

Introduction to PIC Programming

Programming Baseline PICs in C

by David Meiklejohn, Gooligum Electronics

Lesson 6: Analog Comparators

[Baseline assembler lesson 9](#) explained how to use the analog comparators and absolute and programmable voltage references available on baseline PICs, such as the PIC16F506, using assembly language. This lesson demonstrates how to use C to access those facilities, re-implementing the examples using Microchip's XC8 (running in "Free mode") and CCS' PCB compilers¹.

In summary, this lesson covers:

- Basic use of the analog comparator modules available on the PIC16F506
- Using the internal absolute 0.6 V voltage reference
- Configuring and using the internal programmable voltage reference
- Enabling comparator output, to facilitate the addition of external hysteresis
- Wake-up on comparator change
- Driving Timer0 from a comparator output

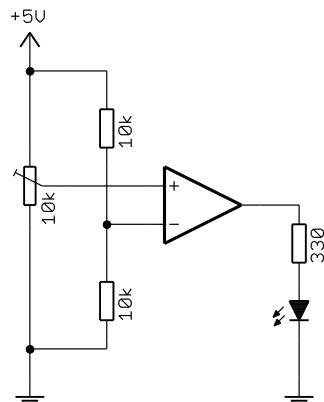
with examples for XC8 and CCS PCB.

Comparators

As we saw in [baseline assembler lesson 9](#), an *analog comparator* is a device which compares the voltages present on its positive and negative inputs. In normal (non-inverted) operation, the comparator's output is set to a logical "high" only when the voltage on the positive input is greater than that on the negative input; otherwise the output is "low". As such, they provide an interface between analog and digital circuitry.

In the circuit shown on the right, the comparator output will go high, lighting the LED, only when the potentiometer is set to a position past "half-way", i.e. positive input is greater than 2.5 V.

Comparators are typically used to detect when an analog input is above or below some threshold (or, if two comparators are used, within a defined band) – very useful for working with many types of real-world sensors. They are also used with digital inputs to match different logic levels, and to shape poorly defined signals.



¹ XC8 is available as a free download from www.microchip.com, and CCS PCB is bundled for free with MPLAB 8

Comparator 1

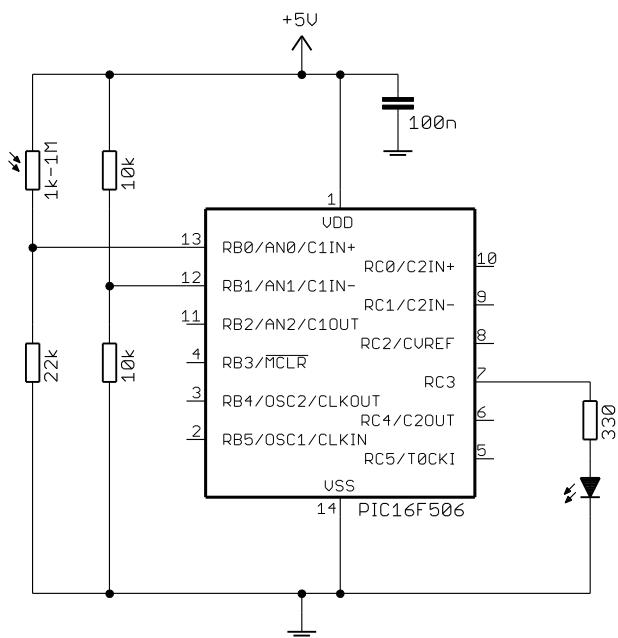
In [baseline assembler lesson 9](#), the circuit on the right, which includes a light dependent resistor (LDR, or CdS photocell), was used to demonstrate the basic operation of Comparator 1, the simpler of the two comparator modules in the PIC16F506.

The exact resistance range of the photocell is not important, but would ideally have a resistance of around 20 kΩ or so for normal indoor lighting conditions, so that the voltage at C1IN+ will vary around 2.5 V or so.

If you have the [Gooligum baseline training board](#), you can implement this circuit by placing a shunt across pins 2 and 3 ('LDR1') of JP24, connecting the photocell in the lower left of the board (PH1) to C1IN+, and in JP19 to enable the LED on RC3.

The connection to C1IN- (labelled 'GP/RA/RB1' on the board) is available as pin 9 on the 16-pin header. +V and GND are brought out on pins 15 and 16, respectively, making it easy to add the 10 kΩ resistors (supplied with the board), forming a voltage divider, by using the solderless breadboard.

If you are using the Microchip Low Pin Count Demo Board, the 10 kΩ potentiometer on that board, and the 1 kΩ resistor in series between it and C1IN+, must be used as the "fixed" resistance, forming the lower arm of the potential divider. You must also remove jumper JP5 (you may need to cut the PCB trace – ideally you'd install a jumper, so that you can reconnect it again later), to disconnect the pot from the +5 V supply. If you turn the pot all the way to the right, you'll have a total resistance of 11 kΩ between C1IN+ and ground. That means that ideally you'd use a photocell with a resistance of around 10 kΩ with normal indoor lighting. The photocell can then be connected between pin 7 on the 14-pin header and +5 V. Note that if you do not have a photocell available, you can still explore comparator operation by connecting the C1IN+ input directly to the centre tap on the 10 kΩ potentiometer.



We saw in [baseline assembler lesson 9](#) that, to configure Comparator 1 to behave like the standalone comparator shown on the previous page, where the output bit (C1OUT) indicates that the voltage on the C1IN+ input is higher than that on the C1IN- input, it is necessary to set the C1PREF, C1NREF and C1POL bits in the CM1CON0 register, and to turn on the comparator module by setting the C1ON bit:

```
movlw  1<<C1POL|1<<C1ON|1<<C1PREF|1<<C1NREF
      ; pos ref is C1IN+ (C1PREF = 1)
      ; neg ref is C1IN- (C1NREF = 1)
      ; normal polarity (C1POL = 1)
      ; comparator on (C1ON = 1)
movwf  CM1CON0           ; -> C1OUT = 1 if C1IN+ > C1IN-
```

The LED attached to RC3 was turned on when the comparator output was high (C1OUT = 1) by:

```
loop   btfsc  CM1CON0,C1OUT    ; if comparator output high
      bsf    LED                 ; turn on LED
      btfss  CM1CON0,C1OUT    ; if comparator output low
      bcf    LED                 ; turn off LED
      goto   loop                ; repeat forever
```

XC8

We saw in [lesson 3](#) that symbols, defined in the XC8 header files, can be used to represent register bits to construct a value to load into the **OPTION** register, for example:

```
OPTION = ~T0CS & ~PSA | 0b111;
```

However the **OPTION** register is an exception. The XC8 header files define most special function registers, including **CM1CON0**, as unions of structures containing bit-fields corresponding to that register's bits.

If you wish to set and/or clear a number of bits in a register such as **CM1CON0** at once, you can use a numeric constant such as:

```
CM1CON0 = 0b00101110;           // configure comparator 1:  
      //---1----          normal polarity (C1POL = 1)  
      //----1---          comparator on (C1ON = 1)  
      //----1--          -ref is C1IN- (C1NREF = 1)  
      //----1-          +ref is C1IN+ (C1PREF = 1)  
      //                      -> C1OUT = 1 if C1IN+ > C1IN-
```

This is ok, as long as you express the value in binary, so that it is obvious which bits are being set or cleared, and clearly commented, as above.

It is certainly much clearer than the equivalent:

```
CM1CON0 = 46;
```

However, a more natural way to approach this, in XC8, is to use a sequence of assignments, to set or clear the appropriate register bits via the bit-fields defined in the header files.

For example:

```
// configure comparator 1  
CM1CON0bits.C1PREF = 1;        // +ref is C1IN+  
CM1CON0bits.C1NREF = 1;        // -ref is C1IN-  
CM1CON0bits.C1POL = 1;         // normal polarity (C1IN+ > C1IN-)  
CM1CON0bits.C1ON = 1;          // turn comparator on
```

This is clear and easy to maintain, but a series of single-bit assignments like this requires more program memory than a whole-register assignment. It is also no longer an *atomic* operation, where all the bits are updated at once. This can be an important consideration in some instances², but it is not relevant here. Note also that the remaining bits in **CM1CON0** are not being explicitly set or cleared; that is ok because in this example we don't care what values they have.

The comparator's output bit, **C1OUT**, is available as the single-bit bit-field '**CM1CON0bits.C1OUT**'.

For example:

```
LED = CM1CON0bits.C1OUT;       // turn on LED iff comparator output high
```

With the above configuration, the LED will turn on when the LDR is illuminated.

