

# Introduction to PIC Programming

## Midrange Architecture and Assembly Language

by David Meiklejohn, Gooligum Electronics

### Lesson 11: Analog Comparators, part 2

[Lesson 9](#) introduced the PIC12F629's single analog comparator module – typical of the comparator modules available in older midrange PIC devices<sup>1</sup>. It showed how a comparator can be used to detect whether an analog signal is above or below a fixed or variable threshold, and to generate an interrupt and/or wake the PIC from sleep when the threshold is crossed. We also saw that the PIC12F629's programmable voltage can be used to generate multiple input thresholds, and that, through input multiplexing, it is possible to use a single comparator to monitor more than one signal.

However, if you need to generate an interrupt, or wake the device from sleep, when a signal crosses one of two thresholds (i.e. goes outside an allowed range), or when either of two independent inputs crosses a threshold, these single-comparator polling and multiplexing techniques are inadequate, and you need two comparators.

This lesson focuses on the PIC16F684's dual comparator module, which is essentially an extended version of the single comparator module described in [lesson 9](#).

In summary, this lesson covers:

- Using two comparators to respond to two inputs
- Using two comparators to measure one signal against two thresholds
- Using two comparators to respond to four independent signals

### PIC16F684 Comparator Module

The PIC16F684 includes a dual comparator module.

It is controlled by the CMCON0 register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CMCON0	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0

As you see, it is very similar to the 12F629's CMCON register, but contains two comparator output bits, C1OUT and C2OUT, and two output inversion bits, C1INV and C2INV.

These bits are exactly analogous to the 12F629's COUT and CINV bits.

---

<sup>1</sup> Newer midrange PICs, such as the 16F690 and 16F887, include a more flexible comparator module, which will be covered in a later lesson.

In normal operation, where C1INV is clear, C1OUT = '1' indicates that the voltage on comparator 1's positive input (C1 VIN+) is higher than that on its negative input (C1 VIN-).

If C1INV is set, the operation is inverted, and C1OUT = '0' if C1 VIN+ > C1 VIN-.

Similarly, if C2INV is clear, C2OUT = '1' indicates that the voltage on comparator 2's positive input (C2 VIN+) is higher than that on its negative input (C2 VIN-).

If C2INV is set, the operation is inverted, and C2OUT = '0' if C2 VIN+ > C2 VIN-.

Like the 12F629, the 16F684 provides a number of "canned" comparator configurations, or operating modes, selected by the CM<2:0> comparator mode bits.

Once again, in some modes, the CIS (comparator input switch) bit selects which pin will be connected to the comparators' negative inputs.

These comparator modes are illustrated graphically in the PIC16F684 data sheet, and are summarised in the following table:

CM<2:0>	CIS	Comparator 1		Comparator 2		Outputs	Description
		VIN-	VIN+	VIN-	VIN+		
000	–	C1IN-	C1IN+	C2IN-	C2IN+	off	Off with inputs connected (default)
001	0	C1IN-	C2IN+	C2IN-	C2IN+	internal	3 inputs with common external reference
001	1	C1IN+	C2IN+	C2IN-	C2IN+	internal	3 inputs with common external reference
010	0	C1IN-	CVREF	C2IN-	CVREF	internal	4 inputs with internal voltage reference
010	1	C1IN+	CVREF	C2IN+	CVREF	internal	4 inputs with internal voltage reference
011	–	C1IN-	C2IN+	C2IN-	C2IN+	internal	2 inputs with common external reference
100	–	C1IN-	C1IN+	C2IN-	C2IN+	internal	Two independent comparators
101	–	(none)	(none)	C2IN-	C2IN+	internal	Comparator 2 only (C1 off, disconnected)
110	–	C1IN-	C2IN+	C2IN-	C2IN+	external	2 inputs with common ref and ext outputs
111	–	(none)	(none)	(none)	(none)	off	Off with inputs disconnected (lowest power)

CVREF is the programmable voltage reference, as described in [lesson 9](#).

From this table, it would seem that, with CIS = 0, mode 001 is equivalent to mode 011, where C1IN- and C2IN- are each compared with a common reference input on C2IN+.

However, in mode 011, C1IN+ is not used, and the pin it shares, RA0, is available as a digital input. But in mode 001, C1IN+ can be selected as an input (by setting CIS), and RA0 is not available as a digital input, even when CIS = 0.

