Introduction to PIC Programming

Midrange Architecture and Assembly Language

by David Meiklejohn, Gooligum Electronics

Lesson 9: Analog Comparators, part 1

We've seen how to respond to simple on/off digital signals, but we live in an "analog" world; many of the sensors you will want your PIC-based projects to respond to, such as temperature or light, have smoothly varying outputs whose magnitude represents the value being measured; they are *analog* outputs.

In many cases, you will want to treat the output of an analog sensor as though it was digital (i.e. on or off, high or low), without worrying about the exact value. It may be that a pulse, that does not meet the requirements for a "digital" input, has to be detected: for example, the output of a Hall-effect sensor attached to a rotating part. Or we may simply wish to respond to a threshold (perhaps temperature) being crossed.

In such cases, where we only need to respond to an input voltage being higher or lower than a threshold, it is usually appropriate to use an analog *comparator*. In fact, analog comparators are so useful that most PICs include one or two of them, as built-in peripherals.

This lesson revisits material from baseline lesson 9, explaining how to use the type of single comparator module found on older midrange PIC devices[1] to respond to analog inputs.

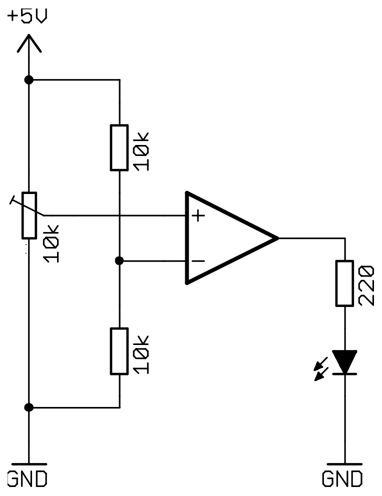
In summary, this lesson covers:

- Using comparators to compare voltage levels
- Adding comparator hysteresis
- Comparator-driven interrupts
- Wake-up on comparator change
- Using the programmable voltage reference
- Using a single comparator to:
    - Compare a signal against a range of input thresholds
    - Compare two independent inputs against a common reference

Comparators

A *comparator* (technically, an analog comparator, since comparators with digital inputs also exist) is a device which compares the voltages present on its positive and negative inputs. In normal (non-inverted) operation, if the voltage on the positive input is greater than that on the negative input, the output is set "high"; otherwise the output is "low".

[1] The older-style dual comparator module will be covered in the next lesson. Newer midrange PICs, such as the 16F690 and 16F887, include a more flexible type of comparator module, which will be covered in a later lesson.



An example is shown in the circuit diagram on the right.

The two 10 kΩ resistors act as a voltage divider, presenting 2.5 V at the comparator's negative input. This sets the on/off threshold voltage.

The potentiometer can be adjusted to set the positive input to any voltage between 0 V and 5 V.

When the potentiometer is set to a voltage under 2.5 V, the comparator's output will be low and the LED will not be lit. But when the potentiometer is turned up past halfway, the comparator's output will go high, lighting the LED.

Comparators are typically used when a circuit needs to react to a sensor's analog output being above or below some threshold, triggering some event (e.g. time to fill the tank, turn off a heater, or start an alarm).

They are also useful for level conversion. Suppose a sensor is outputting pulses which are logically "on/off" (i.e. the output is essentially digital), but do not match the voltage levels needed by the digital devices they are driving. For example, the digital inputs of a PIC, with VDD = 5 V (i.e. TTL-compatible), require at least 2.0 V to be register as "high". That's a problem if a sensor is delivering a stream of 0 - 1 V pulses. By passing this signal through a comparator with an input threshold of 0.5 V, the 1 V pulses would be recognised as being "high"; the output of the comparator being a compliant TTL digital signal.

Similarly, comparators can be used to *shape* or *condition* poorly defined or slowly-changing signals. The logic level of a signal between 0.8 V and 2.0 V is not defined for digital inputs on a PIC with VDD = 5 V. And excessive current can flow when a digital input is at an intermediate value. Input signals which spend any significant amount of time in this intermediate range should be avoided. Such input signals can be cleaned up by passing them through a comparator; the output will have sharply-defined transitions between valid digital voltage levels.

Through the use of built-in comparators, PICs can directly detect analog signal threshold crossings, or can work with "on/off" signals that do not meet the requirements for digital inputs, avoiding the need for external comparators.

PIC12F629 Comparator Module

The PIC12F629 includes a single comparator.

It is controlled by the CMCON register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CMCON	–	COUT	–	CINV	CIS	CM2	CM1	CM0

The comparator's output appears as the COUT bit: it is set to '1' only when the voltage on the comparator's positive input, or *reference*, is higher than that on its negative input.

That's the normal situation, but the operation of the comparator can be inverted by setting the CINV output inversion bit[2]:

CINV = 0 selects normal operation

CINV = 1 selects inverted operation, where COUT = 1 only if positive input < negative input.

[2] CINV is analogous to the comparator polarity bits described in baseline lesson 9.

Unlike the comparator modules in the baseline architecture, there are no separate control bits for input selection and output pin configuration, or enabling/disabling the comparators on the 12F629.

Instead, a number of "canned" configurations, or operating modes, selected by the CM<2:0> comparator mode bits, are available.

In a couple of the comparator modes, the CIS (comparator input switch) bit selects which pin will be connected to the comparator's negative input.

These modes are illustrated graphically in the PIC12F629 data sheet, and are summarised below:

CM<2:0>	CIS	- ref	+ ref	Output	Description
000	–	CIN-	CIN+	off	Off with inputs connected (default)
001	–	CIN-	CIN+	external	Both inputs, external output
010	–	CIN-	CIN+	internal	Both inputs, internal only
011	–	CIN-	CVREF	external	Negative input vs. reference, external output
100	–	CIN-	CVREF	internal	Negative input vs. reference, internal only
101	0	CIN-	CVREF	external	Negative input vs. reference, external output
101	1	CIN+	CVREF	external	Positive input vs. reference, external output
110	0	CIN-	CVREF	internal	Negative input vs. reference, internal only
110	1	CIN+	CVREF	internal	Positive input vs. reference, internal only
111	–	(none)	(none)	off	Off with inputs disconnected (lowest power)

CVREF is the programmable voltage reference[3], described later.

Note that, although mode 011 appears to be equivalent to mode 101 when CIS = 0, the difference is that GP0 is available as a digital input in mode 011, and not in mode 101.

"External" output means that the comparator's output appears on the COUT pin, which is shared with GP2. When a comparator mode with external output is selected, COUT will override whatever value you have written to GP2, but for the output to appear on the COUT pin, TRISIO<2> still has to be cleared, as usual.

Whether the external output is enabled or not, the comparator's output always appears internally as the COUT bit – unless the comparator is turned off (mode 000 or 111), in which case COUT reads as '0' (assuming CINV = 0).

So why have two "off" modes?

The PIC's digital input buffers are designed to operate with signals which are either "low" or "high", as defined by the "DC Characteristics" section of the data sheet. If they are presented with an in-between level, they may draw excessive current.

The PIC cannot "know", when powered on, whether the comparator is going to be used or not. If the comparator is in use, we can expect that sometimes the analog input signals will be outside the "digital" spec, and may cause damage if they are connected to a digital input. Therefore, the safe course of action is for the PIC to start, at power-on, with the digital inputs disconnected from any pin which may be used with

[3] There is no 0.6 V absolute voltage reference, equivalent to that described in baseline lesson 9, available on the PIC12F629.

an analog signal – such as the comparator inputs. As we will see in lesson 13, this also applies to ADC inputs. By default, any pins with an analog function will start in analog mode, with the digital input buffer disconnected (the digital input will read as '0').

Note: On PICs with comparators and/or analog inputs, the comparator and analog inputs are enabled on start-up. To use a pin for as a digital input, any comparator or analog input assigned to that pin must first be disabled.

This is why the default comparator mode has the comparator turned off (saving power), but with both analog inputs connected to the comparator, and the digital inputs on those pins (GP0 and GP1) disabled.

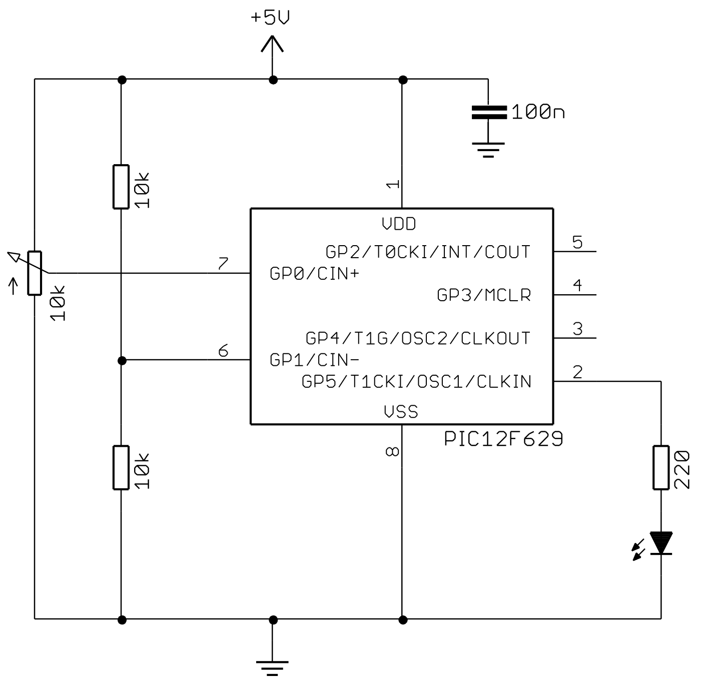
If you are using GP0 and GP1 as digital inputs, or wish to minimise power consumption (and you are sure that there will be no analog signal levels present on CIN- or CIN+), you should select mode '111', or, in decimal, '7'.

