start_xmit         ;start transmission
chk_end
        btfss   SERFLAG,RXDONE     ;receive all?
        goto    chk_end            ;no then keep checking
;
loop    goto    loop               ;spin wheel
;
        ORG     0x100
MESSAGE
        DATA    "The code is: Tea for the Tillerman"
        DATA    0
;
;
        END
```

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV MSC The Netherlands Accredited by the RvA — DNV — ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02