INTRODUCTION
The PIC18CXX2 was intended to make conversions from midrange controllers to enhanced controllers as easily as possible. Changes to register and bit names, and bit locations were kept to a minimum. The PIC18CXX2 was designed to be pin-compatible with 28-pin and 40-pin midrange controllers.

This application note describes the minimum changes required to port code from the PIC16C74A to the PIC18C442, and are typical when migrating code from any midrange controller to any enhanced controller.

Changes to the PIC16C74A code largely consists of renaming registers and bits, moving bits to new registers, and placing variables into the appropriate places in RAM. Where additional features have been added, changes to code have been suggested.

OSCILLATOR OPERATION
Changes to the oscillator circuit or device configuration with respect to the oscillator mode may be required. Oscillator performance should be verified to ensure that it starts and operates as expected.

Crystal oscillator modes may require changes in loading capacitors and/or oscillator mode. RC oscillators may operate at a different frequency than expected on the PIC18C442 (when using the same components).

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.

OSCILLATOR MODES
The PIC18C442 has more oscillator modes than the PIC16C74A. Two RC modes affect the use of one pin. OSC1 is used for the RC oscillator, as before. OSC2 may be used either for clock output (Fosc/4) or as digital I/O pin RA6.

There are now two HS modes. The HS/PLL mode operates with a crystal or resonator from 4 MHz to 10 MHz while the HS mode can use a crystal or resonator up to 20 MHz.

At Power-On Reset, the OST (1024 oscillator cycles) and PWRT (optional 72 ms) delays occur as in the HS mode. When the PLL is enabled, an additional 2 ms delay is added to allow the Phase Locked Loop (PLL) to lock to the crystal frequency and stabilize.

HS/PLL mode uses a PLL to multiply the oscillator frequency by 4, providing an output up to 40 MHz to the instruction clock divider. This is divided to produce the Q clocks, resulting in the 10 MHz instruction clock rate.

When waking from SLEEP, the OST and 2 ms PLL delays are required for the oscillator and PLL to restart in HS/PLL mode.

CLOCK SWITCHING
The PIC18C442 now allows use of the Timer1 oscillator in place of the system oscillator. When operating from the Timer1 oscillator, the system oscillator is shut down as in SLEEP mode, and will require the same delays to restart as when waking from SLEEP. This allows continued operation at very low speed (Fcycle = 32 kHz/4 = 8 kHz), with power consumption almost as low as SLEEP mode. Clock switching is possible between any system oscillator mode and the Timer1 oscillator. However, the system oscillator mode cannot be changed.

Both of the following conditions must be met to allow clock switching:

1. The Timer1 oscillator must be enabled by setting the T1OSCEN bit in T1CON<3>.
2. Clock switching must be enabled by clearing the OSCSEN bit in CONFIG1H<5>.

Note 1: Even though compatible devices are tested to the same electrical specifications, the device characteristics may have changed due to changes in the manufacturing process. These differences should not affect systems that were designed well within the device specifications. For systems that operate close to or outside the specification limits, manufacturing differences may cause the device to behave differently.

Note 2: 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.
The actual clock switching is performed by operating the SCS bit, OSCCON<0>. Clearing the SCS bit causes the controller oscillator to be used for the controller clock, while setting the SCS bit causes the Timer1 oscillator to be used.

Peripherals that depend on the system clock for timing will be affected by the change in clock frequency.

**INSTRUCTION CHANGES**

When migrating code from midrange to enhanced controllers, the user must become aware of changes to the instruction set and make appropriate changes to their code. Usually, this requires examining the points in the code where program execution branches, depending on the state of a STATUS bit. Sometimes new instructions can simplify existing code.

Appendix C lists instructions for which status bit operation has changed from the PIC16C74A to the PIC18C422. Table C-1 lists the instructions that are carried over from the PIC16C74A, but generally affect new status bits. Table C-2 lists the instructions that were not carried over, and provides replacement instructions. Table C-3 lists new instructions, the status bits they affect or detect, and a short description of what the instruction does.

**RAM**

In the PIC16C74A, all variables and Special Function Registers (SFR's) are stored in two banks. Each bank can provide up to 128 addresses. The first 32 locations in each bank are reserved for SFR's while the remainder is used for GPR's in RAM.

The PIC18C442 data memory is grouped into banks of 256 bytes each. All SFR's are contained in bank 15. The PIC18C442 can store all variables in bank 0 with the same addresses used in the PIC16C74A. Addresses should be 12-bits long, and have the form “0xnn” (the first 4 bits are always 0 for bank 0, nn = address used in the PIC16C74A).

The BANKSEL directive and banking instructions of the PIC16C74A are no longer required in the PIC18C442, as all data variables can be stored in bank 0. References to the BANKSEL directive can be commented out, removed, or left in place “as is”. As data memory requirements grow, either the BANKSEL directive or firmware can modify the BSR to select the correct bank.

1. Ensure that all RAM variables have 12-bit addresses assigned to them, and are located in bank 0.
2. When the code clears STATUS bits RP1 and RP0 for the first time, replace these lines with clrf BSR.
3. Comment out or remove all other references to RP0 or RP1.
4. Comment out or remove all references to the BANKSEL directive (optional).

Access banking is automatically used when an operand has a 12-bit address below 0x080 (GPRs in the bottom half of bank 0), or at and above 0x080 (SFRs in the top half of bank 15). When accessing operands in the access bank range (0x000–0x0FF or 0xF80–0xFFF), the BSR is ignored. Addresses in the lower half of the access range access bank 0, while addresses in the upper half of the access range access bank 15.