Note that external output, where the two comparator outputs appear on the C1OUT and C2OUT pins, is only available in mode 110. In this mode, pins RA2 and RC4, which are shared with C1OUT and C2OUT, cannot be used as outputs. For the comparator outputs to appear on these pins, the corresponding TRIS bits, TRISA<2> and TRISC<4>, still have to be cleared, as usual.



because they are connected to pins RC0 and RC1, which are used here as comparator inputs C2IN+ and C2IN-. This means that, if you use the LPC demo board, you must remove jumpers JP1 and JP2 (cutting PCB traces if necessary), as well as removing JP5, as we did in [lesson 9](#). You'll also need to connect the LDRs (four this time), as before. All of the necessary signals are available on the 14-pin header on the LPC demo board, but it may be easier to build the whole circuit on a prototyping board.

This example is an extension of the two-input example in [lesson 9](#), where we used the CIS bit to switch one comparator input between two pins. The other comparator input was connected to the internal voltage reference, set to generate a fixed 1.5 V reference threshold.

Since we have two comparators in the 16F684, we can extend this approach, using CIS to switch two comparator inputs (the negative input on each comparator) between four pins, with the internal voltage reference being used for each comparison.

That means using comparator mode 010 (2):

```

; configure comparators
movlw   b'010'|1<<C1INV|1<<C2INV
; select mode 2 (CM = 010):
; C1 +ref is CVref,
; C1 -ref is C1IN+ or C1IN- (selected by CIS),
; C2 +ref is CVref,
; C2 -ref is C2IN+ or C2IN- (selected by CIS),
; no external outputs,
; both comparators on
; C1 output inverted (C1INV = 1)
; -> C1OUT = 1 if C1 -ref > CVref
; C2 output inverted (C2INV = 1)
; -> C2OUT = 1 if C2 -ref > CVref

banksel CMCON0
movwf   CMCON0

```

Alternatively, this could be written as:

```

movlw   b'00110010'   ; configure comparators:
; -----010         select mode 2 (CM = 010):
;                   C1 +ref is CVref,
;                   C1 -ref is C1IN+ or C1IN- (selected by CIS),
;                   C2 +ref is CVref,
;                   C2 -ref is C2IN+ or C2IN- (selected by CIS),
;                   no external outputs,
;                   both comparators on
; ---1-----       C1 output inverted (C1INV = 1)
;                   -> C1OUT = 1 if C1 -ref > CVref
; --1-----       C2 output inverted (C2INV = 1)
;                   -> C2OUT = 1 if C2 -ref > CVref

banksel CMCON0
movwf   CMCON0

```

The comparator output inversion bits, C1INV and C2INV, have been set, because the logic seems clearer if the comparator outputs go high when the comparator inputs are high (corresponding to a high light level), but it would be perfectly valid to leave C1INV and C2INV clear, and to invert the comparator output tests in the main loop, instead. It's up to you!

To use pins as comparator inputs, their associated TRIS bits must be set to '1':

```

movlw   b'000011'   ; configure comparator inputs on PORTC
banksel TRISC       ; (RC0/C2IN+ and RC1/C2IN-) as inputs
movwf   TRISC

```

There is no need to explicitly set the bits in TRISA corresponding to RA0/C1IN+ and RA1/C1IN-, because all TRIS bits are set by default, at power-on, and the output LEDs are all on PORTC.

We could instead have written:

```
movlw    ~(1<<nLED1|1<<nLED2|1<<nLED3|1<<nLED4)    ; configure LED pins
banksel  TRISC                                     ; as outputs
movwf    TRISC
```

This will work, but expressing it this way implies that we can choose different values for nLED1-4, moving the LEDs to different pins on PORTC, but in reality we cannot use RC0 or RC1 as outputs, because those pins are shared with the comparator 2 inputs.

The first way of writing this expresses that restriction more clearly.

The voltage reference is configured as we did in [lesson 9](#):

```
; configure voltage reference
movlw    1<<VRR|.7|1<<VREN
; CVref = 0.292*Vdd (VRR = 1, VR = 7)
; enable voltage reference (VREN = 1)
banksel  VRCON
movwf    VRCON
; -> CVref = 1.5 V (if Vdd = 5.0 V)
```

The tests in the main loop code are much the same as in the example in [lesson 9](#), except that, for each setting of CIS, we now need to test the output of two comparators:

```
; test inputs 1 and 3
banksel  CMCON0          ; select C1IN+ and C2IN+ pins as -refs
bsf      CMCON0,CIS     ; (CIS = 1)
Delay10us          ; wait 10 us to settle
btfsc    CMCON0,C1OUT   ; if C1IN+ > CVref
bsf      sPORTC,nLED1   ; turn on LED 1
btfsc    CMCON0,C2OUT   ; if C2IN+ > CVref
bsf      sPORTC,nLED3   ; turn on LED 3

; test inputs 2 and 4
banksel  CMCON0          ; select C1IN- and C2IN- pins as -refs
bcf      CMCON0,CIS     ; (CIS = 0)
Delay10us          ; wait 10 us to settle
btfsc    CMCON0,C1OUT   ; if C1IN- > CVref
bsf      sPORTC,nLED2   ; turn on LED 2
btfsc    CMCON0,C2OUT   ; if C2IN- > CVref
bsf      sPORTC,nLED4   ; turn on LED 4
```

Note that, as in the 12F629, we need to wait a little while for the comparator inputs to settle, after changing the input selection. According to the 16F684 data sheet, this can be up to 10  $\mu$ s, so we use a 10  $\mu$ s delay after changing the value of CIS.