If instead you wanted it to operate the other way, so that the LED is lit when the LDR is in darkness, you could invert the comparator output test, so that the LED is set high when **C1OUT** is low:

```
LED = !CM1CON0bits.C1OUT;     // turn on LED iff comparator output low
```

² this can be a consideration with mid-range PICs, where interrupts may be used

Alternatively, you can configure the comparator so that its output is inverted, using either:

```
CM1CON0 = 0b00001110;           // configure comparator 1:  
    //---0----  
    //----1--  
    //----1--  
    //----1-  
    //  
                                inverted polarity (C1POL = 0)  
                                comparator on (C1ON = 1)  
                                -ref is C1IN- (C1NREF = 1)  
                                +ref is C1IN+ (C1PREF = 1)  
                                -> C1OUT = 1 if C1IN+ < C1IN-
```

or:

```
// configure comparator 1  
CM1CON0bits.C1PREF = 1;        // +ref is C1IN+  
CM1CON0bits.C1NREF = 1;        // -ref is C1IN-  
CM1CON0bits.C1POL = 0;        // inverted polarity (C1IN+ < C1IN-)  
CM1CON0bits.C1ON = 1;          // turn comparator on
```

Complete program

Here is the complete inverted polarity version of the program, for XC8:

```
*****  
* Description: Lesson 6, example 1b  
*  
* Demonstrates basic use of Comparator 1 polarity bit  
*  
* Turns on LED when voltage on C1IN+ < voltage on C1IN-  
*  
*****  
* Pin assignments:  
*     C1IN+ = voltage to be measured (e.g. pot output or LDR)  
*     C1IN- = threshold voltage (set by voltage divider resistors)  
*     RC3    = indicator LED  
*  
*****  
#include <xc.h>  
  
***** CONFIGURATION *****/  
// ext reset, no code protect, no watchdog, 4 MHz int clock  
__CONFIG(MCLRE_ON & CP_OFF & WDT_OFF & IOSCFS_OFF & OSC_IntRC_RB4EN);  
  
// Pin assignments  
#define LED      PORTCbits.RC3    // indicator LED on RC3  
#define nLED      3                // (port bit 3)  
  
***** MAIN PROGRAM *****/  
void main()  
{  
    //*** Initialisation  
  
    // configure ports  
    TRISC = ~(1<<nLED);        // configure LED pin (only) as an output  
  
    // configure Comparator 1  
    CM1CON0bits.C1PREF = 1;      // +ref is C1IN+  
    CM1CON0bits.C1NREF = 1;      // -ref is C1IN-  
    CM1CON0bits.C1POL = 0;        // inverted polarity (C1IN+ < C1IN-)  
    CM1CON0bits.C1ON = 1;          // turn comparator on
```

```

//*** Main loop
for (;;)
{
    // continually display comparator output
    LED = CM1CON0bits.C1OUT;
}

```

CCS PCB

As we've come to expect, the CCS PCB compiler provides a built-in function for configuring the comparators: ‘`setup_comparator()`’.

It is used with symbols defined in the device-specific header files. For example, “16F506.h” contains:

```

//Pick one constant for COMP1
#define CP1_B0_B1      0x3000000E
#define CP1_B0_VREF    0x1000000A
#define CP1_B1_VREF    0x20000008

//Optionally OR with one or both of the following
#define CP1_OUT_ON_B2  0x04000040
#define CP1_INVERT     0x00000020
#define CP1_WAKEUP     0x00000001
#define CP1_TIMER0     0x00000010

```

The first set of three symbols defines the positive and negative inputs for the comparator; one of these must be specified. The last set of four symbols are used to select comparator options, such as inverted polarity, by ORing them into the expression passed to the ‘`setup_comparator()`’ function.

For example, to use C1IN+ (which shares its pin with RB0) as the positive input, and C1IN- (which shares its pin with RB1) as the negative input, with normal polarity:

```
setup_comparator(CP1_B0_B1);      // C1 on, C1OUT = 1 if C1IN+ > C1IN-
```

To turn off the comparator, use:

```
setup_comparator(NC_NC_NC_NC); // turn off comparators 1 and 2
```

This actually turns off both comparator modules on the PIC16F506. If `setup_comparator()` is used to configure only one of the comparators, the other is turned off. We'll see later how to configure both comparators.

To make it clear that comparator 2 is being turned off, when setting up comparator 1, you can write:

```
setup_comparator(CP1_B0_B1|NC_NC); // C1 on, C1OUT = 1 if C1IN+ > C1IN-
// (disable C2)
```

Like XC8, CCS PCB makes the C1OUT bit available as the single-bit variable ‘`C1OUT`’, so to copy the comparator output to the LED, we can use:

```
output_bit(LED,C1OUT); // turn on LED iff comparator output high
```

To invert the operation of this circuit, so that the LED turns on when the LDR is in darkness, you could copy the inverse of the comparator output to the LED, using:

```
output_bit(LED,~C1OUT); // turn on LED iff comparator output low
```

Alternatively, you could configure the comparator for inverted output, using:

```
setup_comparator(CP1_B0_B1|CP1_INVERT); // C1 on, C1OUT = 1 if C1IN+ < C1IN-
```

Complete program

Here is the complete inverted polarity version of the program, for CCS PCB:

```
*****
*      Description:    Lesson 6, example 1b
*
*      Demonstrates basic use of Comparator 1 polarity bit
*
*      Turns on LED when voltage on C1IN+ < voltage on C1IN-
*
*****
*      Pin assignments:
*          C1IN+ = voltage to be measured (e.g. pot output or LDR)
*          C1IN- = threshold voltage (set by voltage divider resistors)
*          RC3   = indicator LED
*
*****
#include <16F506.h>

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
#fuses MCLR,NOPROTECT,NOWDT,IOSC4,INTRC_IO

// Pin assignments
#define LED      PIN_C3           // indicator LED on RC3

/***** MAIN PROGRAM *****/
void main()
{
    //*** Initialisation

    // configure comparators
    setup_comparator(CP1_B0_B1|CP1_INVERT); // C1 on, C1OUT = 1 if C1IN+ < C1IN-

    //*** Main loop
    while (TRUE)
    {
        // continually display comparator output
        output_bit(LED,C1OUT);
    }
}
```

Comparisons

The following table summarises the resource usage for the “comparator 1 inverted polarity” assembler and C example programs.

Comp1_LED-neg

Assembler / Compiler	Source code (lines)	Program memory (words)	Data memory (bytes)
Microchip MPASM	20	14	0
XC8 (Free mode)	12	29	0
CCS PCB	7	28	4

The CCS version is amazingly short, at only 7 lines of code – demonstrating again that the use of built-in functions, for actions such as configuring comparators, can lead to very compact code. The XC8 version is longer, because separate statements are used to set or clear each bit in CM1CON0 – but still only around half as long as the assembler version.

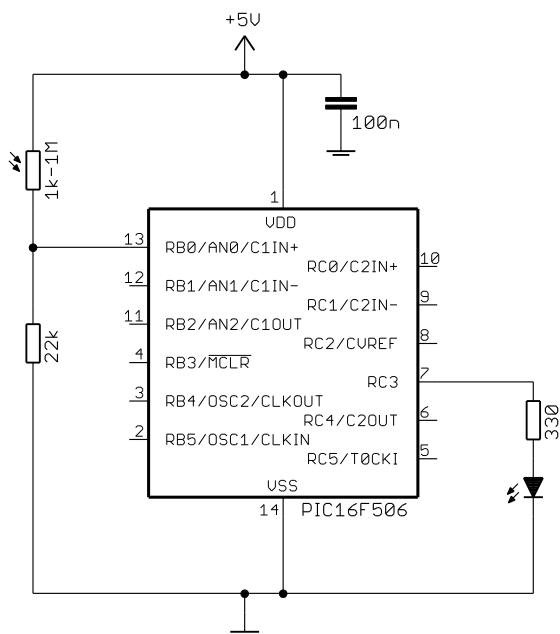
Absolute Voltage Reference

It is possible to assign an internal 0.6 V reference as the negative input for comparator 1.

This means that the external 10 kΩ resistors, forming a voltage divider in the previous example, are unnecessary, and they can be removed – as in the circuit on the right. You should find that removing the resistors makes no difference to the circuit’s operation.

It also means that the RB1 pin is now available for use.

To select the internal 0.6 V reference as the internal input, clear the C1NREF bit in the CM1CON0 register.