In the PIC16C74A, SFRs were contained in the lower 32 locations of each bank. In the PIC18C442, all SFRs are located in the top portion of bank 15, and have 12-bit addresses assigned to them (0xF80 to 0xFFF). When accessing the SFRs, their symbolic names should be used. Access banking will ignore the BSR and automatically select bank 15 to access SFRs. Most SFR names are unchanged or very similar to those in the PIC16C74A. (See Appendix A.)

The FSR register is replaced by the register pair FSR0H:FSR0L, and contains the entire 12-bit address required to access any SFR or GPR in any bank. Change FSR to FSR0L. FSR0H should be cleared.

1. If the FSR is used to access SFRs, replace the corresponding INDF operand with the SFR name.
2. Replace all bsf STATUS,IRP instructions and bcf STATUS,IRP instructions with clrf FSR0H.
3. Replace all occurrences of BANKSEL var-name with clrf FSR0H.
4. Change FSR to FSR0L.
5. Change INDF to INDF0.

If FSR0L is incremented or decremented using INCF or DECF instructions beyond 8 bits, FSR0L will roll over, and the STATUS bits will be affected accordingly. FSR0H will not be affected by the rollover (see Example 1).

**EXAMPLE 1: INCREMENTING FSR0L AND OVERFLOW**

```
clrf FSR0H ; clear FSR0H
movlw 0xFF
movwf FSR0L ; load FSR0L with max ; 8-bit count
incf FSR0L, F ; FSR0L incremented to 0x00
; STATUS,C=1 STATUS,Z=1
; FSR0H=0x00 (no change)
```
PROGRAM MEMORY

The PIC16C74A uses a 13-bit program counter to address program memory words. The PIC18C442 uses a 21-bit program counter to address program memory bytes and counts by two when fetching program instructions. Attempts to write a 1 to PCL<0> will result in PCL<0>=0. Therefore, the program counter will always access program memory with the address LSb always set to 0.

Writing to PCL will cause PCLATH and PCLATU to be written to PCH and PCU, respectively, as in the PIC16C74A. However, a read of PCL will update PCLATH and PCLATU from PCH and PCU.

The CALL and GOTO instructions contain all the addressing information required. Manipulation of PCLATH to prepare for jumps is not required, and will have no practical effect. Such code may be left in the program, commented out, or removed.

Instructions with $

Occasionally, programmers will use a '$' symbol to indicate the program address of the current opcode. '$' by itself still functions as before. However, since the program counter now addresses bytes instead of words, any offsets added to '$' need to be doubled to refer to the correct address (see Example 2).

EXAMPLE 2: $ OFFSETS DOUBLED

```
goto $-6 ; replaces goto $-3
goto $+0x2E ; replaces goto $+0x17
```

EXAMPLE 3: MODIFIED ROUTINE TO SUPPORT 256 ENTRY TABLE

```
movf OFFSET, w ; OFFSET=0x00 to 0xFF
call table
...

ORG 0x3C0

table movwf taboff ; save table offset
bcf STATUS, C ; clear STATUS bit
rcf taboff, F ; multiply by 2, save in taboff

movlw HIGH(table_st1) ; get high byte of table start
btfsc STATUS, C ; test carry bit
incf WREG, W
movwf PCLATH ; modify PCLATH if required

movlw LOW(table_st1) ; get low byte of table address
addwf taboff, w ; add in offset
btfsc STATUS, C ; test for overflow
incf PCLATH, F ; increment if needed
movwf PCL ; make jump, PCLATH and PCLATU are written to PCH and PCU

tab_st1 retlw 0x00 ; table body, first entry, offset=0
retlw 0x01 ; (256 entry, 256 word/512 byte)
...
```
TABLES IN PROGRAM MEMORY

Computed GOTOs using RETLW tables can be performed on the PIC18C442, but this allows only one byte of data to be stored in each program memory instruction (16-bits), and limits the table size to 256 entries.

Table operations allow two 8-bit bytes of data to be stored in each program memory word, doubling table data density. There is no limit to the number of table entries, up to the maximum program memory. Program memory can also be read to calculate a program checksum to verify program integrity.

Table Reads

A 21 bit table pointer to program memory is loaded with the address of the data byte to be read. This pointer is stored in TBLPTRU<4:0>, TBLPTRH<7:0>, and TBLPTRL<7:0>. A TBLRD* instruction causes the data at that address to be placed into TABLAT where the program can use it as data.

TBLPTRU<4:0>, TBLPTRH<7:0>, and TBLPTRL<7:0> are SFRs in memory space. TBLPTRL<0> need not always be 0 as with PCL<0>. The TBLPTR group of registers can be automatically incremented or decremented using variations of the TBLRD* instruction.

Table 1 shows the instructions and their effects on TABLAT and TBLPTR. Pointer increment/decrement operations affect all 21 bits of the TBLPTR registers.

### Table 1: Table Read Instructions and Effects on the Table Pointer

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBLRD*</td>
<td>Places copy of program memory byte in TABLAT</td>
</tr>
<tr>
<td>TBLRD++</td>
<td>Places copy of program memory byte in TABLAT Increments TBLPTR after read</td>
</tr>
<tr>
<td>TBLRD--</td>
<td>Places copy of program memory byte in TABLAT Decrements TBLPTR after read</td>
</tr>
<tr>
<td>TBLRD++</td>
<td>Increments TBLPTR before read Places copy of program memory byte in TABLAT</td>
</tr>
</tbody>
</table>

#### Steps for Table Reads

The steps to perform a table read from program memory are:

1. Set the table pointer to the desired byte address. (TBLPTR may be even or odd as required.)
2. Execute a TBLRD instruction.
3. Read the byte retrieved from program memory in TABLAT.