### Complete program

The structure of this “4-input” example is very similar to the “2-input” example presented in [lesson 9](#). But, here is the full listing, for reference:

```
;*****
;
; Description: Lesson 11, example 1
;
; Demonstrates dual comparator input multiplexing
;*****
```

```

; Turns on: LED 1 when C1IN+ > 1.5 V, *
;           LED 2 when C1IN- > 1.5 V, *
;           LED 3 when C2IN+ > 1.5 V, *
;           and LED 4 when C2IN- > 1.5 V *
; *
;*****
;
; Pin assignments: *
; C1IN+ - input 1 (LDR/resistor divider) *
; C1IN- - input 2 (LDR/resistor divider) *
; C2IN+ - input 3 (LDR/resistor divider) *
; C2IN- - input 4 (LDR/resistor divider) *
; RC2   - indicator LED 1 *
; RC3   - indicator LED 2 *
; RC4   - indicator LED 3 *
; RC5   - indicator LED 4 *
; *
;*****

list      p=16F684
#include  <p16F684.inc>

errorlevel -302          ; no warnings about registers not in bank 0

radix    dec

;***** CONFIGURATION
; ext reset, no code or data protect, no brownout detect,
; no watchdog, power-up timer, int clock with I/O,
; no failsafe clock monitor, two-speed start-up disabled
__CONFIG    _MCLRE_ON & _CP_OFF & _CPD_OFF & _BOD_OFF & _WDT_OFF & _PWRTE_ON
& _INTOSCIO & _FCMEN_OFF & _IESO_OFF

; pin assignments
constant    nLED1=2          ; indicator LED 1 on RC2
constant    nLED2=3          ; indicator LED 2 on RC3
constant    nLED3=4          ; indicator LED 3 on RC4
constant    nLED4=5          ; indicator LED 4 on RC5

;***** MACROS

; 10 us delay
; (assuming 4 MHz processor clock)
Delay10us   MACRO
            goto $+1          ; 2 us delay * 5 = 10 us
            goto $+1
            goto $+1
            goto $+1
            goto $+1
            ENDM

;***** VARIABLE DEFINITIONS
            UDATA_SHR
sPORTC     res 1              ; shadow copy of PORTC

;*****
RESET     CODE    0x0000      ; processor reset vector

```

```

;***** MAIN PROGRAM

;***** Initialisation
; configure ports
movlw    b'000011'          ; configure comparator inputs on PORTC
banksel  TRISC              ; (RC0/C2IN+ and RC1/C2IN-) as inputs
movwf    TRISC
; configure comparators
movlw    b'010'|1<<C1INV|1<<C2INV
; select mode 2 (CM = 010):
; C1 +ref is CVref,
; C1 -ref is C1IN+ or C1IN- (select by CIS),
; C2 +ref is CVref,
; C2 -ref is C2IN+ or C2IN- (select by CIS),
; no external outputs,
; both comparators on
; C1 output inverted (C1INV = 1)
; -> C1OUT = 1 if C1 -ref > CVref
; C2 output inverted (C2INV = 1)
; -> C2OUT = 1 if C2 -ref > CVref
banksel  CMCON0
movwf    CMCON0
; configure voltage reference
movlw    1<<VRR|.7|1<<VREN
; CVref = 0.292*Vdd (VRR = 1, VR = 7)
; enable voltage reference (VREN = 1)
; -> CVref = 1.5 V (if Vdd = 5.0 V)
banksel  VRCON
movwf    VRCON

;***** Main loop
mainloop
    clrf    sPORTC          ; start with all LEDs off

    ; test inputs 1 and 3
    banksel CMCON0          ; select C1IN+ and C2IN+ pins as -refs
    bsf    CMCON0,CIS      ; (CIS = 1)
    Delay10us              ; wait 10 us to settle
    btfsc  CMCON0,C1OUT    ; if C1IN+ > CVref
    bsf    sPORTC,nLED1    ; turn on LED 1
    btfsc  CMCON0,C2OUT    ; if C2IN+ > CVref
    bsf    sPORTC,nLED3    ; turn on LED 3

    ; test inputs 2 and 4
    banksel CMCON0          ; select C1IN- and C2IN- pins as -refs
    bcf    CMCON0,CIS      ; (CIS = 0)
    Delay10us              ; wait 10 us to settle
    btfsc  CMCON0,C1OUT    ; if C1IN- > CVref
    bsf    sPORTC,nLED2    ; turn on LED 2
    btfsc  CMCON0,C2OUT    ; if C2IN- > CVref
    bsf    sPORTC,nLED4    ; turn on LED 4

    ; display test results
    movf    sPORTC,w        ; copy shadow to PORTC
    banksel PORTC
    movwf   PORTC

    ; repeat forever
    goto   mainloop

END