This why, in C programs for devices such as the 12F629, you will often see a line such as:

```
CMCON = 7;    // disable comparator
```

But in this lesson, of course, we don't want to disable the comparator; we want to use it!

Basic comparator operation



We will use the circuit on the right to illustrate the comparator's basic operation.

It can be readily built using the Microchip Low Pin Count Demo Board we've been using in this tutorial series.

The 10 kΩ potentiometer on the demo board is already connected to CIN+ (labelled 'RA0') on the board, via a 1 kΩ resistor (not shown in this diagram), assuming that jumper JP5 is in place.

The LED labelled 'DS4' on the demo board can be connected to GP5 (via a 470 Ω resistor, instead of 220 Ω as shown here, but that is not important) by adding a link between pins 1 and 6 on the 14-pin header.

The connection to CIN- (labelled 'RA1' on the board) is available via pin 8 on the 14-pin header. +5V and GND are brought out to pins 13 and 14, making it easy to add the 10 kΩ resistors, forming a voltage divider, by using a solderless breadboard connected to the 14-pin header.

It is straightforward to configure the PIC as a simple comparator, turning on the LED if the potentiometer is turned more than halfway toward the +5V supply side.

First configure GP5 as an output:

```
        movlw   ~(1<<nLED)      ; configure LED pin (only) as an output
        banksel TRISIO
        movwf   TRISIO
```

When configuring the comparator, we should choose mode '010'(mode 2), where CIN+ and CIN- are connected to the comparator inputs, the comparator is enabled, and the output is internal only (not connected to the COUT pin). And since we want to operate the comparator normally (testing for CIN+ > CIN-), the output should not be inverted.

This can be done by:

```
        movlw   0<<CINV|b'010'
                                ; select mode 2 (CM = 010):
                                ;   +ref is CIN+, -ref is CIN-,
                                ;   comparator on, no external output
                                ; output not inverted (CINV = 0)
        banksel CMCON           ;  -> COUT = 1 if CIN+ > CIN-
        movwf   CMCON
```

To turn on the LED when the comparator output is high, we need to repeatedly test COUT.

This could be done by:

```
        banksel CMCON
loop    btfsc   CMCON,COUT      ; if comparator output high
        bsf     GPIO,nLED       ;   turn on LED
        btfss   CMCON,COUT      ; if comparator output low
        bcf     GPIO,nLED       ;   turn off LED

        goto    loop            ; repeat forever
```

We can get away with this code on the PIC12F629, because the GPIO and CMCON registers happen to be in the same bank. However, if you port this code to a device such as the PIC16F690, where the port registers appear in two banks, but the comparator control registers appear in only one of them, your code may fail if you forget the bank layout differences.

It would be safer to use a '`banksel GPIO`' before each GPIO access, but simply inserting those directives into this code won't work, because '`btfsc`' and '`btfss`' only skip a single instruction. This problem can be avoided by testing once and using '`goto`'s:

```
loop    banksel CMCON           ; test comparator output
        btfss   CMCON,COUT
        goto    cm_low
        banksel GPIO            ; if high,
        bsf     GPIO,nLED       ;   turn on LED
        goto    cm_end
cm_low  banksel GPIO            ; else low, so
        bcf     GPIO,nLED       ;   turn off LED
cm_end
        goto    loop            ; repeat forever
```

This can be done more cleanly if using an unbanked shadow port register:

```
loop    clrf    sGPIO           ; assume COUT = 0 -> LED off
        banksel CMCON
        btfsc   CMCON,COUT      ; if comparator output high
        bsf     sGPIO,nLED      ;   turn on LED

        movf    sGPIO,w         ; copy shadow to GPIO
        banksel GPIO
        movwf   GPIO

        goto    loop            ; repeat forever
```

The first form could be restructured the same way (turn off the LED, test the comparator and turn on the LED only if the comparator output is high), but it means that the LED will flicker. That's ok if the output actually is an LED, since the flickering will be much too fast to see, but it won't be ok for many other types of output. Using a shadow register means that the output is only updated once, with the correct value each time.

Complete program

Here is how these code fragments fit within the complete "basic comparator operation" example program:

```
;************************************************************************
;   Description:    Lesson 9, example 1a                                *
;                                                                       *
;   Demonstrates basic comparator operation                             *
;                                                                       *
;   Turns on LED when voltage on CIN+ > voltage on CIN-                 *
;                                                                       *
;************************************************************************
;   Pin assignments:                                                    *
;       CIN+  - voltage to be measured (e.g. pot output or LDR)         *
;       CIN-  - threshold voltage (set by voltage divider resistors)    *
;       GP5   - indicator LED                                           *
;                                                                       *
;************************************************************************
    list        p=12F629
    #include    <p12F629.inc>

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

    radix       dec


;***** CONFIGURATION
                ; int reset, no code or data protect, no brownout detect,
                ; no watchdog, power-up timer, 4Mhz int clock
    __CONFIG    _MCLRE_OFF & _CP_OFF & _CPD_OFF & _BODEN_OFF & _WDT_OFF &
_PWRTE_ON & _INTRC_OSC_NOCLKOUT

; pin assignments
    constant    nLED=5          ; indicator LED on GP5


;***** VARIABLE DEFINITIONS
        UDATA_SHR
sGPIO   res 1                   ; shadow copy of GPIO


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

;***** MAIN PROGRAM
        ; calibrate internal RC oscillator
        call    0x03FF          ; retrieve factory calibration value
        banksel OSCCAL          ;   then update OSCCAL
        movwf   OSCCAL

;***** Initialisation
        ; configure ports
        movlw   ~(1<<nLED)      ; configure LED pin (only) as an output
        banksel TRISIO
        movwf   TRISIO
        ; configure comparator
        movlw   0<<CINV|b'010'
                                ; select mode 2 (CM = 010):
                                ;   +ref is CIN+, -ref is CIN-,
                                ;   comparator on, no external output
                                ; output not inverted (CINV = 0)
        banksel CMCON           ;  -> COUT = 1 if CIN+ > CIN-
        movwf   CMCON


;***** Main loop
mainloop
        ; display comparator output
        clrf    sGPIO           ; assume COUT = 0 -> LED off
        banksel CMCON
        btfsc   CMCON,COUT      ; if comparator output high
        bsf     sGPIO,nLED      ;   turn on LED

        movf    sGPIO,w         ; copy shadow to GPIO
        banksel GPIO
        movwf   GPIO

        ; repeat forever
        goto    mainloop


        END
```

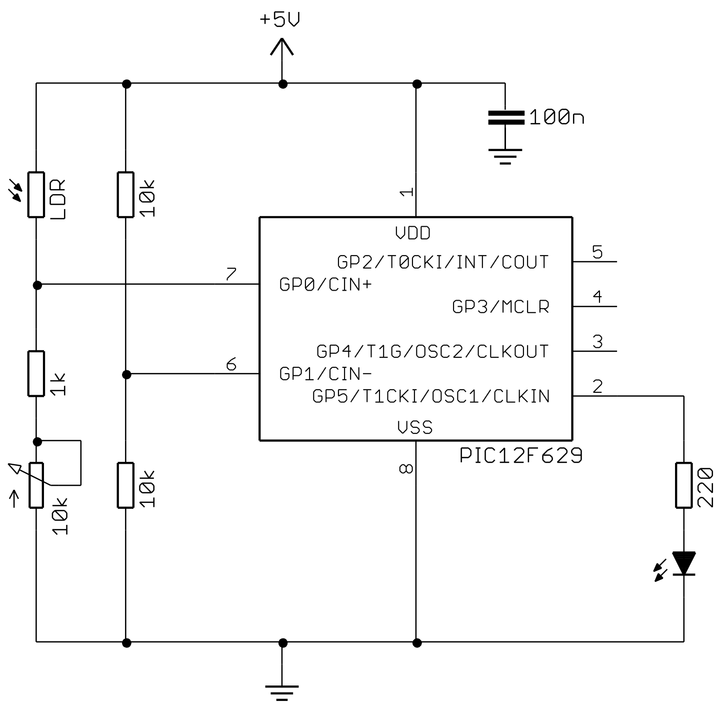
You should find that, as you turn the potentiometer past halfway, the LED ('DS4') turns on and off.