#### Code for Table Reads

An example of table read code is shown in Example 4.

```assembly
EXAMPLE 4: TABLE READ CODE

RdStr movlw HIGH(string) ; load high byte of pointer (0x12)
            movwf TBLPTRH
movlw LOW(string) ; load low byte of pointer (0x34)
movwf TBLPTRL
read tblrd++ ; read byte from program memory, ; and increment pointer one byte
           movff TABLAT,PORTB ; move byte from table latch to output port B
            tstfss TABLAT ; was retrieved byte a null?
goto read ; no, do loop again
           return

ORG 0x1234
String DW "This is a test.",0x00 ; text string
```

---

© 1999 Microchip Technology Inc.

Preliminary
INTERRUPTS

The PIC18C442 resets with the interrupt structure in a PIC16C74A compatible mode. The interrupt vector origin has been changed from 0x0004 in the PIC16C74A to 0x0008 in the PIC18C442.

Interrupt Pins

The RB0/INT pin on the PIC16C74A has been renamed to RB0/INT0. The PIC18C442 offers 2 additional interrupt pins, RB1 and RB2 have been renamed RB1/INT1 and RB2/INT2 to support the additional interrupt functions.

Interrupt Handling Registers

The INTCON register is mostly unchanged. Bits INTE and INTF are renamed INTOIE and INTOIF, respectively.

With respect to interrupt control, all other register and bit names, and bit locations in the PIC16C74A are unchanged in the PIC18C442.

Principle

The PIC18C442 also offers 2 levels of interrupt priority, each with its own interrupt vector. Interrupts assigned high priority take the high priority interrupt vector at 0x0008, while interrupts assigned low priority take the low priority interrupt vector at 0x0018. Interrupt priority is enabled by setting IPEN, RCON<7> (Interrupt Priority Enable).

When IPEN is set, interrupt priority is enabled. The functions of GIE, INTOE<7> and PEIE, INTOE<6> are modified. GIE becomes GIEH (Global Interrupt Enable High) and PEIE becomes GIEL (Global Interrupt Enable Low). If GIEL is clear, all low priority interrupts are disabled. If GIEH is clear, all high and low priority interrupts are disabled.

Each interrupt source has an associated interrupt priority bit to set its priority. When set, interrupt priority is high. When clear, interrupt priority is low. If an interrupt source has an interrupt flag bit in PIR1 or PIR2, corresponding interrupt priority bits will be located in IPR1 or IPR2. The remaining priority bits are located in INTOCON2 and INTOCON3.

The interrupt priorities for Timer0 and RBIF are set using TMR0IP, INTOCON<2>, and RBIP, INTOCON<2>.

Interrupt priorities for the INT1 and INT2 pins are set using INTO1IP, INTOCON3<6>, and INTO2IP, INTOCON3<7>. Clearing these bits will select low priority interrupts.

The interrupt enable bits for INT1 and INT2 are set using INTO1E, INTOCON3<3>, and INTO2E, INTOCON3<4>. The corresponding interrupt flag bits are INTO1F, INTOCON3<3> and INTO2F, INTOCON3<4>.

Return Address Stack

The PIC18C442 stack has a 31 level stack instead of the 8 level stack in the PIC16C74A. The PIC18C442 also allows access to the stack pointer, stack error bits, and top-of-stack contents. PUSH and POP instructions have been added to manipulate the stack contents and stack pointer.

The PIC16C74A stack is 8 levels deep, and functions as a linear buffer. Pushes beyond the 8th push overwrite the 1st push, 2nd push, etc. Pops beyond the 1st push begin returning the 8th push, 7th push, etc. There is no indication of the state of the stack. The stack contents are not available to the program.

The PIC18C442 stack is 31 levels deep, and functions as a linear buffer. The 31st push will set the stack overflow status bit STKFUL, STKPTR<7>. The 32nd push will overwrite the 31st push. All pops beyond the 1st push return 0x0000, set the stack underflow status bit STKUNF, STKPTR<6>, and restarts the program (but does not reset the device). Optionally, a device reset can occur when the stack overflow and underflow status bits are set (see RESETs). The STKFUL and STKUNF bits are reset only by a POR or by software. This allows the program to respond to stack errors.

When the controller is initialized, the stack pointer STKPTR contains 0x00, and points to a stack address that contains 0x0000, which is the RESET vector. A PUS or CALL instruction, or an interrupt will increment the stack pointer to the next higher stack location to become the new top-of-stack where the PC is then stored. A return instruction (RETURN or RETI) will move the contents of the top-of-stack to the PC, and
decrement the stack pointer. The **POP** instruction simply decrements the stack pointer, discarding the contents of the top-of-stack.

The 21-bit top-of-stack can be accessed through the top-of-stack registers TOSU<4:0>, TOSH<7:0>, and TOSL<7:0>. The top-of-stack is readable and writable, allowing data to be stored and retrieved using the stack.