XC8

Assuming that we still want inverted operation, where the LED is on when the LDR is in darkness, simply change the comparator configuration instructions to:

```
// configure comparator 1
CM1CON0bits.C1PREF = 1;           // +ref is C1IN+
CM1CON0bits.C1NREF = 0;           // -ref is 0.6 V internal ref
CM1CON0bits.C1POL = 0;            // inverted polarity (C1IN+ < 0.6 V)
CM1CON0bits.C1ON = 1;             // turn comparator on
```

Or alternatively:

```
CM1CON0 = 0b00001010;           // configure comparator 1:  
//---0----  
//----1---  
//----0--  
//-----1-  
//  
// inverted polarity (C1POL = 0)  
// comparator on (C1ON = 1)  
// -ref is 0.6 V (C1NREF = 0)  
// +ref is C1IN+ (C1PREF = 1)  
// -> C1OUT = 1 if C1IN+ < 0.6 V
```

CCS PCB

To use the internal 0.6 V reference, we only need to change the parameter in the `setup_comparator()` function:

```
setup_comparator(CP1_B0_VREF|CP1_INVERT); // C1 on: C1IN+ < 0.6 V
```

This specifies that RB0 (C1IN+) be used as the positive input and the 0.6 V reference be used as the negative input on comparator 1, with the output inverted.

External Output and Hysteresis

As was explained in [baseline assembler lesson 9](#), it is often desirable to add hysteresis to a comparator, to make it less sensitive to small changes in the input signal due to superimposed noise or other interference. For example, in the above examples using a photocell, you will find that the output LED flickers when the light level is close to the threshold, particularly with mains-powered artificial illumination, which varies at 50 or 60 Hz.

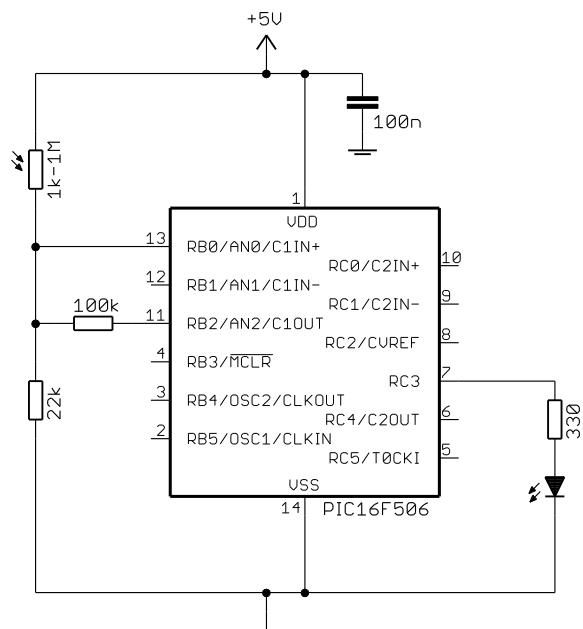
On the PIC16F506, the output of comparator 1 can be made available on the C1OUT pin.

This is done by clearing the `C1OUTEN` bit in CM1CON0.

In the circuit on the right, hysteresis has been introduced by using a 100 kΩ resistor to feed some of the comparator's output, on C1OUT, back into the C1IN+ input.

You can build this with the [Gooligum baseline training board](#) by placing the supplied 100 kΩ resistor between pins 8 ('GP/RA/RB0') and 13 ('GP/RA/RB2') on the 16-pin header.

Or, if you are using the Microchip Low Pin Count Demo Board, you would place the feedback resistor between pins 7 and 9 on the 14-pin header.



Note that, because C1OUT shares its pin with the AN2 analog input, the comparator output will not appear on C1OUT until the AN2 input is disabled; as we'll see in the [next lesson](#), a simple way to disable all the analog inputs is to clear the ADCON0 register³.

Note also that, because hysteresis relies on positive feedback, the comparator output must not be inverted.

³ See [baseline assembler lesson 10](#) for an explanation of the ADCON0 register.

XC8

To clear ADCON0 (disabling all analog inputs, including AN2, to make it possible for the output of comparator 1 to be placed on the C1OUT pin) using XC8, we can write:

```
ADCON0 = 0;                                // disable analog inputs -> C1OUT usable
```

Comparator 1 can then be configured by:

```
// configure comparator 1
CM1CON0bits.C1PREF = 1;          // +ref is C1IN+
CM1CON0bits.C1NREF = 0;          // -ref is 0.6 V internal ref
CM1CON0bits.C1POL = 1;           // normal polarity (C1IN+ > 0.6 V)
CM1CON0bits.nC1OUTEN = 0;      // enable C1OUT (for hysteresis feedback)
CM1CON0bits.C1ON = 1;            // turn comparator on
```

or:

```
CM1CON0 = 0b00101010;           // configure comparator 1:
//-----0-----          // enable C1OUT pin (/C1OUTEN = 0)
//---1-----          // normal polarity (C1POL = 1)
//----1---          // comparator on (C1ON = 1)
//----0--          // -ref is 0.6 V (C1NREF = 0)
//----1-          // +ref is C1IN+ (C1PREF = 1)
//          // -> C1OUT = 1 if C1IN+ > 0.6V,
//          // C1OUT enabled (for hysteresis feedback)
```

Note that the external output is enabled by clearing `C1OUTEN` and that the output is not inverted.

Since we want to light the LED when the photocell is in darkness ($C1IN+ < 0.6$ V), we need to invert the display logic:

```
LED = !CM1CON0bits.C1OUT;        // display comparator output (inverted)
```

CCS PCB

As we'll see in the [next lesson](#), the CCS compiler provides a '`setup_adc_ports()`' built-in function, which is used to select, among other things, whether pins are analog or digital.

It can be used to disable all the analog inputs, as follows:

```
setup_adc_ports(NO_ANALOGS);          // disable analog inputs -> C1OUT usable
```

To enable C1OUT (which shares its pin with RB2), OR the '`CP1_OUT_ON_B2`' symbol into the parameter passed to the `setup_comparator()` function:

```
setup_comparator(CP1_B0_VREF | CP1_OUT_ON_B2); // C1 on: C1IN+ > 0.6 V
//                                     // C1OUT enabled
```

Note again that, to make hysteresis possible, the comparator output bit is no longer inverted.

If we still want the LED to indicate darkness ($C1IN+ < 0.6$ V), we have to invert the display logic instead:

```
// display comparator output (inverted)
output_bit(LED, ~C1OUT);
```

Wake-up on Comparator Change

We saw in [baseline assembler lesson 9](#) that the comparator modules in the PIC16F506 can be used to wake the device from sleep when the comparator output changes – useful for conserving power while waiting for a signal change from a sensor.

To enable wake-up on change for Comparator 1, clear the `C1WU` bit in the `CM1CON0` register.

To determine whether a reset was due to a wake-up on comparator change, test the `CWUF` flag in the `STATUS` register. If `CWUF` is set, we can be sure that the device has been woken from sleep by a comparator change. If it is clear, some other type of reset has occurred.

Note that there is no indication of which comparator was the source of the reset. If you have configured both comparators for wake-up on change, you need to store the previous values of their outputs, so that you can determine which one changed.

In the example in [baseline assembler lesson 9](#), the previous circuit was used (keeping the hysteresis, making the comparator less sensitive), with the LED indicating when a comparator change occurs, by lighting for one second. While waiting for a comparator change, the PIC was placed into sleep mode – immediately after reading `CM1CON0` to prevent false triggering.

XC8

The `CWUF` flag can be tested directly, so we can simply write:

```
if (!STATUSbits.CWUF)
{
    // power-on reset
}
else
{
    // wake-up on comparator change occurred
}
```

The test is inverted here so that the normal power-on initialisation code appears first – it seems clearer that way, since you would normally look toward the start of a program to find the initialisation code.

The comparator configuration code is similar to what we've seen before, with the addition of “`CM1CON0bits.nC1WU = 0;`” to enable the wake-up on change function:

```
// configure comparator 1
CM1CON0bits.C1PREF = 1;      // +ref is C1IN+
CM1CON0bits.C1NREF = 0;      // -ref is 0.6 V internal ref
CM1CON0bits.C1POL = 1;       // normal polarity (C1IN+ > 0.6 V)
CM1CON0bits.nC1OUTEN = 0;    // enable C1OUT (for hysteresis feedback)
CM1CON0bits.nC1WU = 0;    // enable wake-up on change
CM1CON0bits.C1ON = 1;        // turn comparator on
```

Or, this could have been written as:

```
CM1CON0 = 0b00101010;        // configure comparator 1:
                            //-----0---- enable C1OUT pin (/C1OUTEN = 0)
                            //---1---- normal polarity (C1POL = 1)
                            //----1--- comparator on (C1ON = 1)
                            //----0-- -ref is 0.6 V (C1NREF = 0)
                            //----1- +ref is C1IN+ (C1PREF = 1)
//-----0                // enable wake on change (/C1WU = 0)
```

The comparator initialisation is followed by a delay of 10 ms, allowing the comparator to settle before the device is placed into sleep mode, to avoid initial false triggering.

As another (necessary) precaution to avoid false triggering, we read the current value of CM1CON0, immediately before entering sleep mode:

```
CM1CON0;                      // read comparator to clear mismatch condition
SLEEP();                       // enter sleep mode
```

Any statement which reads CM1CON0 could be used.