```

### Using interrupts with two comparators

Each comparator in the dual comparator module can be used, independently, as an interrupt source, in the same way as the single comparator module described in [lesson 9](#).

The only difference is that, instead of having a single comparator interrupt enable bit and comparator interrupt flag, there is two of each – one for each comparator.

Each comparator interrupt is enabled by setting the corresponding bit (C1IE and C2IE) in the PIE1 register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PIE1	EEIE	ADIE	CCP1IE	C2IE	C1IE	OSFIE	TMR2IE	TMR1IE

Once again, to enable any peripheral interrupt, you must also set the PEIE and GIE bits in INTCON:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE	PEIE	T0IE	INTE	RAIE	T0IF	INTF	RAIF

So, to enable comparator interrupts, you must set C1IE and/or C2IE, plus PEIE and GIE.

Each comparator has a corresponding interrupt flag (C1IF and C2IF) in the PIR1 register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PIR1	EEIF	ADIF	CCP1IF	C2IF	C1IF	OSFIF	TMR2IF	TMR1IF

The comparator interrupt flags are set whenever the corresponding comparator output (C1OUT or C2OUT) has changed since the last time CMCON0 was read or written.

To demonstrate this, we'll extend the comparator interrupt example from [lesson 9](#) to two comparators.

That example used a circuit where the comparator output was fed back to its input, generating hysteresis through positive feedback. This was done to make the comparator switch more cleanly, minimising the number of transitions, and hence the number of interrupts.

So we'll do the same here, feeding each comparator's output back to one of its inputs.

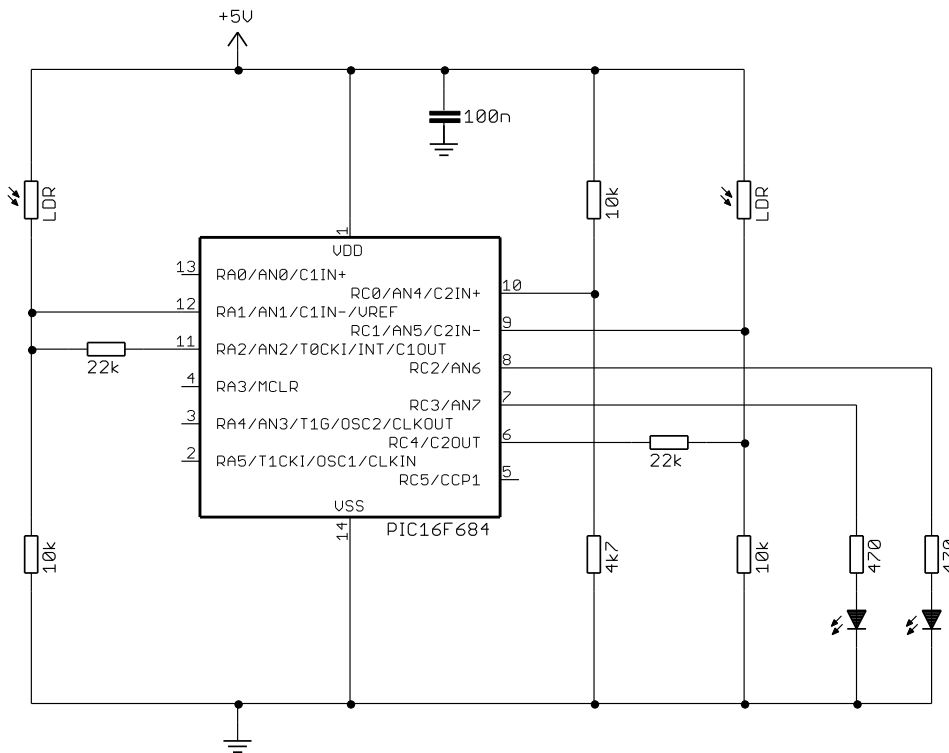
Since the external comparator outputs are only enabled in comparator mode 110, we have to use mode 110.

In this mode, C2IN+ is used as a common external reference (the internal voltage reference cannot be used, because there is no comparator mode where both the internal reference and external outputs are enabled), so we have to use an external resistor voltage divider to generate the threshold voltage.