**Fast Register Stack**

The fast register stack is a group of registers that saves the contents of the WREG, STATUS, and BSR registers every time an interrupt or subroutine call occurs. This stack is one level deep and is not accessible to the user. When a return with the fast option ([**retfie FAST**](#)) is executed, the contents of the fast register stack are restored back to the WREG, STATUS, and BSR registers.

Calls may use the Fast Register Stack. A call with the fast stack option (call **label, FAST**) saves the WREG, STATUS, and BSR registers to the fast register stack. A corresponding return is required to restore these registers ([**return FAST**](#)).

If interrupts are enabled, the fast register stack cannot be used for a return from a call. If an interrupt occurs during a called subroutine, the contents of the fast register stack will still contain the WREG, STATUS, and BSR contents from when the interrupt was executed. If a subroutine should attempt to return using the fast register stack, an improper context will be restored.

If interrupt priority is enabled, only high priority interrupts can use the fast register stack. High priority interrupts may interrupt low priority interrupts at any time.

**RESETS**

The PIC18C442 responds to all the same RESET sources as the PIC16C74A.

The PCON register has been renamed to RCON. The TO and PD bits from the STATUS register have been moved to RCON. All bits retain the same functions in the PIC18C442 as they had in the PIC16C74A.

**Power-On Reset**

The PCON register of the PIC16C74A has been renamed RCON and contains the **BOR** bit. Operation of the **BOR** bit is unchanged.

The Brown-Out Reset (**BOR** module) can be configured as enabled or disabled in the PIC18C442 as in the PIC16C74A. When **BOR** is enabled, the Power-up Timer (PWRT) is also automatically enabled. The state of the PWRT enable bit, PWRTEN, CONFIG2L<0> is ignored. However, the PIC18C442 offers four **BOR** thresholds instead of one, and is selected using BORV1:BORV0, CONFIG2L<3:2>. The time that VDD must remain below VBOR (Parameter D005) has increased from the PIC16C74A (Parameter 35, TBOR).

1. Change **PCON** to **RCON**
2. Change **STATUS, NOT_TO** to **RCON, NOT_TO**
3. Change **STATUS, NOT_PD** to **RCON, NOT_PD**
4. Select V<sub>BO</sub>R threshold in configuration (BORV1:BORV0, CONFIG2L<3:2>)

**MCLR**

MCLR on the PIC18C442 operates the same as the PIC16C74A. No changes to the code or circuit are required.

**WDT**

In the PIC18C442, the WDT now has its own postscaler, independent of the Timer0 prescaler. The WDT is enabled when WDTEN, CONFIG2H<0> is set, disabled when clear. The WDT postscaler is programmed using WDTPS2:WDTPS0, CONFIG2H<3:1> to select a ratio from 1:1 to 1:128.

TO, STATUS<4>, and PD, STATUS<3> bits have been moved to TO, RCON<3> and PD, RCON<2>. The operation of these bits is unchanged.

If the WDT has been disabled by clearing WDTEN, the WDT may be enabled under software control by setting SWDTE, WDTCON<0>, and disabled by clearing this bit. The WDT postscaler ratio cannot be changed.

If the WDT is enabled using WDTEN, then changing SWDTE will have no effect.

**Stack Over/Underflow**

The PIC16C74A has an 8 level stack. Once the PC has been pushed to the stack 8 times, a 9th push would overwrite the 1st stack location without any errors being generated. The PIC18C442 uses a 31 level stack. When the stack is almost full (30 pushes), and another push occurs, the 31st push sets the STKFUL status bit. The 32nd push overwrites the 31st push.

Conversely, the stack is empty (all pushes have been popped) and another pop occurs, the STKUNF bit is set and the PC is loaded with the RESET vector address. This does not reset the controller, but does restart the code.
The device can be configured to perform a reset when either the STKFUL or STKUNF bits are set. Setting SVTREN, CONFIG4L<0> will allow the stack error bits to reset the controller. The STKFUL and STKUNF bits are cleared only by a POR or by software. A reset caused by a stack error will not clear these bits. To determine the cause of this reset, the user will have to poll the STKFUL and STKUNF bits and clear them when taking corrective action.

**RESET Instruction**

The PIC18C442 offers a *RESET* instruction. This instruction performs a device reset similar to a MCLR reset. All peripherals are reset and program execution resumes from the reset vector.

The RCON register contains the RI bit. The RI bit is set by POR, BOR, and WDT resets, and is cleared by the *RESET* instruction. The TO, PD, BOR, and POR bits are unaffected. By polling the RI bit, RCON<4>, the reason for this reset can be determined.

**TIMER0**

The PIC18C442 Timer0 resets to a mode identical to the PIC16C74A. The Timer0 count is read and written using TMR0L instead of TMR0. The OPTION_REG register has been renamed T0CON.

The PSA bit, T0CON<3>, now only enables the Timer0 prescaler when clear, and disables the Timer0 prescaler when set (same effect as in the PIC16C74A with respect to Timer0).

If a different Timer0 prescaler ratio is required, PS2:PS0, OPTION<2:0> has been replaced by T0PS2:T0PS0, T0CON<2:0>, which functions identically with respect to Timer0.

Timer0 will set its interrupt flag bit (T0IF) when TMR0L overflows (same as when Timer0 overflows in the PIC16C74A).

The WDT and its postscaler are unaffected by settings in T0CON. See the section on the WDT.

The RBPU bit, OPTION<7>, has been renamed and moved to RBPU, INTCON2<7>. INTEDG, OPTION<6> has been renamed and moved to INT0EDG, INTCON2<6>.