To make the circuit a little more interesting, and closer to a real-life application, you could add a light-dependent resistor (LDR), as shown below.

To build this using the Low Pin Count Demo Board, you should remove jumper JP5 (you may need to cut the PCB trace and solder a jumper in place, before you can remove it…), to disconnect the potentiometer from the +5 V supply[4].

The LDR can then be connected between pin 7 on the 14-pin header and +5 V.

Note that, if you are not using the demo board, you do not need to include the 1 kΩ resistor in series with the potentiometer; it is only shown in this circuit because the demo board includes that 1 kΩ resistor. Or, you could use a fixed 10 kΩ resistor instead of the pot.



[4] The +5 V supply connected through JP5 is also used to hold $\overline{\text{MCLR}}$ high when the reset line is tri-stated – as it will be if you are using a PICkit 2 with MPLAB and have selected the '3-State on "Release from Reset"' setting. For correct operation with JP5 removed, you must either disable this setting, or use internal $\overline{\text{MCLR}}$.

The exact resistance range of the LDR is not important; any LDR that varies around 10 kΩ or so for normal lighting conditions would be suitable.

If you do not have an LDR, there is no problem with continuing to use the potentiometer-only circuit for these lessons; adjusting the pot simulates changing light levels. But it's certainly more fun to build a circuit that responds to light!

When you build this circuit and use the above code, you will find that the LED turns on when the LDR is illuminated (you may need to adjust the pot to set the threshold for the ambient light level). That's because, when an LDR is illuminated, its resistance falls, increasing the voltage at CIN+.

If you would prefer it to work the other way, so that the LED is lit when the light level falls (simulating e.g. the way that streetlamps turn on automatically when it gets dark), there's no need to change the circuit connections or program logic. Simply change the comparator initialisation instructions, to invert the output, by setting the CINV bit:

```
        movlw   1<<CINV|b'010'
                                ; select mode 2 (CM = 010):
                                ;   +ref is CIN+, -ref is CIN-,
                                ;   comparator on, no external output
                                ; inverted output (CINV = 1)
        banksel CMCON           ;  -> COUT = 1 if CIN+ < CIN-
        movwf   CMCON
```

Programmable voltage reference

The problem with using external resistors to provide a voltage reference, as in the examples above, is that you must use one of the PIC's I/O pins, to act as the reference input. This can be avoided by using an internally-generated reference voltage as one of the comparator's inputs.

The PIC12F629 provides an internal voltage reference (CVREF), which can be configured to generate one of a range of 32 voltages, between 0 V and 0.72 × VDD.

The comparator voltage reference is controlled by the VRCON register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VRCON	VREN	-	VRR	-	VR3	VR2	VR1	VR0

The VREN bit enables (turns on) the voltage reference.

To use the voltage reference, set VREN = 1.

The reference voltage is set by the VR<3:0> bits and VRR, which selects a high or low voltage range:

VRR = 1 selects the low range, where CVREF = VR<3:0>/24 × VDD.

VRR = 0 selects the high range, where CVREF = VDD/4 + VR<3:0>/32 × VDD.

Note that the generated voltages are relative to VDD. Unlike the PIC16F506, described in baseline lesson 9, the PIC12F629 does not provide an absolute voltage reference.

The available reference voltages are summarised in the following table, as a fraction of VDD and as an absolute voltage for the case where VDD = 5 V:

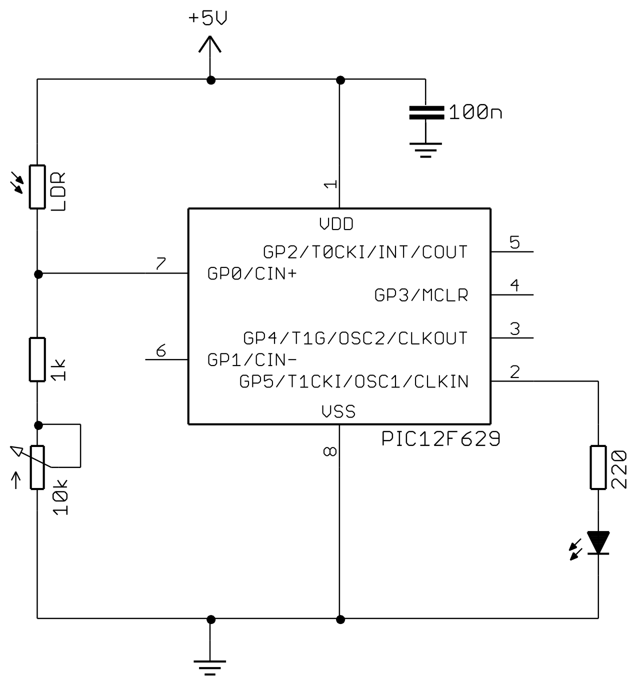
VRR = 1 (low range)		
VR<3:0>	fraction VDD	CVREF (VDD = 5V)
0	0.000	0.00V
1	0.042	0.21V
2	0.083	0.42V
3	0.125	0.62V
4	0.167	0.83V
5	0.208	1.04V
6	0.250	1.25V
7	0.292	1.46V
8	0.333	1.67V
9	0.375	1.87V
10	0.417	2.08V
11	0.458	2.29V
12	0.500	2.50V
13	0.542	2.71V
14	0.583	2.92V
15	0.625	3.12V

VRR = 0 (high range)		
VR<3:0>	fraction VDD	CVREF (VDD = 5V)
0	0.250	1.25V
1	0.281	1.41V
2	0.313	1.56V
3	0.344	1.72V
4	0.375	1.88V
5	0.406	2.03V
6	0.438	2.19V
7	0.469	2.34V
8	0.500	2.50V
9	0.531	2.66V
10	0.563	2.81V
11	0.594	2.97V
12	0.625	3.13V
13	0.656	3.28V
14	0.688	3.44V
15	0.719	3.59V

Note that the low and high ranges overlap, with 0.250×VDD, 0.500×VDD and 0.625×VDD selectable in both. Thus, of the 32 selectable voltages, only 29 are unique.

Since the voltage reference module draws current when enabled, you should turn it off by clearing VREN to minimise power consumption in sleep mode – unless of course you are using a comparator change, with CVREF as a comparator input, to wake the device from sleep (see later in this lesson), in which case the voltage reference needs to remain on.

If we use the programmable voltage reference to generate the fixed threshold, we can remove the 10 kΩ voltage divider resistors from the circuit, as shown on the right.



On the PIC12F629, CVREF can only be used as the positive comparator reference. Since the signal from the LDR in our circuit is connected to CIN+, we have to configure CIN+ as the *negative* reference[5].

Luckily comparator mode 6 (binary ‘110’) does what we want, if we set CIS = 1:

```
        ; configure comparator
        movlw   b'110'|1<<CIS|0<<CINV
                                     ; select mode 6 (CM = 110):
                                     ;   +ref is CVref,
                                     ;   -ref is CIN+ (CIS = 1),
                                     ;   no external output,
                                     ;   comparator on
                                     ; output not inverted (CINV = 0)
        banksel CMCON                ;  -> COUT = 1 if CIN+ < CVref
        movwf   CMCON
```

We then need to configure the voltage reference to generate the voltage we want.

In the previous examples, the resistor divider created a reference voltage of 2.5 V (0.5 × V_{DD}), but the “light meter” will be more sensitive if we lower this threshold, to (say) 1.5 V. Of course, since the reference is programmable, you can choose whatever level works best for you.

The closest match to 1.5 V is generated when VRR = 1 (selecting the low range), and VR = 7, giving a threshold of 0.292 × 5.0 V = 1.46 V (when V_{DD} = 5.0 V):

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

Complete program

Although the rest of the program is the same as in the previous example, here is the full listing, so that you can see where these fragments fit in:

```
;************************************************************************
;   Description:    Lesson 9, example 2                                 *
;                                                                       *
;   Demonstrates basic use of programmable voltage reference           *
;                                                                       *
;   Turns on LED when voltage on CIN+ < 1.5 V                           *
;                                                                       *
;************************************************************************
;   Pin assignments:                                                    *
;       CIN+  - voltage to be measured (e.g. pot output or LDR)         *
;       GP5   - indicator LED                                           *
;                                                                       *
;************************************************************************

    list        p=12F629
    #include    <p12F629.inc>

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

    radix       dec
```

[5] This is the opposite to the examples in baseline lesson 9, because the internal voltage references on the PIC16F506 are only available as negative, not positive, comparator references.