“CM1CON0” is an expression which evaluates to the value of the contents of CM1CON0, but does nothing.

In general, the compiler’s optimiser will discard any such “do nothing” statements.

However, CM1CON0 is declared as a ‘volatile’ variable in the processor header file. This qualifier tells the compiler that the value of this variable may change at any time, to prevent the optimiser from eliminating apparently redundant references to it. It also ensures that, when the variable’s name is used on its own in this way, the compiler will generate code which reads the variable’s memory location and discards the result, which is exactly what we want.

Complete program

Here is how the above code fragments fit together, within the complete “wake-up on comparator change demo” program:

```
*****
*                                         *
*   Description:      Lesson 6, example 2          *
*                                         *
*   Demonstrates wake-up on comparator change       *
*                                         *
*   Turns on LED for 1s when comparator 1 output changes,  *
*   then sleeps until the next change               *
*   (internal 0.6 V reference with hysteresis)     *
*                                         *
*****                                         *
*                                         *
*   Pin assignments:                           *
*       C1IN+ = voltage to be measured (e.g. pot output or LDR)  *
*       C1OUT = comparator output (fed back to input via resistor)  *
*       RC3    = indicator LED                   *
*                                         *
*****                                         /
#include <xc.h>

#define _XTAL_FREQ 4000000      // oscillator frequency for delay functions

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
__CONFIG(MCLRE_ON & CP_OFF & WDT_OFF & IOSCFS_OFF & OSC_IntRC_RB4EN);

// Pin assignments
#define LED      PORTCbits.RC3    // indicator LED on RC3
#define nLED     3                //      (port bit 3)

/***** MAIN PROGRAM *****/
void main()
{
```

```

//*** Initialisation

// configure ports
LED = 0;                                // start with LED off
TRISC = ~(1<<nLED);                    // configure LED pin (only) as an output
ADCON0 = 0;                                // disable analog inputs -> C1OUT usable

// check for wake-up on comparator change
if (!STATUSbits.CWUF)
{
    // power-on reset has occurred:

    // configure comparator 1
    CM1CON0bits.C1PREF = 1;      // +ref is C1IN+
    CM1CON0bits.C1NREF = 0;      // -ref is 0.6 V internal ref
    CM1CON0bits.C1POL = 1;       // normal polarity (C1IN+ > 0.6 V)
    CM1CON0bits.nC1OUTEN = 0;    // enable C1OUT (for hysteresis feedback)
    CM1CON0bits.nC1WU = 0;       // enable wake-up on change
    CM1CON0bits.C1ON = 1;        // turn comparator on

    // delay 10 ms to allow comparator to settle
    __delay_ms(10);
}
else
{
    // wake-up on comparator change occurred:

    // flash LED
    LED = 1;                      // turn on LED
    __delay_ms(1000);              // delay 1 sec
}

//*** Sleep until comparator change
LED = 0;                                // turn off LED
CM1CON0;                                 // read comparator to clear mismatch condition
SLEEP();                                  // enter sleep mode
}

```

CCS PCB

In [lesson 4](#), we saw that, although CCS PCB provides a ‘`restart_cause()`’ function, which returns a value indicating why the device has been reset, it does not support wake on pin change resets.

Unfortunately, this function does not support wake on comparator change resets either.

Instead, we need to test the CWUF flag, which the PCB compiler does not normally provide direct access to. The solution, as we saw in [lesson 4](#), is to use the `#bit` directive, as follows:

```
#bit CWUF = 0x03.6          // CWUF flag in STATUS register
```

This flag can then be referenced directly, in the same way as we did with XC8:

```

if (!CWUF)
{
    // power-on reset
}
else
{
    // wake-up on comparator change occurred
}

```

To configure comparator 1 for wake-up on change, OR the ‘CP1_WAKEUP’ symbol into the parameter passed to the `setup_comparator()` function:

```
setup_comparator(CP1_B0_VREF|CP1_OUT_ON_B2|CP1_WAKEUP);
```

For clarity, you may wish to split this function call across multiple lines, so that the symbols can be commented separately:

```
setup_comparator(CP1_B0_VREF|           // C1 on: C1IN+ > 0.6V,
                  CP1_OUT_ON_B2|       //
                  CP1_WAKEUP);        //      wake-up on change enabled
```

We cannot use the same method as we did with XC8 to read the current comparator output prior to entering sleep mode, because the CCS PCB compiler will not generate any instructions when an expression is not used for anything. So, to read a bit or register, we must assign it to a variable.

Since the PCB compiler exposes the C1OUT bit, we can use:

```
temp = C1OUT;           // read comparator to clear mismatch condition
sleep();                // enter sleep mode
```

Since C1OUT is a single-bit variable, the `temp` variable can be declared to be ‘int1’ (single bit), although ‘int8’ (one byte, or 8 bits) is also appropriate; the generated code size is the same for both.

Complete program

The following listing shows how these code fragments fit into the CCS PCB version of the “wake-up on comparator change demo” program:

```
*****
*   Description:    Lesson 6, example 2
*
*   Demonstrates wake-up on comparator change
*
*   Turns on LED for 1 s when comparator 1 output changes,
*   then sleeps until the next change
*   (internal 0.6 V reference with hysteresis)
*
*****
*   Pin assignments:
*       C1IN+ = voltage to be measured (e.g. pot output or LDR)
*       C1OUT = comparator output (fed back to input via resistor)
*       RC3   - indicator LED
*
*****
#include <16F506.h>
#bit CWUF = 0x03.6          // CWUF flag in STATUS register
#use delay (clock=4000000)   // oscillator frequency for delay_ms()

***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
#fuses MCLR,NOPROTECT,NOWDT,INTRC_IO,IOSC4

// Pin assignments
#define LED      PIN_C3         // indicator LED on RC3
```

```

***** MAIN PROGRAM *****/
void main()
{
    int1     temp;                      // temp variable for reading C1

    //*** Initialisation

    // configure ports
    setup_adc_ports(NO_ANALOGS);        // disable analog inputs -> C1OUT usable

    // check for wake-up on comparator change
    if (!CWUF)
    {
        // power-on reset has occurred:

        // configure comparators
        setup_comparator(CP1_B0_VREF|           // C1 on: C1IN+ > 0.6V,
                          CP1_OUT_ON_B2|         //          C1OUT pin enabled,
                          CP1_WAKEUP);         //          wake-up on change enabled

        // delay 10 ms to allow comparator to settle
        delay_ms(10);
    }
    else
    {
        // wake-up on comparator change occurred:

        // flash LED

        output_high(LED);                // turn on LED
        delay_ms(1000);                 // delay 1 sec
    }

    //*** Sleep until comparator change
    output_low(LED);                  // turn off LED
    temp = C1OUT;                    // read comparator to clear mismatch condition
    sleep();                         // enter sleep mode
}

```

Comparisons

Here is the resource usage for the “wake-up on comparator change demo” assembler and C examples.

Comp1_Wakeup

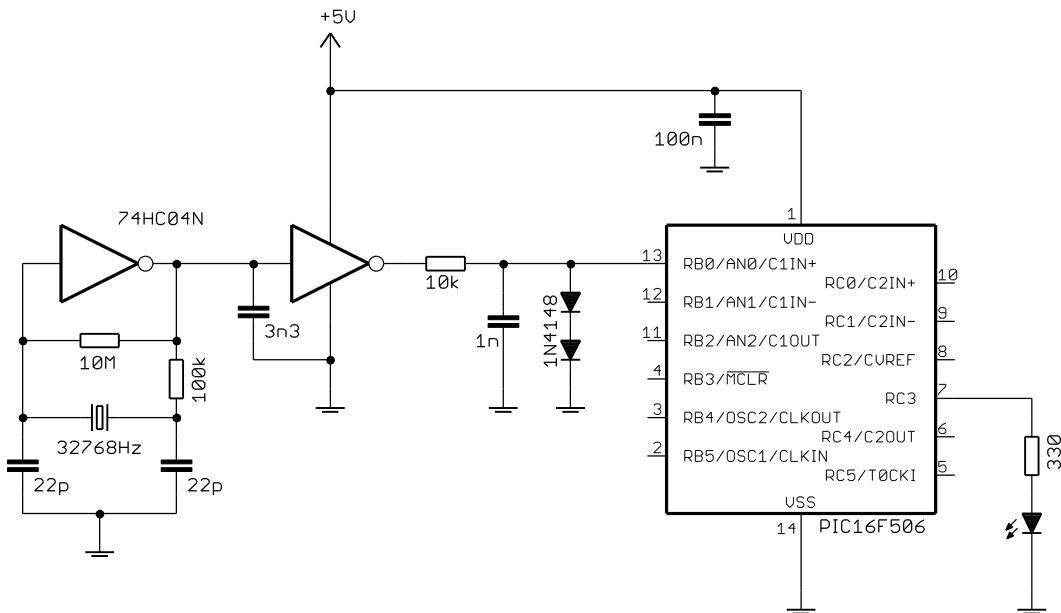
Assembler / Compiler	Source code (lines)	Program memory (words)	Data memory (bytes)
Microchip MPASM	33	43	3
XC8 (Free mode)	23	58	3
CCS PCB	17	66	7

Although the CCS source code continues to be the shortest, the CCS compiler generates the least efficient code in this example – even larger than the (unoptimised) XC8 version.