We can't feed the comparator outputs back to this common reference, because then the operation of one comparator would affect the other.

So we need to feed C1OUT back into C1IN-, and C2OUT back into C2IN-. The circuit diagram on the next page shows how this is done.

These pins are connected as negative comparator inputs in mode 110, so it is necessary to invert the comparator outputs, by setting the C1INV and C2INV bits, to produce the overall positive feedback required to generate hysteresis.



The 10 k and 4.7 k resistors connected to C2IN+ form a voltage divider, creating a reference voltage of close to 1.6 V from the 5 V supply.

In the example in [lesson 9](#), the LED was used to indicate low light, so we'll do the same here.

The LED on RC2 will be turned on when the voltage at C1IN- is below 1.6 V, and the LED on RC3 will indicate a low voltage (light level) on C2IN-.

Comparator mode 6 (110) is selected, and the outputs inverted to create positive feedback, by:

```
movlw    b'110'|1<<C1INV|1<<C2INV
; select mode 6 (CM = 110):
; C1 -ref is C1IN-,
; C1 +ref is C2IN+
; C2 -ref is C2IN-,
; C2 +ref is C2IN+,
; both external outputs enabled,
; both comparators on
; C1 output inverted (C1INV = 1)
; -> C1OUT = 1 if C1IN- > C2IN+
; C2 output inverted (C2INV = 1)
; -> C2OUT = 1 if C2IN- > C2IN+

banksel  CMCON0
movwf   CMCON0
```

To allow the comparators to control the output pins, we must clear the corresponding TRIS bits:

```
banksel  TRISA
bcf      TRISA,2
bcf      TRISC,4
; enable comparator external outputs:
; C1OUT (RA2)
; C2OUT (RC4)
```

The interrupts are configured in the much same way as the single comparator interrupt in [lesson 9](#), but this time there are two comparators:

```
movlw    1<<PEIE|1<<GIE
movwf   INTCON
banksel  CMCON0
movf    CMCON0,w
; enable peripheral and global interrupts
; enable comparator interrupts:
; read CMCON0 to clear mismatch
```

```

banksel PIR1
bcf     PIR1,C1IF      ; clear interrupt flags
bcf     PIR1,C2IF
banksel PIE1
bsf     PIE1,C1IE      ; set enable bits
bsf     PIE1,C2IE

```

At the start of the interrupt service routine (just after saving the processor context), we need to determine which source triggered this interrupt. Since only the comparator interrupts are enabled in this example, we only need to check the comparator interrupt flags, in the PIR1 register, to see which comparator was the interrupt source:

```

; *** Identify interrupt source
banksel PIR1
btfsc  PIR1,C1IF      ; comparator 1
goto   c1_int
btfsc  PIR1,C2IF      ; comparator 2
goto   c2_int
goto   isr_end        ; none of the above, so exit ISR

```

The code which then handles each comparator interrupt is very similar to the comparator interrupt handler from [lesson 9](#):

```

c1_int ; *** Service comparator 1 interrupt
; Triggered on any comparator 1 output change,
; caused by C1IN- input crossing C2IN+ (1.6 V) threshold
;
banksel CMCON0
movf   CMCON0,w      ; clear mismatch condition
banksel PIR1
bcf    PIR1,C1IF      ; clear interrupt flag
; turn on LED 1 if C1IN- < C2IN+
bcf    sPORTC,nLED1   ; assume C1OUT = 1 -> LED 1 off
banksel CMCON0
btfss  CMCON0,C1OUT   ; if comparator output low (C1IN- < C2IN+)
bsf    sPORTC,nLED1   ; turn on LED 1 (using shadow register)
; exit ISR
goto   isr_end

c2_int ; *** Service comparator 2 interrupt
; Triggered on any comparator 2 output change,
; caused by C2IN- input crossing C2IN+ (1.6 V) threshold
;
banksel CMCON0
movf   CMCON0,w      ; clear mismatch condition
banksel PIR1
bcf    PIR1,C2IF      ; clear interrupt flag
; turn on LED 2 if C2IN- < C2IN+
bcf    sPORTC,nLED2   ; assume C2OUT = 1 -> LED 2 off
banksel CMCON0
btfss  CMCON0,C2OUT   ; if comparator output low (C2IN- < C2IN+)
bsf    sPORTC,nLED2   ; turn on LED 2 (using shadow register)
; exit ISR
goto   isr_end