The changes required in code are:

1. Rename and move OPTION, NOT_RBPU to INTCON2, NOT_RBPU.
2. Rename and move OPTION_REG, INTEDG to INTCON2, INTEDG0.
3. Change OPTION_REG to T0CON.
4. Rename PS2:PS0 to T0PS2:T0PS0.
5. Operations modifying PS2:PS0 and PSA for the WDT are commented out. Modify CONFIG2H instead.
6. Timer0 reads/writes use TMR0L instead of TMR0.

Timer0 can operate as a 16-bit timer by clearing T08BIT, T0CON<6>. In this mode, TMR0H is written to the Timer0 high byte when TMR0L is written. TMR0H is updated from the Timer0 high byte when TMR0L is read. Timer0 interrupts will occur when Timer0 (in 16-bit mode) rolls over from 0xFFFF to 0x0000.

**TIMER1**

The PIC18C442 Timer1 module is upwardly compatible with the PIC16C74A Timer1 module.

When RD16, T1CON<7> is set, a read of TMR1L causes TMR1H to be updated from the Timer1 high byte. A write to TMR1L will cause the Timer1 high byte to be updated from TMR1H. In this mode, the user does not have to check to see if the low byte rolled over while reading the high byte, or stop the timer when loading it.

The Timer1 oscillator of the PIC18C442 is functionally identical to the PIC16C74A Timer1 oscillator. However, due to process changes, operation of the Timer1 oscillator should be verified to operate as expected.

**TIMER2**

The Timer2 module of the PIC18C442 is identical to the Timer2 module of the PIC16C74A. No code changes are required.

**TIMER3**

The PIC18C442 provides a fourth timer not present in the PIC16C74A. This timer is identical to Timer1 as implemented in the PIC18C442. Both timers can serve as a timebase for the CCP capture and compare functions, and may use the Timer1 oscillator. Both may be reset by the CCP compare special event trigger.

**CAPTURE/COMPARE/PWM**

The capture, compare, and PWM functions of the PIC16C74A are fully compatible with the PIC18C442. No code changes or circuit modifications are required. The PIC18C442 CCP module offers an extra mode not present in the PIC16C74A. The compare mode can toggle the CCP output pin on a match.
A/D

The A/D module on the PIC18C442 resets to the same state as in the PIC16C74A. Code written for the PIC16C74A will run with only one change on the PIC18C442. Because the PIC18C442 has a 10-bit A/D module, two 8-bit registers are now required to make the 10-bit result available. ADRESH, and will contain the 8 MSbs of the result. As long as the user treats the ADCON0 and ADCON1 registers as if they were part of the PIC16C74A, the A/D module will function the same as in the PIC16C74A.

A new register and three new bits in ADCON1 offer some enhanced features over the PIC16C74A. ADRESL holds the additional bits of the 10-bit conversion result. ADFM, ADCON1<7> controls the justification of the 10-bit result in ADRESH:ADRESL. If the user wishes to use an 8-bit result, clear bit ADFM (RESET state, compatible with the PIC16C74A), and the 8 MSbs are placed in ADRESH. The remaining 2 bits are stored in ADRESL<7:6>. Bits ADRESL<6:0> will be cleared. This format allows the user to use the 8-bit result in ADRESH in 8-bit math operations.

If the user wishes to make use of all 10 bits, set bit ADFM. The 8 LSbs are stored in ADRESL<7:0> and the 2 MSbs of the result are stored in ADRESH.<1:0>. Bits ADRESH<7:2> are cleared. This format is useful for taking the 10-bit result from ADRESH:ADRESL, and using it "as is" in 16-bit math operations.

The instruction clock can now run at 10 MHz (100 nS) when a 10 MHz oscillator drives the PLL. The clock sources available in ADCS1:ADCS0, ADCON0<7:6> do not allow $T_{AD}$ to be set to a minimum of 1.6 μSec with such high clock speeds. A third A/D clock select bit is provided in ADCS2, ADCON1<6>. When ADCS2 is set, the instruction clock is divided by 2 allowing $T_{AD}$ to be set correctly. The internal A/D RC oscillator is not affected.

The user can use an external voltage reference for the A/D conversion in the PIC16C74A and the PIC18C442. The PIC18C442 also allows the use of a low reference voltage. Depending on the setting of PCFG3:PCFG0, ADCON1<3:0>, the user can select as references the controller supply and ground, an external high reference and the controller ground, or external high and low references.

The references can not exceed the controller supply rails, but can modify the conversion range by introducing a positive offset, and reducing the full scale input voltage. Check the electrical specifications for limits on the reference voltages (Parameters A20, A20A, A21, A22, and A25).

Example 5 shows how to initialize the A/D module.

EXAMPLE 5: A/D SETUP

This example shows how to initialize the A/D module for the following conditions:

- AN0 may be anywhere in the range of 0.5V to 3.5V.
- The input is currently at 2.296V. The controller is using an 8 MHz oscillator and the PLL. A 10-bit result is desired for use in 16-bit calculations.
- The clock source is selected by setting ADCS2:ADCS0 to B'110'.
- \( \text{ADCS2} = \text{B}'1'; \text{ADCS1:ADCS0} = \text{B}'10' \).
- The 32 MHz PLL output is divided by 64 to produce a 2.0 μSec $T_{AD}$.
- The port is configured to make AN0 and AN1 analog inputs, AN2 as the low reference ($V_{REF-}$) and AN3 as the high reference ($V_{REF+}$) by setting PCFG2:PCFG0 to B'1101'. 3.5V is applied to AN3, and 0.5V to AN2.