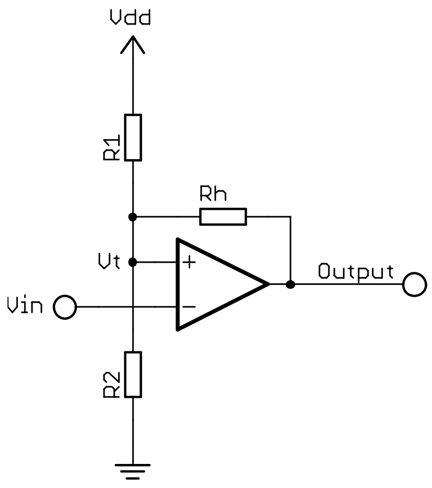
```
;***** CONFIGURATION
                ; ext reset, no code or data protect, no brownout detect,
                ; no watchdog, power-up timer, 4Mhz int clock
    __CONFIG    _MCLRE_ON & _CP_OFF & _CPD_OFF & _BODEN_OFF & _WDT_OFF &
_PWRTE_ON & _INTRC_OSC_NOCLKOUT

; pin assignments
    constant    nLED=5              ; indicator LED on GP5


;***** VARIABLE DEFINITIONS
        UDATA_SHR
sGPIO   res 1                       ; shadow copy of GPIO


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

;***** MAIN PROGRAM
        ; calibrate internal RC oscillator
        call    0x03FF              ; retrieve factory calibration value
        banksel OSCCAL              ;   then update OSCCAL
        movwf   OSCCAL

;***** Initialisation
        ; configure ports
        movlw   ~(1<<nLED)          ; configure LED pin (only) as an output
        banksel TRISIO
        movwf   TRISIO
        ; configure comparator
        movlw   b'110'|1<<CIS|0<<CINV
                                    ; select mode 6 (CM = 110):
                                    ;   +ref is CVref,
                                    ;   -ref is CIN+ (CIS = 1),
                                    ;   no external output,
                                    ;   comparator on
                                    ; output not inverted (CINV = 0)
        banksel CMCON               ;  -> COUT = 1 if CIN+ < CVref
        movwf   CMCON
        ; configure voltage reference
        movlw   1<<VRR|.7|1<<VREN
                                    ; CVref = 0.292*Vdd (VRR = 1, VR = 7)
                                    ; enable voltage reference (VREN = 1)
        banksel VRCON               ; -> CVref = 1.5 V (if Vdd = 5.0 V)
        movwf   VRCON

;***** Main loop
mainloop
        ; display comparator output
        clrf    sGPIO           ; assume COUT = 0 -> LED off
        banksel CMCON
        btfsc   CMCON,COUT      ; if comparator output high (CIN+ < 1.5 V)
        bsf     sGPIO,nLED      ;   turn on LED

        movf    sGPIO,w         ; copy shadow to GPIO
        banksel GPIO
        movwf   GPIO
        ; repeat forever
        goto    mainloop

        END
```

Adding hysteresis

You will notice, as the light level changes slowly past the threshold where the LED turns on and off, that the LED appears to fade in and out in brightness. This is caused by noise in the circuit and fluctuations in the light source (particularly at 50 or 60 Hz, from mains-powered incandescent or fluorescent lamps): when the input is very close to the threshold voltage, small input voltage variations due to noise and/or fluctuating light levels cause the comparator to rapidly switch between high and low. This rapid switching, similar to switch bounce, can be a problem if the microcontroller is supposed to count input transitions, or to perform an action on each change.

To avoid this phenomenon, *hysteresis* can be added to a comparator, by adding positive feedback – feeding some of the comparator's output back to its positive input.

Consider the comparator circuit shown on the right.



The threshold voltage, Vt, is set by a voltage divider, formed by resistors R1 and R2.

This would normally set the threshold at $Vt = \frac{R2}{R1 + R2} Vdd$.

However, resistor Rh feeds some of the comparator's output back, increasing the threshold to some higher level, Vth, when the output is high, and decreasing it to a lower level, Vtl, when the output is low.

Now consider what happens when Vin is low (less than Vtl) and begins to increase. Initially, since Vin < Vt, the comparator output is high, and the threshold is high: Vt = Vth.

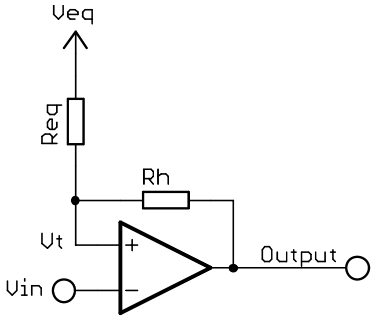
Eventually Vin rises above Vth, and the comparator output goes low, lowering the threshold: Vt = Vtl.

Now suppose the input voltage begins to fall. As it falls past the high threshold, Vth, nothing happens. It has to keep falling, all the way down to the low threshold, Vtl, before the comparator output changes again.

There are now two voltage thresholds: one (the higher) applying when the input signal is rising; the other (lower) applying when the input is falling. The comparator's output depends not only on its inputs, but on their history – a key characteristic of hysteresis.

The voltage difference between the two thresholds is known as the hysteresis band: Vhb = Vth – Vtl.

This is the amount the input signal has to fall, after rising through the high threshold, or rise, after falling through the low threshold, before the comparator output switches again. It should be higher than the expected noise level in the circuit, making the comparator immune to most noise.



To calculate the high and low thresholds, recall Thévenin's theorem, which states that any two-terminal network of resistors and voltage sources can be replaced by a single voltage source, Veq, in series with a single resistor, Req.

Thus, the circuit above is equivalent to that on the left, where

$$Veq = \frac{R2}{R1 + R2} Vdd$$

and Req is the parallel combination of R1 and R2: $Req = \frac{R1R2}{R1 + R2}$

Therefore, when the comparator's output is low (= 0 V):

$$Vtl = \frac{Rh}{Rh + Req} Veq \qquad \text{(thus Vtl < Veq)}$$

and, when the comparator's output is high (= Vdd):

$$Vth = Veq + \frac{Req}{Rh + Req}\left(Vdd - Veq\right) \qquad \text{(thus Vth > Veq)}$$

This little bit of mathematics proves that Vth > Vtl, that is, that the high input threshold is higher than the low input threshold.

The comparator output can be made available on the COUT pin (shared with GP2); we can use this to add hysteresis to the circuit.

The comparator output is enabled in comparator modes 1, 3 and 5 (binary 001, 011 and 101).

Additionally, TRISIO<2> must be cleared to allow the COUT signal to appear on the GP2/COUT pin.

Note: To use a pin as a digital output, any comparator output assigned to that pin must be disabled. Comparator outputs are disabled on start-up.

In most examples of comparator hysteresis, the comparator's positive input is used as the threshold, and positive feedback is used to alter that threshold, as shown above.

However, if the programmable voltage reference is used as the positive reference, it is not possible to use feedback to adjust the threshold; being an internal reference, there is no way to drive it higher or lower.

But that's not a problem – it is also possible to introduce hysteresis by feeding the comparator output into the negative input signal (i.e. the signal being measured) – we only need to invert the comparator output, so that the overall feedback is still positive.

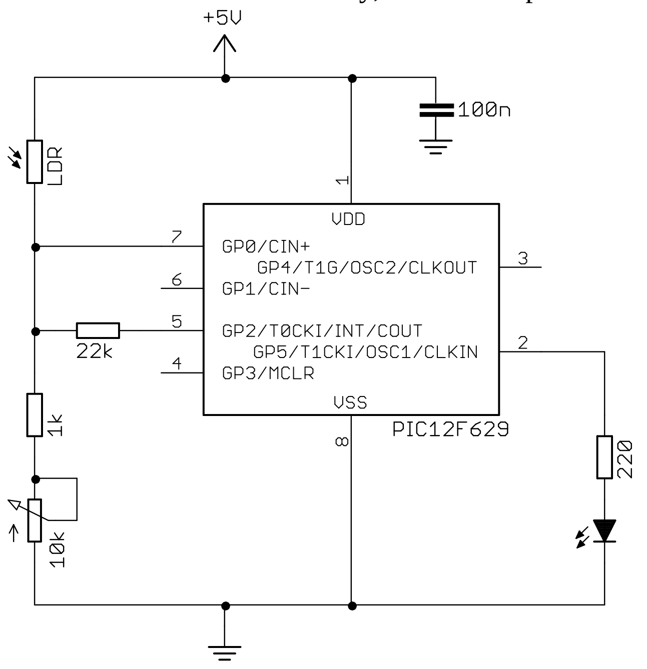
When the negative input is higher than the threshold, the (inverted) comparator output goes high, pulling the input even higher, via the feedback resistor. The circuit driving the input then has to effectively work harder, against the comparator's output, to bring the input back below the threshold. Similarly, when the input is low, the comparator's output goes low, dragging the input even lower, meaning that the input drive has to increase further before the comparator will change state again.

Suppose there is no feedback resistor and that the voltage on CIN+ is equal to the threshold voltage of 1.5 V. This would happen if the pot is set to 9 kΩ, for a resistance between CIN+ and ground of 10 kΩ, with the light level such that the LDR has a resistance of 23.3 kΩ:

Vin = 10 / (10 + 23.3) × 5 V = 1.5 V

With the input so close to the threshold, we would expect the output to jitter.

Now suppose that we add a 22 kΩ feedback resistor, as shown on the right.