Clocking Timer0

As explained in [baseline assembler lesson 9](#), the output of comparator 1 can be used to clock Timer0. This is useful for counting pulses which do not meet the signal requirements for digital inputs (specified in the “DC Characteristics” table in the device data sheet).

To demonstrate this, the circuit below was used. The external clock module from [baseline assembler lesson 5](#) is used to supply a “clean” 32.768 kHz signal, which is degraded by being passed through a low-pass filter and clipped by two diodes, creating a signal with 32.768 kHz pulses which peak at around 1 V.



This circuit can be built with the [Gooligum baseline training board](#) and the supplied 10 k Ω resistor, 1 nF capacitor and two 1N4148 diodes. The 32.768 kHz oscillator output is available on pin 1 ('32 kHz') of the 16-pin header, the C1IN+ input is pin 8 ('GP/RA/RB0') and ground is pin16 ('GND'). You should also remove the shunt from JP24 (disconnecting the pot or photocell from C1IN+).

Note: you must only connect these additional components to C1IN+ after programming the PIC, to avoid interference with the programming process. You need to program the PIC before making the connection to C1IN+. You can then apply power (whether from a PICkit 2, PICkit3, or external power supply) and release reset – and the LED on RC3 should start flashing.

The degraded signal cannot be used to drive a digital input directly, but the clock pulses can be detected by a comparator with a 0.6 V input voltage reference.

The example program in [baseline assembler lesson 9](#) used Timer0, driven from the 32.768 kHz clock, via comparator 1, to flash the LED at 1 Hz. This was done by assigning the prescaler to Timer0, selecting a prescale ratio of 1:128, and then copying the value to TMR0<7> (which is then cycling at 1 Hz) to the LED output.

To use comparator 1 as the source for Timer0, clear the `C1T0CS` bit in the CM1CON0 register (to enable the comparator 1 timer output), and set the `T0CS` bit in the OPTION register (to select Timer0 external counter mode).

XC8

As we saw in [lesson 3](#), to configure Timer0 for counter mode, using XC8, we can use:

```
OPTION = T0CS & ~PSA | 0b110;      // configure Timer0:
                                    //   counter mode (T0CS = 1)
                                    //   prescaler assigned to Timer0 (PSA = 0)
                                    //   prescale = 128 (PS = 110)
                                    //   -> incr at 256 Hz with 32.768 kHz input
```

To configure comparator 1, with the timer output enabled, we have:

```
// configure comparator 1
CM1CON0bits.C1REF = 1;           // +ref is C1IN+
CM1CON0bits.C1NREF = 0;          // -ref is 0.6 V internal ref
CM1CON0bits.C1POL = 1;           // normal polarity (C1IN+ > 0.6 V)
CM1CON0bits.nC1T0CS = 0;       // enable TMR0 clock source
CM1CON0bits.C1ON = 1;            // turn comparator on
                                // -> C1OUT = 1 if C1IN+ > 0.6 V,
                                //     TMR0 clock from C1
```

We can then copy **TMR0<7>** to the LED output, using a shadow register, as we've done before:

```
sFLASH = (TMR0 & 1<<7) != 0;      // sFLASH = TMR0<7>
PORTC = sPORTC.port;              // copy shadow to PORTC
```

Complete program

Here is how the code for the “comparator 1 timer output demo” program fits together, using XC8:

```
*****
*
* Description:    Lesson 6, example 3
*
* Demonstrates use of comparator 1 to clock TMR0
*
* LED flashes at 1 Hz (50% duty cycle),
* with timing derived from 32.768 kHz input on C1IN+
*
*****
*
* Pin assignments:
*     C1IN+ = 32.768 kHz signal
*     RC3    = flashing LED
*
****/
```

```
#include <xc.h>
#include <stdint.h>

***** CONFIGURATION *****
// ext reset, no code protect, no watchdog, 4 MHz int clock
__CONFIG(MCLRE_ON & CP_OFF & WDT_OFF & IOSCFS_OFF & OSC_IntRC_RB4EN);

// Pin assignments
#define sFLASH  sPORTC.RC3           // flashing LED (shadow)
```

```

***** GLOBAL VARIABLES *****/
union {
    uint8_t          port;           // shadow copy of PORTC
    struct {
        unsigned     RC0      : 1;
        unsigned     RC1      : 1;
        unsigned     RC2      : 1;
        unsigned     RC3      : 1;
        unsigned     RC4      : 1;
        unsigned     RC5      : 1;
    };
} sPORTC;

***** MAIN PROGRAM *****/
void main()
{
    //*** Initialisation

    // configure ports
    TRISC = 0b110111;           // configure RC3 (only) as an output

    // configure timer
    OPTION = T0CS & ~PSA | 0b110; // configure Timer0:
                                   // counter mode (T0CS = 1)
                                   // prescaler assigned to Timer0 (PSA = 0)
                                   // prescale = 128 (PS = 110)
                                   // -> incr at 256 Hz with 32.768 kHz input

    // configure comparator 1
    CM1CON0bits.C1PREF = 1;     // +ref is C1IN+
    CM1CON0bits.C1NREF = 0;     // -ref is 0.6 V internal ref
    CM1CON0bits.C1POL = 1;      // normal polarity (C1IN+ > 0.6 V)
    CM1CON0bits.nC1T0CS = 0;    // enable TMR0 clock source
    CM1CON0bits.C1ON = 1;       // turn comparator on
                               // -> C1OUT = 1 if C1IN+ > 0.6 V,
                               //      TMR0 clock from C1

    //*** Main loop
    for (;;)
    {
        // TMR0<7> cycles at 1 Hz, so continually copy to LED
        sFLASH = (TMR0 & 1<<7) != 0; // sFLASH = TMR0<7>

        PORTC = sPORTC.port;        // copy shadow to PORTC
    }
}

```

CCS PCB

We saw in [lesson 3](#) that to configure Timer0 for counter mode, using CCS PCB, we can use:

```
setup_timer_0(RTCC_EXT_L_TO_H|RTCC_DIV_128);
```

To enable the timer output on comparator 1, we need to include the ‘CP1_TIMER0’ symbol in the parameter passed to the `setup_comparator()` function:

```
setup_comparator(CP1_B0_VREF|CP1_TIMER0);           // C1 on: C1IN+ > 0.6 V,
                                                       // TMRO clock enabled
```

Finally, bit 7 of TMRO can be copied to the LED output, using a shadow register, by:

```
sFLASH = (get_timer0() & 1<<7) != 0;      // sFLASH = TMRO<7>
output_c(sPORTC.port);                      // copy shadow to PORTC
```

Complete program

Here is the complete “comparator 1 timer output demo” program, using CCS PCB:

```
***** Description: Lesson 6, example 3 *****
* Demonstrates use of comparator 1 to clock TMRO
* LED flashes at 1 Hz (50% duty cycle),
* with timing derived from 32.768 kHz input on C1IN+
*****
* Pin assignments:
*   C1IN+ = 32.768 kHz signal
*   RC3    = flashing LED
*
****

#include <16F506.h>

***** CONFIGURATION *****
// ext reset, no code protect, no watchdog, 4 MHz int clock
#fuses MCLR,NOPROTECT,NOWDT,INTRC_IO,IOSC4

// Pin assignments
#define sFLASH sPORTC.RC3           // flashing LED (shadow)

***** GLOBAL VARIABLES *****
union {                                // shadow copy of PORTC
    unsigned int8  port;
    struct {
        unsigned  RC0     : 1;
        unsigned  RC1     : 1;
        unsigned  RC2     : 1;
        unsigned  RC3     : 1;
        unsigned  RC4     : 1;
        unsigned  RC5     : 1;
    };
} sPORTC;

***** MAIN PROGRAM *****
```

```

void main()
{
    //*** Initialisation

    // configure Timer0
    setup_timer_0(RTCC_EXT_L_TO_H|RTCC_DIV_128); // counter mode, prescale = 128
                                                    // -> increment at 256 Hz
                                                    //      with 32.768 kHz input

    // configure comparators
    setup_comparator(CP1_B0_VREF|CP1_TIMER0);     // C1 on: C1IN+ > 0.6 V,
                                                    //          TMRO clock enabled

    //*** Main loop
    while (TRUE)
    {
        // TMRO<7> cycles at 1 Hz, so continually copy to LED
        sFLASH = (get_timer0() & 1<<7) != 0;        // sFLASH = TMRO<7>

        output_c(sPORTC.port);                      // copy shadow to PORTC
    }
}

```

Comparator 2 and the Programmable Voltage Reference

As described in greater detail in [baseline assembler lesson 9](#), comparator 2 is very similar to comparator 1, except that:

- A wider range of inputs can be used as the positive reference: C2IN+, C2IN- and C1IN+
- The negative reference can be either the C2IN- pin, or an internal programmable voltage reference
- The fixed 0.6 V internal voltage reference *cannot* be used with comparator 2
- The output of comparator 2 is *not* available as an input to Timer0

Comparator 2 is controlled by the CM2CON0 register.