These references will apply to all conversions on all channels.

- The ADFM bit is set to select right justification of the result.

The AN0 channel is selected by setting CHS2:CHS0 to B'000', and the conversion started by setting the GO/DONE bit, ADCON0<2>. When the GO/DONE bit clears, ADRESH:ADRESL will contain 0x0265.

USART

The PIC16C74A USART is upwardly compatible with the PIC18C442 USART. No code changes are required.

The USART can monitor the serial data in 9-bit mode. Setting the ADDEN, RCSTA<3> bit causes the USART to generate an interrupt only when the 9th received data bit is set, instead of every time a new data byte has been received. RCREG is unchanged until the 9th bit is set. When set, RCREG is loaded with the received address.

This is useful for networks that indicate that the other 8 bits are a device address by setting the 9th bit. When the 9th bit is clear, data is being sent.

SSP

SSPCON has been renamed SSPCON1. Otherwise, the PIC16C74A SSP module is upwardly compatible with the PIC18C442 MSSP module.

SPI Mode

The PIC16C74A SSP module is fully compatible with the PIC18C442 with respect to SPI mode. No changes to code are required.
I²C Mode
The PIC16C74A SSP module is upwardly compatible with the PIC18C442 MSSP module. No changes to code are required for conversion.

The SSP module used in the PIC16C74A provides limited support for master mode I²C. The MSSP module is used in the PIC18C442, and supports I²C master and multi-master modes in hardware.

Master mode has been added. SSPCON2 has been added to support hardware master modes. Slave mode now provides general call support.

Master Mode
The master SSP module (MSSP) supports master mode I²C in hardware through the use of SSPCON2. Arbitration for multi-master operation is provided. The baud rate generator is used to generate SCL.

In the PIC16C74A, master mode was implemented in software that monitored and controlled the SCL and SDA pins. Start (S) and stop (P) bit interrupts where provided by SSP hardware.

The PIC18C442 still supports such operation without code changes. However, master and multi-master modes are now provided by hardware.

When master mode is selected (SSPM3:SSPM0, SSPCON1<3:0> = B"1011"), and SSPEN, SSPCON1<5> is set, the SSP module will control SCL and SDA. Data and slave addresses (7-bit or 10-bit) with R/W bit are sent in SSPBUF, and the baud rate is set in SSPADD<6:0>.

Only one operation can be completed at a time. Each operation must complete before the next can be started. Successful completion is indicated by setting SSPIF when the SSP module is becomes idle. Operations are start, repeated start, stop, sending 1 byte of data, receiving 1 byte of data, and sending an ACK/ NACK. If an attempt is made to program the next operation before the module becomes idle, the programming is ignored, the WCOL status bit, SSPCON1<7>, is set and SSPIF remains clear. WCOL is cleared by software. If the bus was busy, BCLIF, PIR2<3>, is set generating an interrupt.

Slave General Call Support
Slave general call support is enabled by setting GCEN, SSPCON2<7>. When an address match occurs (either slave address, or the general call address of 0x00 when GCEN is set), several actions occur.

- The received address is placed in the SSPBUF register
- The BUFFER FULL status bit BF, SSPSTAT<0>, is set
- An ACK pulse is generated
- An interrupt is generated by setting SSPIF, PIR1<3>

When the interrupt is serviced, SSPBUF must be read to determine if the interrupt was generated by a slave or general call address match.

When the slave is configured for 10-bit addresses with GCEN set, and the general call address is detected, the UA, SSPSTAT<1> bit will not be set. Instead, the slave will begin receiving data after the ACK is sent.

CONCLUSION
Conversion of a PIC16C74A application to run on a PIC18C442 consists of the following:

- Checking to make sure that the main and Timer1 (if used) oscillators work as expected
- Placing variables into bank 0 and assigning 12-bit addresses
- Modifying computed goto subroutines for reading tables
- Modifying the names and locations of bits and registers
- If the Brown-out Reset is enabled, select a BOR threshold

If desired, the user can make use of additional features offered by the PIC18C442. These are:

- Additional system oscillator modes
- System clock switching for reduced power requirements
- Additional memory (program and data)
- Data retrieval using program memory
- Additional interrupt pins
- Interrupt priority
- Stack access and status
- Optional stack error reset
- Fast register stack to save and restore context
- RESET instruction
- Timer0 operates as an 8 or 16-bit timer/counter
- A fourth timer, Timer3, that duplicates Timer1
- A 10-bit A/D module
- Full I²C master mode
APPENDIX A: CHANGED REGISTER/BIT LOCATIONS/NAMES