The effect is similar to that for threshold voltage feedback. The circuit is equivalent to a source (input) voltage of 1.5 V, connected to CIN+ via a resistance of:

Req = (23.3 × 10) / (23.3 + 10) kΩ = 7.0 kΩ

When the comparator output is low, the feedback resistor pulls the input voltage down to:

Vinl = 22 / (22 + 7.0) × 1.5 V = 1.14 V

When the comparator is high, the input voltage is pulled up to:

Vinh = 1.5 V + 7.0 / (22+ 7.0) × (5.0 V – 1.5 V) = 2.34 V

Note that the voltages calculated above are not input thresholds. The comparator threshold is still the internal voltage reference of 1.5 V. These calculations only serve to demonstrate that the addition of positive feedback pulls the input lower when the comparator output is low and higher when the output is high, making the input much less sensitive to small changes.

Note also that, if you build this circuit using Microchip's Low Pin Count demo board and use a PICkit 2 to program and power it, you will see different voltage levels if you test this circuit while the PICkit 2 is connected. This is because the GP0 (CIN+) pin is used for in-circuit programming and debugging, and is loaded by the PICkit 2, changing the effective resistances from those shown.

The code is essentially the same as before, but we must select comparator mode 5 (binary 101) instead of mode 6, to enable COUT:

```
        ; configure comparator
        movlw   b'101'|1<<CIS|1<<CINV
                                    ; select mode 5 (CM = 101):
                                    ;   +ref is CVref,
                                    ;   -ref is CIN+ (CIS = 1),
                                    ;   external output enabled,
                                    ;   comparator on
                                    ; inverted output (CINV = 1)
        banksel CMCON               ;  -> COUT = 1 if CIN+ > CVref,
        movwf   CMCON               ;     COUT pin enabled
```

It is very important to set the CINV bit. If it is not set, the comparator output, appearing on COUT, will not be inverted, and, given that it is being fed into the comparator's negative input, the overall feedback will be negative, instead of the positive feedback needed to create hysteresis.

But before the comparator output can actually appear on the COUT pin, that pin has to be configured as an output:

```
        banksel TRISIO
        bcf     TRISIO,GP2          ; configure COUT (GP2) as an output
```

Note that we could have done this when initialising the port, for example:

```
        movlw   ~(1<<nLED|1<<GP2)   ; configure LED pin and COUT as outputs
        banksel TRISIO
        movwf   TRISIO
```

However, it is usually better to keep all code associated with the comparator configuration (such as enabling the COUT pin) with the rest of the comparator initialisation code; if you later want to change how the comparator is configured, you don't have to remember to make a corresponding in change in some other section of initialisation code. Modular code tends to be more easily maintained.

Finally, because the comparator output is now inverted, the bit test in the display code has to be changed, from:

```
        btfsc   CMCON,COUT      ; if comparator output high (CIN+ < 1.5 V)
        bsf     sGPIO,nLED      ;   turn on LED
```

to:

```
        btfss   CMCON,COUT      ; if comparator output low (CIN+ < 1.5 V)
        bsf     sGPIO,nLED      ;   turn on LED
```

So the code for turning on a LED, when the LDR is illuminated, using the internal programmable reference, with hysteresis, is:

```
;***** Initialisation
        ; configure ports
        movlw   ~(1<<nLED)          ; configure LED pin (only) as an output
        banksel TRISIO
        movwf   TRISIO
        ; configure comparator
        movlw   b'101'|1<<CIS|1<<CINV
                                    ; select mode 5 (CM = 101):
                                    ;   +ref is CVref,
                                    ;   -ref is CIN+ (CIS = 1),
                                    ;   external output enabled,
                                    ;   comparator on
                                    ; inverted output (CINV = 1)
        banksel CMCON               ;  -> COUT = 1 if CIN+ > CVref,
        movwf   CMCON               ;     COUT pin enabled
        banksel TRISIO
        bcf     TRISIO,GP2          ; configure COUT (GP2) as an output
        ; configure voltage reference
        movlw   1<<VRR|.7|1<<VREN
                                    ; CVref = 0.292*Vdd (VRR = 1, VR = 7)
                                    ; enable voltage reference (VREN = 1)
        banksel VRCON               ; -> CVref = 1.5 V (if Vdd = 5.0 V)
        movwf   VRCON


;***** Main loop
mainloop
        ; display inverse of comparator output
        clrf    sGPIO           ; assume COUT = 1 -> LED off
        banksel CMCON
        btfss   CMCON,COUT      ; if comparator output low (CIN+ < 1.5 V)
        bsf     sGPIO,nLED      ;   turn on LED

        movf    sGPIO,w         ; copy shadow to GPIO
        banksel GPIO
        movwf   GPIO

        ; repeat forever
        goto    mainloop
```

Comparator interrupt

The comparator module can be selected as an interrupt source. This is similar to the interrupt-on-change (IOC) facility described in lesson 7, except that, instead of an interrupt being generated when a digital input changes, the comparator interrupt is triggered whenever the comparator's output changes.

This means that, if you wish to respond to a change in the comparator output (i.e. the analog input crossing a threshold), there is no need to continually poll COUT, as we were doing above. Instead, you can choose to generate an interrupt whenever COUT changes, and handle the "comparator change" event in your interrupt service routine (ISR) – much as we did for pin change interrupts in lesson 7.

As with all other interrupts (see lesson 6), the comparator interrupt has an associated enable bit, which has to be set to allow the comparator to trigger interrupts, and an interrupt flag bit, which is set whenever the comparator's output changes.

So far, all of the interrupt enable and flag bits we've looked at have been in the INTCON register. But the comparator is considered to be a *peripheral* (not part of the PIC core), and its interrupt bits are held in peripheral interrupt registers.

The comparator interrupt is enabled by setting the CMIE bit in the PIE1 (peripheral interrupt enable 1) register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PIE1	EEIE	–	–	–	CMIE	–	–	TMR1IE

We saw in lesson 6 that, to enable any interrupts to occur, it is necessary to set the GIE (global interrupt enable) bit in the INTCON register, as well as the enable bits (such as CMIE) for whichever interrupt sources you wish to use.

However, to enable any peripheral interrupts (where the enable bit is in PIE1), you much also set the PEIE (peripheral interrupt enable) bit in INTCON:

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

So, to enable comparator interrupts, we must set CMIE, PEIE and GIE.

The comparator interrupt flag, CMIF, is found in PIR1 (peripheral interrupt register 1):

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PIR1	EEIF	–	–	–	CMIF	–	–	TMR1IF

CMIF is set whenever the comparator output (COUT) has changed since the last time CMCON was read or written.

As with other interrupt sources, the comparator interrupt flag is active, whether or not comparator interrupts are enabled. This means that you could, in principle, poll CMIF to detect changes in COUT – although it would normally make more sense to simply poll COUT directly.

In the interrupt service routine, you should read or write CMCON, to clear any existing *mismatch condition*, to allow you to successfully clear the CMIF flag. And to avoid false triggering (an interrupt occurring because of a previous change), you should also read or write CMCON and clear CMIF, immediately before enabling the comparator interrupt.

To demonstrate this, we can re-implement the last example, using interrupts.

Instead of polling COUT, we'll set the comparator module to trigger an interrupt whenever the input (light level) passes through the threshold, and have the ISR turn on or off the LED.

This then leaves the main program code free to perform other tasks – although in this example, all we will do is update the LED display.

We can keep the comparator and voltage reference configuration the same as before. It's a good idea to leave the hysteresis in place, as this minimises the number of comparator transitions – limiting the number of times the interrupt will be triggered.

In the initialisation code, we need to enable peripheral and global interrupts:

```
        movlw   1<<PEIE|1<<GIE      ; enable peripheral and global interrupts
        movwf   INTCON
```

Then we can enable the comparator interrupt, after reading CMCON to clear any comparator mismatch condition, and clearing the interrupt flag:

```
        banksel CMCON               ; enable comparator interrupt:
        movf    CMCON,w             ;   read CMCON to clear mismatch
        banksel PIR1
        bcf     PIR1,CMIF           ;   clear interrupt flag
        banksel PIE1
        bsf     PIE1,CMIE           ;   set enable bit
```

In the interrupt handler, we must clear the comparator mismatch condition which triggered this interrupt and (as always) clear the interrupt flag:

```
        banksel CMCON
        movf    CMCON,w             ; clear mismatch condition
        banksel PIR1
        bcf     PIR1,CMIF           ; clear interrupt flag
```

Since the comparator interrupt is triggered by any comparator output change, we can't know whether the interrupt was caused by the comparator input going high or low.

So, we check the value of COUT, and set the LED output accordingly:

```
        ; turn on LED if CIN+ < 1.5 V
        clrf    sGPIO               ; assume COUT = 1 -> LED off
        banksel CMCON
        btfss   CMCON,COUT          ; if comparator output low (CIN+ < 1.5 V)
        bsf     sGPIO,nLED          ;   turn on LED (using shadow register)
```

Note that, as we have done in the previous lessons on interrupts, the ISR does not update the LED output directly. Instead, a shadow register is used, to avoid any read-modify-write issues (see lesson 1) which may arise from the ISR updating pins independently of the main code.