The programmable voltage reference can be set to one of 32 available voltages, from 0 V to $0.72 \times VDD$.

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 $CVREF = VR<3:0>/24 \times VDD$.

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

With a 5 V supply, the available output range is from 0 V to 3.59 V.

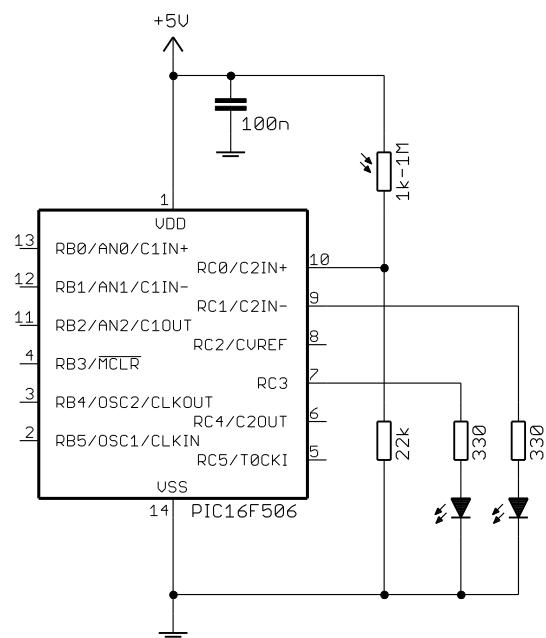
Since the low and high ranges overlap, only 29 of the 32 selectable voltages are unique ($0.250 \times VDD$, $0.500 \times VDD$ and $0.625 \times VDD$ are selectable in both ranges).

The programmable voltage reference can optionally be output on the CVREF pin, whether or not it is also being used as the negative reference for comparator 2.

In [baseline assembler lesson 9](#), the circuit on the right was used to demonstrate how comparator 2 can be used with the programmable voltage reference to test whether an input signal is within an allowed band.

The C2IN+ input is used with a photocell to detect light levels, as before. The LED on RC3 indicates a low level of illumination, and the LED on RC1 indicates bright light. When neither LED is lit, the light level will be in the middle; not too dim or too bright.

If you are using the [Gooligum baseline training board](#), you should remove the shunt from of JP24 and instead place a shunt in position 2 ('C2IN+') of JP25, connecting photocell PH2 to C2IN+. You should also place shunts in JP17 and JP19, enabling the LEDs on RC1 and RC3.



To test whether the input is within limits, the programmable voltage reference is first configured to generate the “low” threshold voltage, and the input is compared with this low level. The voltage reference is then reconfigured to generate the “high” threshold and the input is compared 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 the [next lesson](#)).

This example uses 2.0 V as the “low” threshold and 3.0 V as the “high” threshold, but, since the reference is programmable, you can always choose your own levels!

XC8

Comparator 2 can be configured in much the same way as we have been configuring comparator 1, either by assigning a binary value to CM2CON0:

```
CM2CON0 = 0b00101010;           // configure comparator 2:
//---1----          normal polarity (C2POL = 1)
//-----1-          +ref is C2IN+ (C2PREF1 = 1)
//-----0--          -ref is CVref (C2NREF = 0)
//-----1---          comparator on (C2ON = 1)
//                           -> C2OUT = 1 if C2IN+ > CVref
```

or by a block of statements assigning values to the bit-fields defined in the header files for CM2CON0:

```
// configure comparator 2
CM2CON0bits.C2PREF1 = 1;        // +ref is C2IN+
CM2CON0bits.C2NREF = 0;         // -ref is CVref
CM2CON0bits.C2POL = 1;          // normal polarity (C2IN+ > CVref)
CM2CON0bits.C2ON = 1;           // turn comparator on
//                           -> C2OUT = 1 if C2IN+ > CVref
```

Similarly, to configure the programmable voltage reference, we could assign a binary value to the variable corresponding to VRCON:

```
VRCON = 0b10000101;           // configure programmable voltage reference:
//1-----          enable voltage reference (VREN = 1)
//--0-0101          CVref = 0.406*Vdd (VRR = 0, VR = 5)
//                           -> CVref = 2.03 V
```

Or, using the bit-fields defined in the processor header files, we could write:

```
// configure voltage reference
VRCONbits.VRR = 0;           // CVref = 0.406*Vdd (2.03V) if VR = 5
                             // or 0.594*Vdd (2.97V) if VR = 11
VRCONbits.VREN = 1;          // turn voltage reference on
```

We also need to assign a value between 0 and 15 to the VR<3:0> bits, to select the reference voltage.

Although they are available as single-bit fields VR0 to VR3, they are also (much more conveniently) made available as a 4-bit field named VR within VRCONbits, allowing us to simply write:

```
VRCONbits.VR = 5;           // CVref = 0.406*Vdd (2.03V)
```

Both of these examples selected a reference of $0.406 \times VDD$, giving $CVREF = 2.03$ V with a 5 V supply.

To generate a reference voltage close to 3.0 V, we can use:

```
VRCON = 0b10001011;        // configure programmable voltage reference:
                           // enable voltage reference (VREN = 1)
                           // VR = 11
                           // CVref = 0.594*Vdd (VRR = 0, VR = 11)
                           // -> CVref = 2.97 V
```

or:

```
VRCONbits.VRR = 0;          // select high range
VRCONbits.VR = 11;          // CVref = 0.594*Vdd = 2.97 V
VRCONbits.VREN = 1;          // turn voltage reference on
```

Note that, by coincidence, the only difference between the settings for the two voltages is VR = 5 to select 2.03 V and VR = 11 to select 2.97 V.

This allows us to configure the other voltage reference settings just once, in the initialisation code:

```
// configure voltage reference
VRCONbits.VRR = 0;           // CVref = 0.406*Vdd (2.03V) if VR = 5
                             // or 0.594*Vdd (2.97V) if VR = 11
VRCONbits.VREN = 1;          // turn voltage reference on
```

Then in the main loop we can switch between the two voltages, using:

```
VRCONbits.VR = 5;           // select low CVref (2.03 V)
```

and:

```
VRCONbits.VR = 11;          // select high CVref (2.97 V)
```

After changing the voltage reference, it can take a little while for it to settle and stably generate the newly-selected voltage. According to the data sheet, for the PIC16F506 this settling time can be up to 10 μ s.

Therefore, we should insert a 10 μ s delay after selecting a new voltage, before testing the comparator output.

As we saw in [lesson 1](#), we can do this using the '`_delay_us()`' macro built into XC8.

Complete program

Here is how the above code fragments fit together, to form the complete “comparator 2 and programmable voltage reference demo” program for XC8:

```
*****  
*  
*   Description:      Lesson 6, example 4  
*  
*   Demonstrates use of comparator 2 and programmable voltage reference *  
*  
*   Turns on Low LED  when C2IN+ < 2.0 V (low light level)           *  
*           or High LED when C2IN+ > 3.0 V (high light level)          *  
*  
*****  
*  
*   Pin assignments:  
*       C2IN+ = voltage to be measured (LDR/resistor divider)  
*       RC3    = "Low" LED  
*       RC1    = "High" LED  
*  
*****  
  
#include <xc.h>  
#include <stdint.h>  
  
#define _XTAL_FREQ 4000000      // oscillator frequency for delay functions  
  
***** CONFIGURATION *****  
// ext reset, no code protect, no watchdog, 4 MHz int clock  
__CONFIG(MCLRE_ON & CP_OFF & WDT_OFF & IOSCFS_OFF & OSC_IntRC_RB4EN);  
  
// Pin assignments  
#define sLO     sPORTC.RC3          // "Low" LED (shadow)  
#define sHI     sPORTC.RC1          // "High" LED (shadow)  
  
***** GLOBAL VARIABLES *****  
union {                                // shadow copy of PORTC  
    uint8_t      port;  
    struct {  
        unsigned    RC0      : 1;  
        unsigned    RC1      : 1;  
        unsigned    RC2      : 1;  
        unsigned    RC3      : 1;  
        unsigned    RC4      : 1;  
        unsigned    RC5      : 1;  
    };  
} sPORTC;  
  
***** MAIN PROGRAM *****  
void main()  
{  
    //*** Initialisation  
  
    // configure ports  
    TRISC = 0b110101;                  // configure RC1 and RC3 (only) as outputs  
  
    // configure comparator 2  
    CM2CON0bits.C2PREF1 = 1;         // +ref is C2IN+
```

```

CM2CON0bits.C2NREF = 0;           // -ref is CVref
CM2CON0bits.C2POL = 1;           // normal polarity (C2IN+ > CVref)
CM2CON0bits.C2ON = 1;            // turn comparator on
                                // -> C2OUT = 1 if C2IN+ > CVref

// configure voltage reference
VRCONbits.VRR = 0;               // CVref = 0.406*Vdd (2.03V) if VR = 5
                                // or 0.594*Vdd (2.97V) if VR = 11
VRCONbits.VREN = 1;              // turn voltage reference on

//*** Main loop
for (;;)
{
    // Test for low illumination
    VRCONbits.VR = 5;             // select low CVref (2.03 V)
    __delay_us(10);               // wait 10 us to settle
    sLO = !CM2CON0bits.C2OUT;     // if C2IN+ < CVref turn on Low LED

    // Test for high illumination
    VRCONbits.VR = 11;            // select high CVref (2.97 V)
    __delay_us(10);               // wait 10 us to settle
    sHI = CM2CON0bits.C2OUT;      // if C2IN+ > CVref turn on High LED

    // Display test results
    PORTC = sPORTC.port;          // copy shadow to PORTC
}
}