```

**Complete program**

Again, although structure of this “dual-comparator interrupt” example is the same as the “single-comparator interrupt” example from [lesson 9](#), here is the full listing, to show how the pieces fit together:

```

;*****
;
; Description: Lesson 11, example 2
;
; Demonstrates use of interrupts with two comparators
; (assumes hysteresis is used to reduce triggering)
;
; Turns on: LED 1 when C1IN- > C2IN+,
; and LED 2 when C2IN- > C2IN+
;
;*****
;
; Pin assignments:
; C1IN- - input 1 (LDR/resistor divider)
; C2IN- - input 2 (LDR/resistor divider)
; C2IN+ - common reference (LDR/resistor divider)
; RC2 - indicator LED 1
; RC3 - indicator LED 2
;
;*****

list p=16F684
#include <p16F684.inc>

errorlevel -302 ; no warnings about registers not in bank 0
errorlevel -312 ; no "page or bank selection not needed" messages

radix dec

;***** CONFIGURATION
; ext reset, no code or data protect, no brownout detect,
; no watchdog, power-up timer, int clock with I/O,
; no failsafe clock monitor, two-speed start-up disabled
__CONFIG _MCLRE_ON & _CP_OFF & _CPD_OFF & _BOD_OFF & _WDT_OFF & _PWRTE_ON
& _INTOSCIO & _FCMEN_OFF & _IESO_OFF

; pin assignments
constant nLED1=2 ; indicator LED 1 on RC2
constant nLED2=3 ; indicator LED 2 on RC3

;***** VARIABLE DEFINITIONS
CONTEXT UDATA_SHR ; context saving
cs_W res 1
cs_STATUS res 1

GENVAR UDATA_SHR ; general variables
sPORTC res 1 ; shadow copy of PORTC

;*****
RESET CODE 0x0000 ; processor reset vector
pagesel Start
goto Start

```

```

;***** INTERRUPT SERVICE ROUTINE
ISR      CODE      0x0004
        ; *** Save context
        movwf     cs_W                ; save W
        movf      STATUS,w           ; save STATUS
        movwf     cs_STATUS

        ; *** Identify interrupt source
        banksel   PIR1
        btfsc    PIR1,C1IF           ; comparator 1
        goto     c1_int
        btfsc    PIR1,C2IF           ; comparator 2
        goto     c2_int
        goto     isr_end              ; none of the above, so exit ISR

c1_int   ; *** Service comparator 1 interrupt
        ; Triggered on any comparator 1 output change,
        ; caused by C1IN- input crossing C2IN+ (1.6 V) threshold
        ;
        banksel   CMCON0
        movf      CMCON0,w           ; clear mismatch condition
        banksel   PIR1
        bcf       PIR1,C1IF           ; clear interrupt flag
        ; turn on LED 1 if C1IN- < C2IN+
        bcf       sPORTC,nLED1       ; assume C1OUT = 1 -> LED 1 off
        banksel   CMCON0
        btfss    CMCON0,C1OUT        ; if comparator output low (C1IN- < C2IN+)
        bsf       sPORTC,nLED1       ; turn on LED 1 (using shadow register)
        ; exit ISR
        goto     isr_end

c2_int   ; *** Service comparator 2 interrupt
        ; Triggered on any comparator 2 output change,
        ; caused by C2IN- input crossing C2IN+ (1.6 V) threshold
        ;
        banksel   CMCON0
        movf      CMCON0,w           ; clear mismatch condition
        banksel   PIR1
        bcf       PIR1,C2IF           ; clear interrupt flag
        ; turn on LED 2 if C2IN- < C2IN+
        bcf       sPORTC,nLED2       ; assume C2OUT = 1 -> LED 2 off
        banksel   CMCON0
        btfss    CMCON0,C2OUT        ; if comparator output low (C2IN- < C2IN+)
        bsf       sPORTC,nLED2       ; turn on LED 2 (using shadow register)
        ; exit ISR
        goto     isr_end

isr_end  ; *** Restore context then return
        movf      cs_STATUS,w        ; restore STATUS
        movwf     STATUS
        swapf     cs_W,f             ; restore W
        swapf     cs_W,w
        retfie

;***** MAIN PROGRAM
MAIN     CODE

;***** Initialisation
Start    ; configure ports
        movlw    b'000011'          ; configure comparator input pins on PORTC