In the main loop, we simply copy the shadow register to GPIO, as we have done before:

```
loop    banksel GPIO        ; continually copy shadow GPIO to port
        movf    sGPIO,w
        movwf   GPIO

        goto    loop        ; repeat forever
```

Complete program

Here is how it all fits together:

```
;************************************************************************
;                                                                       *
;   Description:    Lesson 9 example 4                                  *
;                                                                       *
;   Demonstrates use of comparator interrupt                            *
;   (assumes hysteresis is used to reduce triggering)                   *
;                                                                       *
;   Turns on LED when voltage on CIN+ < 1.5 V                           *
;                                                                       *
;************************************************************************
;                                                                       *
;   Pin assignments:                                                    *
;       CIN+  - voltage to be measured (e.g. pot output or LDR)         *
;       COUT  - comparator output (fed back to input via resistor)      *
;       GP5   - indicator LED                                           *
;                                                                       *
;************************************************************************

    list        p=12F629
    #include    <p12F629.inc>

    errorlevel  -302    ; no "register not in bank 0" warnings
    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, 4 Mhz int clock
    __CONFIG    _MCLRE_ON & _CP_OFF & _CPD_OFF & _BODEN_OFF & _WDT_OFF &
_PWRTE_ON & _INTRC_OSC_NOCLKOUT

; pin assignments
    constant    nLED=5              ; indicator LED on GP5


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

GENVAR      UDATA_SHR               ; general variables
sGPIO       res 1                       ; shadow copy of GPIO


;************************************************************************
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
        ; *** Service comparator interrupt
        ;   Triggered on any comparator output change,
        ;   caused by comparator input crossing 1.5 V threshold
        ;
        banksel CMCON
        movf    CMCON,w             ; clear mismatch condition
        banksel PIR1
        bcf     PIR1,CMIF           ; clear interrupt flag
        ; turn on LED if CIN+ < 1.5 V
        clrf    sGPIO               ; assume COUT = 1 -> LED off
        banksel CMCON
        btfss   CMCON,COUT          ; if comparator output low (CIN+ < 1.5 V)
        bsf     sGPIO,nLED          ;   turn on LED (using shadow register)

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
Start   ; calibrate internal RC oscillator
        call    0x03FF              ; retrieve factory calibration value
        banksel OSCCAL              ;   then update OSCCAL
        movwf   OSCCAL

;***** Initialisation
        ; configure ports
        movlw   ~(1<<nLED)          ; configure LED pin (only) as an output
        banksel TRISIO
        movwf   TRISIO
        ; configure comparator
        movlw   b'101'|1<<CIS|1<<CINV
                                    ; select mode 5 (CM = 101):
                                    ;   +ref is CVref,
                                    ;   -ref is CIN+ (CIS = 1),
                                    ;   external output enabled,
                                    ;   comparator on
                                    ; inverted output (CINV = 1)
        banksel CMCON               ;  -> COUT = 1 if CIN+ > CVref,
        movwf   CMCON               ;     COUT pin enabled
        banksel TRISIO
        bcf     TRISIO,GP2          ; configure COUT (GP2) as an output
        ; configure voltage reference
        movlw   1<<VRR|.7|1<<VREN
                                    ; CVref = 0.292*Vdd (VRR = 1, VR = 7)
                                    ; enable voltage reference (VREN = 1)
        banksel VRCON               ; -> CVref = 1.5 V (if Vdd = 5.0 V)
        movwf   VRCON
        ; configure interrupts
        movlw   1<<PEIE|1<<GIE      ; enable peripheral and global interrupts
        movwf   INTCON
        banksel CMCON               ; enable comparator interrupt:
        movf    CMCON,w             ;   read CMCON to clear mismatch
        banksel PIR1
        bcf     PIR1,CMIF           ;   clear interrupt flag
        banksel PIE1
        bsf     PIE1,CMIE           ;   set enable bit
```

```
;***** Main loop
loop
        ; continually copy shadow GPIO to port
        banksel GPIO
        movf    sGPIO,w
        movwf   GPIO

        ; repeat forever
        goto    loop


        END
```

Wake-up on comparator change

As we saw in lesson 7, most PICs can be put into standby, or sleep mode, to conserve power until they are woken by an external event. We also saw that, for midrange PICs, that event can be any which is capable of triggering an interrupt (without the oscillator running), such as a change on an IOC-enabled digital input; it can also be a change on a comparator output.

That's useful if, for example, your application is battery-powered and has to spend a long time waiting to respond to an input level change from a sensor.

Recall that, if an enabled interrupt source is triggered while the PIC is in sleep mode, the device will wake from sleep, set the corresponding interrupt flag to indicate which event occurred, execute the instruction following 'sleep' and, if global interrupts are enabled (GIE = 1), then enter the interrupt service routine.

If you only want to use an external event, such as a comparator change, to wake the device from sleep, without actually triggering an interrupt, simply disable interrupts (clear GIE) before entering sleep mode.

To demonstrate this, we can use the previous circuit and configure the PIC to wake on comparator change, and then go to sleep. When the input signal crosses the threshold level, the comparator output will change, and the PIC will wake. To indicate this we can turn on the LED for one second. The PIC can then go back to sleep, to wait until the next comparator change.

The comparator and voltage reference can be configured as we have done before, but we should add a delay of 10 ms or so to allow it to settle before entering standby. This is necessary because signal levels can take a while to settle after power-on, and if the comparator output was changing while going into sleep mode, the PIC would immediately wake and the LED would flash – a false trigger.

To enable wake on comparator change, we need to enable the comparator interrupt, but not global interrupts (since we're not actually using interrupts here; we're only using an interrupt source to wake the device):

```
        ; enable comparator interrupt (for wake on change)
        bsf     INTCON,PEIE         ; enable peripheral interrupts
        banksel PIE1
        bsf     PIE1,CMIE           ; and comparator interrupt
```

Within the main loop, we can start by turning off the LED, because we don't want it lit on start-up:

```
        ; turn off LED
        banksel GPIO
        bcf     GPIO,nLED
```

Before entering sleep mode, it is very important to clear any existing comparator mismatch condition, as well as CMIF:

```
        banksel CMCON               ; read CMCON to clear comparator mismatch
        movf    CMCON,w
        banksel PIR1
        bcf     PIR1,CMIF           ; clear comparator interrupt flag

        sleep
```

The PIC will now sleep until it is woken by a comparator change, which we wish to show by lighting the LED for one second:

```
        ; turn on LED for 1 second
        banksel GPIO                ; turn on LED
        bsf     GPIO,nLED

        DelayMS 1000                ; delay 1 sec
```

Note that the delay is generated by the `DelayMS` macro, introduced in lesson 5.

Complete program

Here is the complete comparator wakeup example program:

```
;************************************************************************
;                                                                       *
;   Description:    Lesson 9, example 5                                 *
;                                                                       *
;   Demonstrates wake-up on comparator change                           *
;                                                                       *
;   Turns on LED for 1 sec when comparator output changes               *
;   then sleeps until the next change                                   *
;   (threshold is 1.5 V, set by programmable voltage ref),              *
;                                                                       *
;************************************************************************
;                                                                       *
;   Pin assignments:                                                    *
;       CIN+  - voltage to be measured (e.g. pot output or LDR)         *
;       GP5   - indicator LED                                           *
;                                                                       *
;************************************************************************

    list        p=12F629
    #include    <p12F629.inc>

    #include    <stdmacros-mid.inc>   ; DelayMS - delay in milliseconds

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

    radix       dec

    EXTERN      delay10       ; W x 10ms delay


;***** CONFIGURATION
                ; ext reset, no code or data protect, no brownout detect,
                ; no watchdog, power-up timer, 4Mhz int clock
    __CONFIG    _MCLRE_ON & _CP_OFF & _CPD_OFF & _BODEN_OFF & _WDT_OFF &
_PWRTE_ON & _INTRC_OSC_NOCLKOUT
```

```
; pin assignments
    constant    nLED=5              ; indicator LED on GP5


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

;***** MAIN PROGRAM
        ; calibrate internal RC oscillator
        call    0x03FF              ; retrieve factory calibration value
        banksel OSCCAL              ;   then update OSCCAL
        movwf   OSCCAL

;***** Initialisation
        ; configure ports
        movlw   ~(1<<nLED)          ; configure LED pin (only) as an output
        banksel TRISIO
        movwf   TRISIO
        ; configure comparator
        movlw   b'110'|1<<CIS|0<<CINV
                                    ; select mode 6 (CM = 110):
                                    ;   +ref is CVref,
                                    ;   -ref is CIN+ (CIS = 1),
                                    ;   no external output,
                                    ;   comparator on
                                    ; output not inverted (CINV = 0)
        banksel CMCON               ;  -> COUT = 1 if CIN+ < CVref
        movwf   CMCON
        ; configure voltage reference
        movlw   1<<VRR|.7|1<<VREN
                                    ; CVref = 0.292*Vdd (VRR = 1, VR = 7)
                                    ; enable voltage reference (VREN = 1)
        banksel VRCON               ; -> CVref = 1.5 V (if Vdd = 5.0 V)
        movwf   VRCON
        ; delay 10 ms to allow comparator and voltage ref to settle
        DelayMS 10
        ; enable comparator interrupt (for wake on change)
        bsf     INTCON,PEIE         ; enable peripheral interrupts
        banksel PIE1
        bsf     PIE1,CMIE           ; and comparator interrupt


;***** Main loop
mainloop
        ; turn off LED
        banksel GPIO
        bcf     GPIO,nLED

        ; enter sleep mode
        banksel CMCON               ; read CMCON to clear comparator mismatch
        movf    CMCON,w
        banksel PIR1
        bcf     PIR1,CMIF           ; clear comparator interrupt flag

        sleep

        ; turn on LED for 1 second
        banksel GPIO                ; turn on LED
        bsf     GPIO,nLED

        DelayMS 1000                ; delay 1 sec
        ; repeat forever
        goto    mainloop


        END
```