```

CCS PCB

As was alluded to earlier, the CCS PCB built-in ‘setup_comparator()’ function is used to configure both comparators, not only comparator 1.

To configure comparator 2, OR one of the following symbols (defined in the “16F506.h” header file) into the expression passed to `setup_comparator()`:

```

#define CP2_B0_B1      0x30001C00
#define CP2_C0_B1      0x20100E00
#define CP2_C1_B1      0x20200C00
#define CP2_B0_VREF    0x10001800
#define CP2_C0_VREF    0x00100A00
#define CP2_C1_VREF    0x30200800

```

You’ll notice that there are more available input combinations than there were for comparator 1, and that for comparator 2, ‘VREF’ refers to the programmable voltage reference, instead of the 0.6 V reference.

You may also notice a mistake: **RB1** shares its pin with **C1IN-**, which is not available as an input to comparator 2. *CCS mistakenly included ‘B1’ in these symbols, when they should have written ‘C1’, referring to C2IN-, which shares its pin with RC1.*

If you do not include any of the comparator 1 configuration symbols in the expression, only comparator 2 will be setup, with comparator 1 being turned off. You can make this explicit by including the symbol ‘NC_NC’ into the expression.

For this example, we need:

```

setup_comparator(NC_NC|CP2_C0_VREF);      // C1: off
                                            // C2: C2IN+ > CVref

```

To enable one of comparator 2's options, such as inverted polarity, output on C2OUT, or wake-up on change, OR one or more of these symbols into the `setup_comparator()` expression:

```
//Optionally OR with one or both of the following
#define CP2_OUT_ON_C4 0x00084000
#define CP2_INVERT    0x00002000
#define CP2_WAKEUP    0x00000100
```

To setup the programmable voltage reference, CCS provides another built-in function: '`setup_vref()`'.

It is used in this example as follows:

```
setup_vref(VREF_HIGH | 5);           // CVref = 0.406*Vdd = 2.03 V
```

where '5' is the value the VR bits are being set to.

The first symbol is either '`VREF_LOW`' or '`VREF_HIGH`', and specifies the voltage range you wish to select. It is ORed with a number between 0 and 15, which is loaded into `VR<3:0>`, to specify the voltage.

The symbol '`VREF_A2`' can optionally be ORed into the expression, to indicate that the reference voltage should be output on the `CVREF` pin.

Finally, as in the XC8 version, we should insert a 10 μ s delay after configuring the voltage reference, to allow it to settle.

This can be done with the built-in function '`delay_us()`':

```
delay_us(10);                      // wait 10us to settle
```

Complete program

The following listing shows how these code fragments fit together in the CCS PCB version of the "comparator 2 and programmable voltage reference demo" program:

```
*****
*   Description:      Lesson 6, example 4
*
*   Demonstrates use of Comparator 2 and programmable voltage reference
*
*   Turns on Low LED  when C2IN+ < 2.0 V (low light level)
*   or High LED when C2IN+ > 3.0 V (high light level)
*
*****
*
*   Pin assignments:
*       C2IN+ = voltage to be measured (LDR/resistor divider)
*       RC3   = "Low" LED
*       RC1   = "High" LED
*
*****
```

```
#include <16F506.h>

#use delay (clock=4000000) // oscillator frequency for delay_ms()

***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
#fuses MCLR,NOPROTECT,NOWDT,INTC_IO,IOSC4
```

```

// Pin assignments
#define sLO      sPORTC.RC3           // "Low" LED (shadow)
#define sHI      sPORTC.RC1           // "High" LED (shadow)

***** GLOBAL VARIABLES *****
union {                                // shadow copy of PORTC
    unsigned int8  port;
    struct {
        unsigned   RC0      : 1;
        unsigned   RC1      : 1;
        unsigned   RC2      : 1;
        unsigned   RC3      : 1;
        unsigned   RC4      : 1;
        unsigned   RC5      : 1;
    };
} sPORTC;

***** MAIN PROGRAM *****
void main()
{
    //*** Initialisation

    // configure comparators
    setup_comparator(NC_NC|CP2_C0_VREF);    // C1: off
                                                // C2: C2IN+ > CVref

    //*** Main loop
    while (TRUE)
    {
        // Test for low illumination
        setup_vref(VREF_HIGH | 5);          // CVref = 0.406*Vdd = 2.03 V
        delay_us(10);                     // wait 10 us to settle
        sLO = ~C2OUT;                    // if C2IN+ < CVref turn on Low LED

        // Test for high illumination
        setup_vref(VREF_HIGH | 11);         // CVref = 0.594*Vdd = 2.97 V
        delay_us(10);                     // wait 10 us to settle
        sHI = C2OUT;                     // if C2IN+ > CVref turn on High LED

        // Display test results
        output_c(sPORTC.port);           // copy shadow to PORTC
    }
}

```

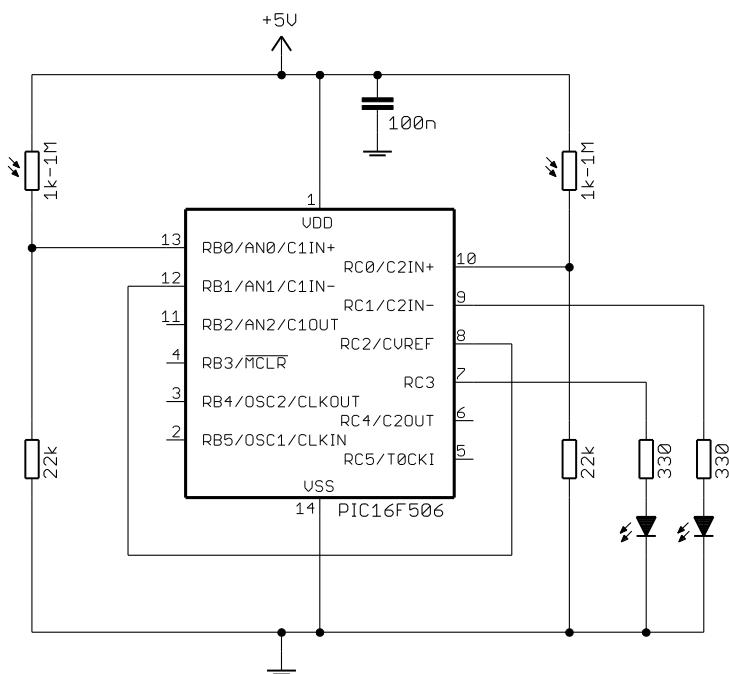
Using Both Comparators with the Programmable Voltage Reference

As 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 would need to use two comparators, with an input signal connected to each, and a single threshold voltage level connected to both.

What if we want to use the programmable voltage reference to generate the common threshold?

We've seen that CVREF cannot be selected as an input to comparator 1, so it would seem that it's not possible to use the programmable voltage reference with comparator 1.

But although no internal connection is available, that doesn't rule out an external connection – and as we saw above, the programmable reference can be made available on the CVREF pin.



So, to use the programmable voltage reference with comparator 1, we need to set the VROE bit in the VRCON register, to enable the CVREF output, and connect the CVREF pin to a comparator 1 input – as shown in the circuit on the right, where CVREF is connected to C1IN-.