```

```

banksel TRISC          ; (RC0/C2IN+ and RC1/C2IN-) as inputs
movwf  TRISC
; configure comparators
movlw  b'110'|1<<C1INV|1<<C2INV
; select mode 6 (CM = 110):
; C1 -ref is C1IN-,
; C1 +ref is C2IN+
; C2 -ref is C2IN-,
; C2 +ref is C2IN+,
; both external outputs enabled,
; both comparators on
; C1 output inverted (C1INV = 1)
; -> C1OUT = 1 if C1IN- > C2IN+
; C2 output inverted (C2INV = 1)
; -> C2OUT = 1 if C2IN- > C2IN+

banksel CMCON0
movwf  CMCON0
banksel TRISA          ; enable comparator external outputs:
bcf    TRISA,2         ; C1OUT (RA2)
bcf    TRISC,4         ; C2OUT (RC4)
; configure interrupts
movlw  1<<PEIE|1<<GIE ; enable peripheral and global interrupts
movwf  INTCON
banksel CMCON0         ; enable comparator interrupts:
movf   CMCON0,w       ; read CMCON0 to clear mismatch
banksel PIR1
bcf    PIR1,C1IF      ; clear interrupt flags
bcf    PIR1,C2IF
banksel PIE1
bsf    PIE1,C1IE      ; set enable bits
bsf    PIE1,C2IE

;***** Main loop
mainloop
; continually copy shadow to PORTC
movf   sPORTC,w
banksel PORTC
movwf  PORTC

; repeat forever
goto  mainloop

END

```

### **Wake-up when signal goes outside limits**

We saw in [lesson 9](#) that a comparator can be used to wake the PIC from sleep mode, whenever the comparator's output changes.

Assuming that one of the comparator's inputs is a fixed reference, defining a threshold level, the comparator's output will change, waking the PIC, whenever the second input crosses this threshold.

Thus, the comparator can be used as an "alarm", waking the PIC whenever a particular input becomes too high or too low. But a single comparator can only test for either "too high" or "too low" – not both. What if you had a signal representing a condition (say, temperature), which had to be maintained within a certain range? We've seen that you can do that by combining a single comparator with a programmable voltage reference, and regularly switching the reference between two levels, and polling the comparator output. But you can't



The output inversion bits have been configured so that each output bit is a '1' only when an error condition (input too low or high) exists.

Recall that, to configure an interrupt source to wake the device from sleep, the corresponding interrupt must be enabled:

```

; enable comparator interrupts (for wake on change)
bsf    INTCON,PEIE          ; enable peripheral interrupts
banksel PIE1
bsf    PIE1,C1IE           ; and comparator interrupts
bsf    PIE1,C2IE

```

Note that, because the comparator is a peripheral, the peripheral interrupt enable bit, PEIE, must be set, in addition to the two comparator interrupt enable bits. Of course, if you were only using one of the comparators for wake-up, you would only set that comparator's interrupt enable bit.

And, because we're not actually using interrupts in this example, we leave the global interrupt enable bit, GIE, clear.

In the example in [lesson 9](#), the LED was turned on for one second, whenever the comparator output changed (in either direction). But that doesn't seem appropriate when we're using two LEDs, to indicate when the input is too high or too low.

So, in this example, the "Low" LED stays on as long as the input is lower than the low threshold (defined by the voltage on C1IN-):

```

ck_lo   ; test for and wait while input low
banksel CMCON0
btfss  CMCON0,C1OUT          ; if C2IN+ < C1IN- (1.1 V),
goto   ck_hi
banksel PORTC                ; turn on "low" LED
bsf    PORTC,nLO
goto   ck_lo                 ; continue to poll while low

```

Similarly, the "High" LED stays on as long as the input is higher than the high threshold (defined by the voltage on C2IN-):

```

ck_hi   ; test for and wait while input high
banksel CMCON0
btfss  CMCON0,C2OUT          ; if C2IN+ > C2IN- (2.0 V),
goto   ck_end
banksel PORTC                ; turn on "high" LED
bsf    PORTC,nHI
goto   ck_hi                 ; continue to poll while high
ck_end

```

After getting through these two polling loops, we can be sure that the input on C2IN+ is neither too low nor too high, so we can turn off the LEDs (which may have been turned on in the polling loops):

```

; turn off LEDs
banksel PORTC
clrf   PORTC

```

And since the input is now within the allowed range, we can sleep until it crosses the low or high threshold again.

In practice, this isn't a great strategy, because it is likely that, when the input crosses a threshold, it is likely to cross many times, in quick succession, and it would be better if we only enter sleep mode when we know the input is reasonably stable.

To avoid this, you could introduce hysteresis into the circuit, as was done in the interrupt example above (but this time feeding the comparator outputs back to the voltage dividers, not the common input signal), or you could write code similar to that developed for switch debouncing (a similar problem) in [lesson 3](#).

On the other hand, it doesn't really matter if the PIC rapidly goes in and out of sleep mode, while an input threshold is crossed; it doesn't damage the hardware or anything. So we'll keep it simple here.