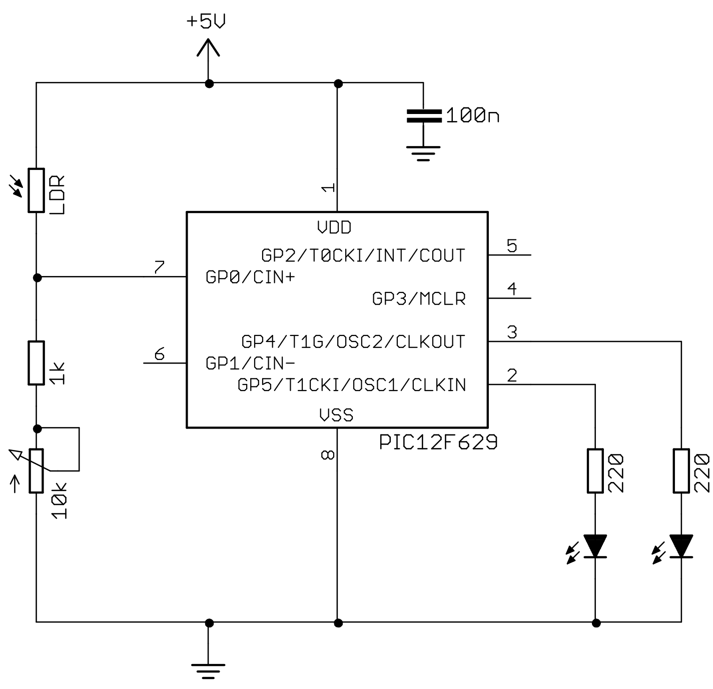
Finally, you should be aware that, if a comparator is turned on when the PIC enters sleep mode, it will continue to draw current. If the comparator is to wake the PIC up when an input level changes, then of course it has to remain active, using power, while the PIC is in standby. But if you're not using wait on comparator change, you should turn off the comparator module (select mode 0 or mode 7) before entering sleep mode, to save power.

Note: To minimise power consumption, turn off comparators before entering sleep mode.

Comparing a signal against a range of thresholds

It is sometimes necessary to check that a signal is within certain limits; a band of allowed values – and have to take action if the signal is too high or too low.

Or you may have a number of thresholds you need to compare the signal against, perhaps defining "warning", "danger" and "automatic shut-down".



To demonstrate how the programmable voltage reference can be used to achieve this, we can add another LED to the LDR circuit we used earlier, as shown on the right. The LED on GP5 will indicate a low level of illumination, and the LED on GP4 will indicate bright light. When neither LED is lit, the light level will be in the middle; not too dim or too bright.

To test whether the input is within limits, we will first configure the programmable voltage reference to generate the "low" threshold voltage, compare the input with this low level, and then reconfigure the voltage reference to generate the "high" threshold and compare the input with this higher level.

This process could be extended to multiple input thresholds, by configuring the voltage reference to generate each threshold in turn. However, if you wish to test against more than a few threshold levels, you would probably be better off using an analog-to-digital converter (described in lesson 13).

This example uses 1.0 V as the "low" threshold and 2.0 V as the "high" threshold, but, since the reference is programmable, you can always choose your own levels!

The comparator is configured to compare CIN+ with CVREF, as we have done before.

The voltage reference can be configured to generate approximately 1.0 V, by:

```
        movlw   1<<VRR|.5|1<<VREN   ; configure voltage reference:
                                    ;   CVref = 0.208*Vdd (VRR = 1, VR = 5)
                                    ;   enable voltage reference (VREN = 1)
        banksel VRCON               ;   -> CVref = 1.04 V (if Vdd = 5.0 V)
        movwf   VRCON
```

The closest match to 2.0 V is obtained by:

```
        movlw   0<<VRR|.5|1<<VREN   ; configure voltage reference:
                                    ;   CVref = 0.406*Vdd (VRR = 0, VR = 5)
                                    ;   enable voltage reference (VREN = 1)
        banksel VRCON               ;   -> CVref = 2.03 V (if Vdd = 5.0 V)
        movwf   VRCON
```

After changing the voltage reference, it can take a little while for it to settle and generate a stable voltage. According to table 12-7 in the PIC12F629/675 data sheet, this settling time can be up to 10 µs.

Therefore, we should insert a 10 µs delay after configuring the voltage reference, before reading the comparator output.

As we saw in lesson 1, a useful instruction for generating a two-instruction-cycle delay is '`goto $+1`'. Since each instruction cycle is 1 µs (with a 4 MHz processor clock), each '`goto $+1`' creates a 2 µs delay, and five of these instructions will give us the 10 µs delay we are after.

Since we need to insert this delay twice (once for each time we re-configure the voltage reference), it makes sense to define it as a macro:

```
; 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
```

This '`Delay10us`' macro can then be used to add each of the required 10 µs delays.

Complete program

Here is how these code fragments fit together:

```
;************************************************************************
;   Description:    Lesson 9, example 6                                 *
;                                                                       *
;   Demonstrates use of programmable voltage reference                  *
;   to test that a signal is within limits                              *
;   Turns on Low LED  when CIN+ < 1.0 V (low light level)               *
;         or High LED when CIN+ > 2.0 V (high light level)              *
;                                                                       *
;************************************************************************
;   Pin assignments:                                                    *
;       CIN+  - voltage to be measured (e.g. pot output or LDR)         *
;       GP5   - "Low" LED                                               *
;       GP4   - "High" LED                                              *
;                                                                       *
```

```
;************************************************************************

    list        p=12F629
    #include    <p12F629.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, 4Mhz int clock
    __CONFIG    _MCLRE_ON & _CP_OFF & _CPD_OFF & _BODEN_OFF & _WDT_OFF &
_PWRTE_ON & _INTRC_OSC_NOCLKOUT

; pin assignments
    constant    nLO=GP5             ; "Low" LED
    constant    nHI=GP4             ; "High" LED


;***** 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
sGPIO   res 1                       ; shadow copy of GPIO


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

;***** MAIN PROGRAM
        ; calibrate internal RC oscillator
        call    0x03FF              ; retrieve factory calibration value
        banksel OSCCAL              ;   then update OSCCAL
        movwf   OSCCAL

;***** Initialisation
        ; configure port
        movlw   ~(1<<nLO|1<<nHI)    ; configure LED pins as outputs
        banksel TRISIO
        movwf   TRISIO
        ; configure comparator
        movlw   b'110'|1<<CIS|1<<CINV
                                    ; select mode 6 (CM = 110):
                                    ;   +ref is CVref,
                                    ;   -ref is CIN+ (CIS = 1),
                                    ;   no external output,
                                    ;   comparator on
                                    ; inverted output (CINV = 1)
        banksel CMCON                   ;  -> COUT = 1 if CIN+ > CVref
        movwf   CMCON


;***** Main loop
mainloop
        clrf    sGPIO                   ; start with both LEDs off

        ;*** Test for low illumination
        ; set low input threshold
        movlw   1<<VRR|.5|1<<VREN       ; configure voltage reference:
                                        ;   CVref = 0.208*Vdd (VRR = 1, VR = 5)
                                        ;   enable voltage reference (VREN = 1)
        banksel VRCON                   ;   -> CVref = 1.04 V (if Vdd = 5.0 V)
        movwf   VRCON

        Delay10us                       ; wait 10us to settle

        ; compare with input
        banksel CMCON
        btfss   CMCON,COUT              ; if CIN+ < CVref
        bsf     sGPIO,nLO               ;   turn on Low LED

        ;*** Test for high illumination
        ; set high input threshold
        movlw   0<<VRR|.5|1<<VREN       ; configure voltage reference:
                                        ;   CVref = 0.406*Vdd (VRR = 0, VR = 5)
                                        ;   enable voltage reference (VREN = 1)
        banksel VRCON                   ;   -> CVref = 2.03 V (if Vdd = 5.0 V)
        movwf   VRCON

        Delay10us                       ; wait 10us to settle

        ; compare with input
        banksel CMCON
        btfsc   CMCON,COUT              ; if CIN+ > CVref
        bsf     sGPIO,nHI               ;   turn on High LED

        ;*** Display test results
        movf    sGPIO,w                 ; copy shadow to GPIO
        banksel GPIO
        movwf   GPIO

        ; repeat forever
        goto    mainloop


        END
```