TABLE A-1: CHANGED REGISTER/BIT LOCATIONS/NAMES

<table>
<thead>
<tr>
<th>PIC16C74A</th>
<th>PIC18C442</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION_REG NOT_RBPU</td>
<td>INTCON2 NOT_RBPU</td>
<td></td>
</tr>
<tr>
<td>OPTION_REG INTEDG</td>
<td>INTCON2 INTEDG0</td>
<td></td>
</tr>
<tr>
<td>OPTION_REG TOCS</td>
<td>T0CON T0CS</td>
<td></td>
</tr>
<tr>
<td>OPTION_REG T0SE</td>
<td>T0CON T0SE</td>
<td></td>
</tr>
<tr>
<td>OPTION_REG PSA</td>
<td>T0CON PSA</td>
<td>WDT has own postscaler. Enabled using WDTEN, CONFIG2H&lt;0&gt;</td>
</tr>
<tr>
<td>OPTION_REG PS2</td>
<td>T0CON T0PS2</td>
<td>WDT postscaler set using WDTPS2, CONFIG2H&lt;3&gt;</td>
</tr>
<tr>
<td>OPTION_REG PS1</td>
<td>T0CON T0PS1</td>
<td>WDT postscaler set using WDTPS1, CONFIG2H&lt;2&gt;</td>
</tr>
<tr>
<td>OPTION_REG PS0</td>
<td>T0CON T0PS0</td>
<td>WDT postscaler set using WDTPS0, CONFIG2H&lt;1&gt;</td>
</tr>
<tr>
<td>PCON NOT_POR</td>
<td>RCON NOT_POR</td>
<td></td>
</tr>
<tr>
<td>PCON NOT_BOR</td>
<td>RCON NOT_BOR</td>
<td></td>
</tr>
<tr>
<td>STATUS NOT_TO</td>
<td>RCON NOT_TO</td>
<td></td>
</tr>
<tr>
<td>STATUS NOT_PD</td>
<td>RCON NOT_PD</td>
<td></td>
</tr>
<tr>
<td>INDF</td>
<td>—</td>
<td>INDF0 —</td>
</tr>
<tr>
<td>FSR</td>
<td>—</td>
<td>FSR0L —</td>
</tr>
<tr>
<td>TMR0</td>
<td>—</td>
<td>TMR0L —</td>
</tr>
<tr>
<td>SSPCON</td>
<td>—</td>
<td>SSPCON1 —</td>
</tr>
<tr>
<td>ADRES</td>
<td>—</td>
<td>ADRESH —</td>
</tr>
</tbody>
</table>

APPENDIX B: CODE CHANGES

; PIC18C442 code ; replaced PIC16C74A code

clrf BSR ; bcf STATUS,RP0
; bcf STATUS,RP1
; (first occurrence only)
; (otherwise comment out or remove)

movf INDF0,w ; movf INDF,w
movf TMRL0,w ; movf TMRL,w
movf FSR0L,w ; movf FSR0L,w
movf SSPCON1,w ; movf SSPCON1,w
movf ADRESH,w ; movf ADRESH,w
movf TOCON,w ; movf OPTION_REG,w (TIMER0 OPERATIONS ONLY)
movf RCON,w ; movf PCON,w
APPENDIX C: INSTRUCTION CHANGES FROM PIC16C74A TO PIC18C442

TABLE C-1: DIFFERENCES IN STATUS BIT OPERATION

<table>
<thead>
<tr>
<th>Instruction</th>
<th>STATUS Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16C74A</td>
<td>18C442</td>
</tr>
<tr>
<td>ADDLW</td>
<td>C, DC, Z</td>
<td>C, DC, Z, OV, N</td>
</tr>
<tr>
<td>ADDWF</td>
<td>C, DC, Z</td>
<td>C, DC, Z, OV, N</td>
</tr>
<tr>
<td>ANDLW</td>
<td>Z</td>
<td>Z, N</td>
</tr>
<tr>
<td>ANDWF</td>
<td>Z</td>
<td>Z, N</td>
</tr>
<tr>
<td>COMP</td>
<td>Z</td>
<td>Z, N</td>
</tr>
<tr>
<td>DECFS</td>
<td>Z</td>
<td>C, DC, Z, OV, N</td>
</tr>
<tr>
<td>INCFS</td>
<td>Z</td>
<td>C, DC, Z, OV, N</td>
</tr>
<tr>
<td>IGROLW</td>
<td>Z</td>
<td>Z, N</td>
</tr>
<tr>
<td>IGORSF</td>
<td>Z</td>
<td>Z, N</td>
</tr>
<tr>
<td>MOVF</td>
<td>Z</td>
<td>Z, N</td>
</tr>
<tr>
<td>RETFIE</td>
<td>GIE</td>
<td>GIE/GIEH, PEIE/GIEL</td>
</tr>
<tr>
<td>RLF</td>
<td>C, DC, Z</td>
<td>C, Z, N</td>
</tr>
<tr>
<td>RRFS</td>
<td>C, DC, Z</td>
<td>C, Z, N</td>
</tr>
<tr>
<td>SUBLW</td>
<td>C, DC, Z</td>
<td>C, DC, Z, OV, N</td>
</tr>
<tr>
<td>SUBRSF</td>
<td>C, DC, Z</td>
<td>C, DC, Z, OV, N</td>
</tr>
<tr>
<td>XORLW</td>
<td>Z</td>
<td>Z, N</td>
</tr>
<tr>
<td>XORRSF</td>
<td>Z</td>
<td>Z, N</td>
</tr>
</tbody>
</table>

Legend: STATUS bits in **Bold** are affected in code conversion from PIC16C74A to PIC18C442

TABLE C-2: INSTRUCTIONS NO LONGER SUPPORTED

<table>
<thead>
<tr>
<th>16C Instructions</th>
<th>16C STATUS Bits</th>
<th>Work-Around</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLRWF</td>
<td>Z</td>
<td>Use CLRF WREG instead</td>
</tr>
</tbody>
</table>

Legend: STATUS bits in **Bold** are affected in code conversion from PIC16C74A to PIC18C442
### TABLE C-3: NEW PIC18C442 INSTRUCTIONS