To prepare to enter sleep mode, we must clear any existing mismatch condition by reading the **CMCON0** register, and clear the comparator interrupt flags, so that the PIC doesn't wake immediately on entering sleep mode:

```

; enter sleep mode
banksel CMCON0           ; read CMCON0 to clear comparator mismatch
movf    CMCON0,w
banksel PIR1
bcf     PIR1,C1IF        ; clear comparator interrupt flags
bcf     PIR1,C2IF

sleep

```

When the PIC comes out of sleep mode, one of the comparator outputs must have changed, meaning that the input has crossed one of the thresholds, so we loop back and test the comparator outputs again:

```

; repeat forever
goto   mainloop

```

### ***Complete program***

Here's how it all fits together:

```

;*****
; Description:      Lesson 10, example 3                               *
;                                                         *
; Demonstrates wake-up on dual comparator change             *
;                                                         *
; Device wakes when input crosses a low or high threshold   *
;                                                         *
; Turns on Low LED when C2IN+ < C1IN- (1.1 V)              *
; or High LED when C2IN+ > C2IN- (2.0 V)                   *
; then sleeps until the next change                          *
;                                                         *
; (thresholds are set by external voltage dividers)         *
;                                                         *
;*****
;
; Pin assignments:                                          *
; C2IN+ - voltage to be measured (e.g. pot output or LDR)  *
; C1IN- - low threshold (1.1 V)                             *
; C2IN- - high threshold (2.0 V)                            *
; RC3   - "Low" LED                                         *
; RC2   - "High" LED                                        *
;*****

```

```

list      p=16F684
#include   <p16F684.inc>

errorlevel -302      ; no warnings about registers not in bank 0
errorlevel -312      ; no "page or bank selection not needed" messages

radix     dec

;***** CONFIGURATION
                ; ext reset, no code or data protect, no brownout detect,
                ; no watchdog, power-up timer, int clock with I/O,
                ; no failsafe clock monitor, two-speed start-up disabled
__CONFIG     _MCLRE_ON & _CP_OFF & _CPD_OFF & _BOD_OFF & _WDT_OFF & _PWRTE_ON
& _INTOSCIO & _FCMEN_OFF & _IESO_OFF

; pin assignments
constant     nLO=3          ; "Low" LED on RC3
constant     nHI=2          ; "High" LED on RC2

;*****
RESET        CODE    0x0000          ; processor reset vector

;***** MAIN PROGRAM

;***** Initialisation
                ; configure ports
movlw        ~(1<<nLO|1<<nHI)        ; configure LED pins (only) as outputs
banksel     TRISC
movwf       TRISC
banksel     PORTC          ; start with both LEDs off
clrf       PORTC
                ; configure comparators
movlw        b'011'|1<<C1INV|0<<C2INV
                                ; select mode 3 (CM = 011):
                                ; C1 -ref is C1IN-,
                                ; C1 +ref is C2IN+,
                                ; C2 -ref is C2IN-,
                                ; C2 +ref is C2IN+,
                                ; no external outputs,
                                ; both comparators on
                                ; C1 output inverted (C1INV = 1)
                                ; -> C1OUT = 1 if C2IN+ < C1IN-
                                ; C2 output not inverted (C2INV = 0)
                                ; -> C2OUT = 1 if C2IN+ > C2IN-
banksel     CMCON0
movwf       CMCON0
                ; enable comparator interrupts (for wake on change)
bsf         INTCON,PEIE          ; enable peripheral interrupts
banksel     PIE1
bsf         PIE1,C1IE            ; and comparator interrupts
bsf         PIE1,C2IE

;***** Main loop
mainloop

ck_lo      ; test for and wait while input low
banksel     CMCON0
btfss      CMCON0,C1OUT          ; if C2IN+ < C1IN- (1.1 V),
goto       ck_hi

```

```

        banksel PORTC                ; turn on "low" LED
        bsf      PORTC,nLO
        goto     ck_lo                ; continue to poll while low

ck_hi   ; test for and wait while input high
        banksel CMCON0
        btfss   CMCON0,C2OUT         ; if C2IN+ > C2IN- (2.0 V),
        goto     ck_end
        banksel PORTC                ; turn on "high" LED
        bsf      PORTC,nHI
        goto     ck_hi                ; continue to poll while high

ck_end

        ; turn off LEDs
        banksel PORTC
        clrf    PORTC

        ; enter sleep mode
        banksel CMCON0                ; read CMCON0 to clear comparator mismatch
        movf    CMCON0,w
        banksel PIR1
        bcf     PIR1,C1IF             ; clear comparator interrupt flags
        bcf     PIR1,C2IF

        sleep

        ; repeat forever
        goto    mainloop

END

```

The 16F684's dual comparator module also provides some facilities which are used with Timer1, which will be described when we look at Timer1 in a future lesson.

We've been doing a lot by flashing LEDs, but to go much further we'll need to be able to display output in numerical form, so the [next lesson](#) introduces 7-segment displays, allowing us to explore the topics of look-up tables and display multiplexing.