If you are using the [Googum baseline training board](#), you can keep the board set up as before, with shunts in JP17, JP19, and position 2 ('C2IN+') of JP25, and add a shunt across pins 2 and 3 ('LDR1') of JP24, to also connect photocell PH1 to C1IN+. You also need to connect CVREF to C1IN-, which you can do by linking pins 9 ('GP/RA/RB1') and 11 ('RC2') on the 16-pin header.

Note: You should disconnect your PICkit 2 or PICkit 3 from the board when you run the program (applying external power instead), because the programmer loads RB1/AN1/C1IN-, pulling down the reference voltage delivered by the CVREF pin.

XC8

Most of the initialisation and main loop code is very similar to that used in earlier examples, although setting up both comparators this time, but when configuring the voltage reference, we must ensure that the VROE bit is set, so that CVREF is available externally:

```
// configure voltage reference
VRCONbits.VRR = 1; // select low range
VRCONbits.VR = 12; // CVref = 0.500*Vdd
VRCONbits.VROE = 1; // enable CVref output pin
VRCONbits.VREN = 1; // turn voltage reference on
// -> CVref = 2.50 V (if Vdd = 5 V),
// CVref output pin enabled
```

Complete program

Here is the complete XC8 version of the “two inputs with a common programmed voltage reference” program:

```
*****
*
* Description: Lesson 6, example 5
*
* Demonstrates use of comparators 1 and 2
* with the programmable voltage reference
*
* Turns on: LED 1 when C1IN+ > 2.5 V
*           and LED 2 when C2IN+ > 2.5 V
*
*****
```

```

*   Pin assignments:                                     *
*     C1IN+ = input 1 (LDR/resistor divider)          *
*     C1IN- = connected to CVref                      *
*     C2IN+ = input 2 (LDR/resistor divider)          *
*     CVref = connected to C1IN-                      *
*     RC1    = indicator LED 2                        *
*     RC3    = indicator LED 1                        *
*
***** /



#include <xc.h>
#include <stdint.h>




/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
__CONFIG(MCLRE_ON & CP_OFF & WDT_OFF & IOSCFS_OFF & OSC_IntRC_RB4EN);

// Pin assignments
#define sLED1    sPORTC.RC3      // indicator LED 1
#define sLED2    sPORTC.RC1      // indicator LED 2






/***** GLOBAL VARIABLES *****/
union {                                              // shadow copy of PORTC
    uint8_t      port;
    struct {
        unsigned    RC0      : 1;
        unsigned    RC1      : 1;
        unsigned    RC2      : 1;
        unsigned    RC3      : 1;
        unsigned    RC4      : 1;
        unsigned    RC5      : 1;
    };
} sPORTC;






/***** MAIN PROGRAM *****/
void main()
{
    //*** Initialisation

    // configure ports
    TRISC = 0b110101;                                // configure RC1 and RC3 (only) as outputs

    // configure comparator 1
    CM1CON0bits.C1PREF = 1;                          // +ref is C1IN+
    CM1CON0bits.C1NREF = 1;                          // -ref is C1IN- (= CVref)
    CM1CON0bits.C1POL = 1;                           // normal polarity (C1IN+ > C1IN-)
    CM1CON0bits.C1ON = 1;                            // turn comparator on
                                                       // -> C1OUT = 1 if C1IN+ > C1IN- (= CVref)

    // configure comparator 2
    CM2CON0bits.C2PREF1 = 1;                         // +ref is C2IN+
    CM2CON0bits.C2NREF = 0;                          // -ref is CVref
    CM2CON0bits.C2POL = 1;                           // normal polarity (C2IN+ > CVref)
    CM2CON0bits.C2ON = 1;                            // turn comparator on
                                                       // -> C2OUT = 1 if C2IN+ > CVref

    // configure voltage reference
    VRCONbits.VRR = 1;                             // select low range
}

```

```

VRCONbits.VR = 12;           // CVref = 0.500*Vdd
VRCONbits.VROE = 1;          // enable CVref output pin
VRCONbits.VREN = 1;          // turn voltage reference on
                            // -> CVref = 2.50 V (if Vdd = 5 V),
                            //      CVref output pin enabled

/** Main loop
for (;;)
{
    // start with shadow PORTC clear
    sPORTC.port = 0;

    // test comparator inputs
    sLED1 = CM1CON0bits.C1OUT; // turn on LED 1 if C1IN+ > CVref
    sLED2 = CM2CON0bits.C2OUT; // turn on LED 2 if C2IN+ > CVref

    // display test results
    PORTC = sPORTC.port;      // copy shadow to PORTC
}
}

```

CCS PCB

As we've seen, the 'setup_comparator()' function can be used to configure both comparators at once, so we can write:

```

// configure comparators
setup_comparator(CP1_B0_B1|CP2_C0_VREF); // C1: C1IN+ > C1IN-
                                            // C2: C2IN+ > CVref

```

To enable the CVREF pin when configuring the voltage reference, we need OR the symbol 'VREF_A2' into the expression passed to 'setup_vref()':

```

setup_vref(VREF_LOW | 12 | VREF_A2);           // CVref = 0.500*Vdd = 2.50 V,
                                                // CVref output pin enabled

```

Complete program

Here is the complete CCS version of the “two inputs with a common programmed voltage reference” program:

```

*****
*
* Description:    Lesson 6, example 5
*
* Demonstrates use of comparators 1 and 2
* with the programmable voltage reference
*
* Turns on: LED 1 when C1IN+ > 2.5 V
*           and LED 2 when C2IN+ > 2.5 V
*
*****
*
* Pin assignments:
*   C1IN+ = input 1 (LDR/resistor divider)
*   C1IN- = connected to CVref
*   C2IN+ = input 2 (LDR/resistor divider)
*
```

```

*      CVref = connected to C1IN-
*      RC1   = indicator LED 2
*      RC3   = indicator LED 1
*
***** /



#include <16F506.h>

***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
#fuses MCLR,NOPROTECT,NOWDT,INTRC_IO,IOSC4

// Pin assignments
#define sLED1    sPORTC.RC3          // indicator LED 1
#define sLED2    sPORTC.RC1          // indicator LED 2

***** GLOBAL VARIABLES *****/
union {                                     // shadow copy of PORTC
    unsigned int8  port;
    struct {
        unsigned  RC0      : 1;
        unsigned  RC1      : 1;
        unsigned  RC2      : 1;
        unsigned  RC3      : 1;
        unsigned  RC4      : 1;
        unsigned  RC5      : 1;
    };
} sPORTC;

***** MAIN PROGRAM *****/
void main()
{
    //*** Initialisation

    // configure comparators
    setup_comparator(CP1_B0_B1|CP2_C0_VREF);      // C1 on: C1IN+ > C1IN-
                                                    // C2 on: C2IN+ > CVref

    // configure voltage reference
    setup_vref(VREF_LOW | 12 | VREF_A2);          // CVref = 0.500*Vdd = 2.50 V,
                                                    // CVref output pin enabled

    //*** Main loop
    while (TRUE)
    {
        // start with shadow PORTC clear
        sPORTC.port = 0;

        // test comparator inputs
        sLED1 = C1OUT;                         // turn on LED 1 if C1IN+ > CVref
        sLED2 = C2OUT;                          // turn on LED 2 if C2IN+ > CVref

        // display test results
        output_c(sPORTC.port);                // copy shadow to PORTC
    }
}

```

Comparisons

Here is the resource usage summary for this example.

2xComp+VR

Assembler / Compiler	Source code (lines)	Program memory (words)	Data memory (bytes)
Microchip MPASM	29	21	1
XC8 (Free mode)	34	52	1
CCS PCB	22	41	6

The CCS source code continues to be the shortest, being 50% shorter than of the XC8 source code – mainly because of the availability of built-in functions. But although the CCS compiler generates smaller code than the XC8 compiler (running in “Free mode”, with most optimisation disabled), the CCS-generated code is nevertheless nearly twice the size of the hand-written assembler version.

Summary

We have seen that it is possible to effectively utilise the comparators and voltage references available on baseline devices, such as the PIC16F506, using either the XC8 or CCS C compilers.

As we have come to expect, source code written for the CCS compiler is consistently concise, due to the availability of built-in functions.

However, we have also seen that, at least for the version of the compiler distributed with MPLAB⁴, CCS PCB lacks support for detecting wake-up on comparator change resets and that the symbols (defined in header files) for setting up the comparator 2 inputs are misleading.

On the whole, although the XC8 source code is longer, it's more straightforward in some ways – if you're familiar with the PIC registers. The CCS compiler lets you keep more distance from the detail of the internals, which as we've seen can be better or worse, depending on the situation.

But both approaches work!

The [next lesson](#) concludes our review of the baseline PIC architecture, covering analog to digital conversion and scaling and simple filtering of ADC readings for display (revisiting material from baseline lessons [10](#) and [11](#)).

⁴ As of August, 2012