<table>
<thead>
<tr>
<th>Instructions</th>
<th>STATUS Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDWFC</td>
<td>C, DC, OV, N</td>
<td>Add WREG, F, and carry bit</td>
</tr>
<tr>
<td>BC</td>
<td></td>
<td>Conditional branch depending on carry STATUS bit</td>
</tr>
<tr>
<td>BN</td>
<td></td>
<td>Conditional branch depending on negative STATUS bit</td>
</tr>
<tr>
<td>BNC</td>
<td></td>
<td>Conditional branch depending on carry STATUS bit</td>
</tr>
<tr>
<td>BNN</td>
<td></td>
<td>Conditional branch depending on negative STATUS bit</td>
</tr>
<tr>
<td>BNV</td>
<td></td>
<td>Conditional branch depending on overflow STATUS bit</td>
</tr>
<tr>
<td>BNZ</td>
<td></td>
<td>Conditional branch depending on zero STATUS bit</td>
</tr>
<tr>
<td>BOV</td>
<td></td>
<td>Conditional branch depending on overflow STATUS bit</td>
</tr>
<tr>
<td>BRA</td>
<td></td>
<td>Unconditional branch</td>
</tr>
<tr>
<td>BZ</td>
<td></td>
<td>Conditional branch depending on zero STATUS bit</td>
</tr>
<tr>
<td>CPFSEQ</td>
<td></td>
<td>Branch depending on result of unsigned subtraction</td>
</tr>
<tr>
<td>CPFSGT</td>
<td></td>
<td>Branch depending on result of unsigned subtraction</td>
</tr>
<tr>
<td>CPFSLT</td>
<td></td>
<td>Branch depending on result of unsigned subtraction</td>
</tr>
<tr>
<td>DAW</td>
<td>C</td>
<td>Decimal Adjust WREG</td>
</tr>
<tr>
<td>DCFSNZ</td>
<td></td>
<td>Decrement file, skip next instruction if result is not zero</td>
</tr>
<tr>
<td>INFSNZ</td>
<td></td>
<td>Increment file, skip next instruction if result is not zero</td>
</tr>
<tr>
<td>LFSR</td>
<td></td>
<td>Move literal to file pointed to by FSR</td>
</tr>
<tr>
<td>MOVF</td>
<td>Z, N</td>
<td>Move file f</td>
</tr>
<tr>
<td>MOVFF</td>
<td></td>
<td>Move any file to any file</td>
</tr>
<tr>
<td>MOVLB</td>
<td></td>
<td>Move literal to BSR (Bank Select Register)</td>
</tr>
<tr>
<td>MULLW</td>
<td></td>
<td>Multiply literal with WREG</td>
</tr>
<tr>
<td>MULWF</td>
<td></td>
<td>Multiply WREG with file</td>
</tr>
<tr>
<td>NEGF</td>
<td>C, DC, OV, N</td>
<td>2’s complement, toggles sign of 8-bit signed data</td>
</tr>
<tr>
<td>POP</td>
<td></td>
<td>Discard top of stack, decrement stack pointer</td>
</tr>
<tr>
<td>PUSH</td>
<td></td>
<td>Increment stack pointer, copy PC to top of stack</td>
</tr>
<tr>
<td>RCALL</td>
<td></td>
<td>Call subroutine at offset</td>
</tr>
<tr>
<td>RESET</td>
<td>ALL</td>
<td>Reset controller</td>
</tr>
<tr>
<td>RLNCF</td>
<td>Z, N</td>
<td>Rotate left without using carry bit</td>
</tr>
<tr>
<td>RRNCF</td>
<td>Z, N</td>
<td>Rotate right without using carry bit</td>
</tr>
<tr>
<td>SETF</td>
<td></td>
<td>Set all bits of file</td>
</tr>
<tr>
<td>SUBFWB</td>
<td>C, DC, OV, N</td>
<td>WREG - f with borrow (carry)</td>
</tr>
<tr>
<td>SUBWFB</td>
<td>C, DC, OV, N</td>
<td>f - WREG with borrow (carry)</td>
</tr>
<tr>
<td>TBLRD</td>
<td></td>
<td>Read byte from table in program memory</td>
</tr>
<tr>
<td>TBLWT</td>
<td></td>
<td>Write byte to table in program memory</td>
</tr>
<tr>
<td>TSTFSZ</td>
<td></td>
<td>Test file, skip next instruction if file=0</td>
</tr>
</tbody>
</table>
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.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, S2EVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.
AMERICAS
Corporate Office
2335 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
2335 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:  404-874-0200  Fax:  404-874-0205

Boston
2 Lan Drive, Suite 120
Westford, MA  01886
Tel:  978-845-0330  Fax:  978-845-0335

Chicago
333 Pierce Road, Suite 180
Itasca, IL  60143
Tel:  630-695-0550  Fax:  630-695-0555

Dallas
4670 Westgrove Drive, Suite 160
Addison, TX  75001
Tel:  972-819-0300  Fax:  972-819-0305

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

Kokomo
2107 North First Street, Suite 590
San Jose, CA  95131
Tel:  408-436-7955  Fax:  408-436-7955

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA  95131
Tel:  408-436-7955  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
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 71 Wusi Road
Fuzhou 350001, China
Tel:  86-591-7503506 Fax:  86-591-7503521

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, 189880
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
Laurup hoj 1-3
Ballerpur 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 - 1er Etage
91900 Massy, France
Tel:  33-1-69-30-90-79 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
© 2002 Microchip Technology Inc.