Testing two independent inputs

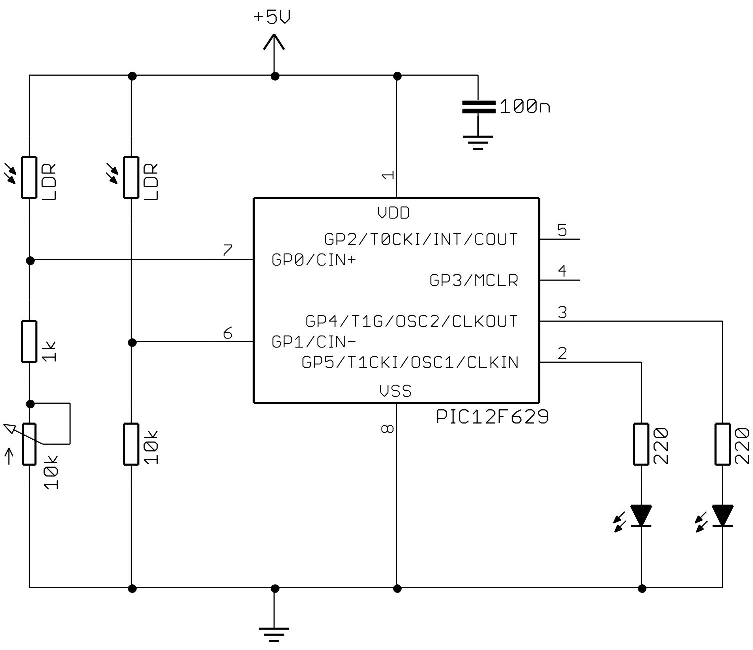
For a final example, suppose that we want to test two input signals (say, light level in two locations) by comparing them against a common reference. We could use two comparators, with each input connected to a different comparator, and the common reference voltage connected to both. But, what if we only have a single comparator, as in the PIC12F629?

Recall that, in comparator modes 5 and 6 (binary 101 and 110), the CIS bit in the CMCON register selects whether the CIN+ or CIN- pin is connected to the comparator's negative input.

In these modes, the negative input is said to be *multiplexed*, making it easy to switch between the CIN+ and CIN- pins, on the fly.

This means that we can select CIN+ as an input, perform a comparison with the internal reference voltage, and then select CIN- and repeat the comparison[6].

To demonstrate this, we can use the circuit on the right, where an additional LDR has been added to the circuit in the last example. We'll light the LED on GP4 when the CIN+ input is above 1.5 V (bright light), and the LED on GP5 when the CIN- input is above 1.5 V.



First, when configuring the comparator, there is no need to explicitly initialise CIS, since we'll be modifying it later:

```
        movlw   b'110'|1<<CINV
                                ; select mode 6 (CM = 110):
                                ;   +ref is CVref,
                                ;   -ref is CIN+ or CIN- (selected by CIS),
                                ;   no external output,
                                ;   comparator on
                                ; inverted output (CINV = 0)
        banksel CMCON           ;  -> COUT = 1 if -ref > CVref
        movwf   CMCON
```

Within the main loop, we can then select each input in turn:

```
        ; test input 1
        banksel CMCON           ; select CIN+ pin as -ref
        bsf     CMCON,CIS       ;   (CIS = 1)
```

After changing the comparator's configuration, or mode, it can take a while for it to settle and generate a valid output. According to table 12-6 in the PIC629/675 data sheet, this ("Comparator Mode Change to Output Valid") can take up to 10 µs. By coincidence, that's the same as the voltage reference settling time, discussed in the last example, although they are actually unrelated.

Therefore, we need to add a 10 µs delay after selecting a new input, before reading the comparator output.

[6] Since the voltage reference is programmable, you could setup a different threshold for each input, by reprogramming the reference voltage each time you switch between inputs – but for clarity we won't do that in this example.

This is similar to what we did in the last example, where we added a delay after changing the voltage reference, and we can re-use the 10 µs delay macro from that example:

```
        ; test input 1
        banksel CMCON                   ; select CIN+ pin as -ref
        bsf     CMCON,CIS               ;   (CIS = 1)
        Delay10us                       ; wait 10 us to settle
        btfsc   CMCON,COUT              ; if CIN+ > CVref
        bsf     sGPIO,nLED1             ;   turn on LED 1
```

We can then do the same again, selecting the other comparator input:

```
        ; test input 2
        banksel CMCON                   ; select CIN- pin as -ref
        bcf     CMCON,CIS               ;   (CIS = 0)
        Delay10us                       ; wait 10 us to settle
        btfsc   CMCON,COUT              ; if CIN- > CVref
        bsf     sGPIO,nLED2             ;   turn on LED 2
```

Complete program

Here is the full listing for the "two inputs with a common programmed voltage reference" program, showing how these fragments fit together:

```
;************************************************************************
;                                                                       *
;   Description:    Lesson 9, example 7                                 *
;                                                                       *
;   Demonstrates comparator input multiplexing                          *
;                                                                       *
;   Turns on: LED 1 when CIN+ > 1.5 V                                   *
;         and LED 2 when CIN- > 1.5 V                                   *
;                                                                       *
;************************************************************************
;                                                                       *
;   Pin assignments:                                                    *
;       CIN+  - input 1 (LDR/resistor divider)                          *
;       CIN-  - input 2 (LDR/resistor divider)                          *
;       GP4   - indicator LED 1                                         *
;       GP5   - indicator LED 2                                         *
;                                                                       *
;************************************************************************

    list        p=12F629
    #include    <p12F629.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, 4Mhz int clock
    __CONFIG    _MCLRE_ON & _CP_OFF & _CPD_OFF & _BODEN_OFF & _WDT_OFF &
_PWRTE_ON & _INTRC_OSC_NOCLKOUT

; pin assignments
    constant    nLED1=GP4           ; indicator LED 1
    constant    nLED2=GP5           ; indicator LED 2
```

```
;***** 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
sGPIO   res 1                       ; shadow copy of GPIO


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

;***** MAIN PROGRAM
        ; calibrate internal RC oscillator
        call    0x03FF              ; retrieve factory calibration value
        banksel OSCCAL              ;   then update OSCCAL
        movwf   OSCCAL

;***** Initialisation
        ; configure ports
        movlw   ~(1<<nLED1|1<<nLED2)    ; configure LED pins as outputs
        banksel TRISIO
        movwf   TRISIO
        ; configure comparator
        movlw   b'110'|1<<CINV
                                    ; select mode 6 (CM = 110):
                                    ;   +ref is CVref,
                                    ;   -ref is CIN+ or CIN- (selected by CIS),
                                    ;   no external output,
                                    ;   comparator on
                                    ; inverted output (CINV = 1)
        banksel CMCON               ;  -> COUT = 1 if -ref > CVref
        movwf   CMCON
        ; configure voltage reference
        movlw   1<<VRR|.7|1<<VREN
                                    ; CVref = 0.292*Vdd (VRR = 1, VR = 7)
                                    ; enable voltage reference (VREN = 1)
        banksel VRCON               ; -> CVref = 1.5 V (if Vdd = 5.0 V)
        movwf   VRCON


;***** Main loop
mainloop
        clrf    sGPIO               ; start with both LEDs off

        ; test input 1
        banksel CMCON               ; select CIN+ pin as -ref
        bsf     CMCON,CIS           ;   (CIS = 1)
        Delay10us                   ; wait 10 us to settle
        btfsc   CMCON,COUT          ; if CIN+ > CVref
        bsf     sGPIO,nLED1         ;   turn on LED 1
        ; test input 2
        banksel CMCON               ; select CIN- pin as -ref
        bcf     CMCON,CIS           ;   (CIS = 0)
        Delay10us                   ; wait 10 us to settle
        btfsc   CMCON,COUT          ; if CIN- > CVref
        bsf     sGPIO,nLED2         ;   turn on LED 2

        ; display test results
        movf    sGPIO,w             ; copy shadow to GPIO
        banksel GPIO
        movwf   GPIO

        ; repeat forever
        goto    mainloop


        END
```

We've seen in the last two examples that a single comparator can be used to "mimic" two or more comparators, by reprogramming the voltage reference and selecting different inputs, on the fly.

But that only works when you're polling the output, as we were in these examples. If you need to, for example, wake the PIC when an input goes either above a high threshold or below a low threshold, or respond immediately (via an interrupt) when either of two inputs passes a threshold, then you really do need two comparators.

So in the next lesson we'll introduce the 14-pin PIC16F684, which includes a dual comparator module, and take a look at some of the 16F684's enhanced features.