

Introduction to PIC Programming

Programming Baseline PICs in C

by David Meiklejohn, Gooligum Electronics

Lesson 5: Analog Comparators

[Baseline 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 the free HI-TECH C¹ (in “Lite” mode), PICC-Lite and CCS PCB compilers (introduced in [lesson 1](#)).

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

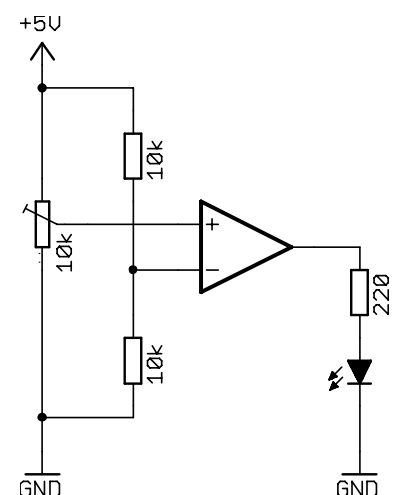
with examples for HI-TECH C, PICC-Lite and CCS PCB.

Comparators

As we saw in [baseline 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.



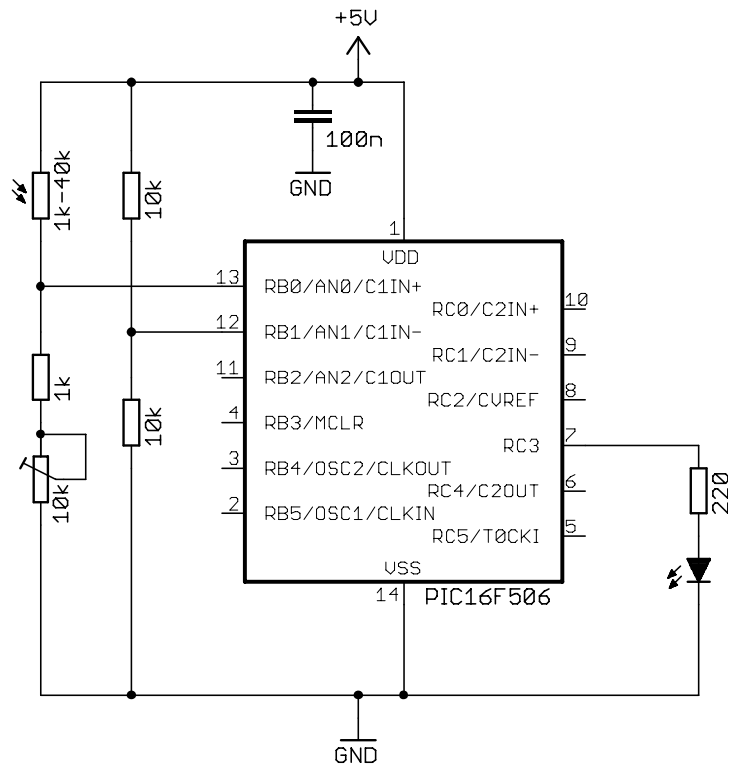
¹ PICC-Lite was bundled with versions of MPLAB up to 8.10. HI-TECH C (earlier known as “HI-TECH C PRO”) was bundled with MPLAB 8.15 and later, although you should download the latest version from www.htsoft.com.

Comparator 1

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

Note that if you do not have an LDR available, you can still explore comparator operation by connecting the C1IN+ input directly to the centre tap on the 10 kΩ potentiometer, as in the circuit diagram on the previous page.

If you are building this circuit using Microchip's Low Pin Count Demo Board, you will find that the centre tap on the 10 kΩ pot on the demo board is already connected to C1IN+ via a 1 kΩ resistor. This doesn't appreciably affect the voltage from the pot, because the comparator inputs are very high impedance. Note that if you wish to reconfigure the circuit to include an LDR, as shown, you need to remove jumper JP5 from the LPC demo board (which will involve cutting the PCB trace across it, if you have not already modified your board), to disconnect the potentiometer from the +5 V supply². Note also that the C1IN+ pin is also used for programming, and so is loaded by the PICkit 2 programmer if it is connected to the LPC demo board, slightly affecting the operation of this circuit.



We saw in [baseline lesson 9](#) that, to configure Comparator 1 to behave like the standalone comparator shown on the previous page, such that 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 operation (C1POL = 1)
; turn 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
```

² 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 $\overline{\text{MCLR}}$.

HI-TECH C PRO or PICC-Lite

We saw in [lesson 2](#) that symbols, defined in the HI-TECH C header files, can be used to represent register bits to construct a value to load into the OPTION register, for example:

```
OPTION = ~T0CS & ~PSA | 0b1111;
```

However, unlike the OPTION bits, the HI-TECH C header files define most register bits, including those in CM1CON0, as single-bit variables, and not as symbols which can be combined in arithmetic or logical expressions.

This means that if you wish to set and/or clear a number of bits in a register such as CM1CON0 at once, you must use a numeric constant such as:

```
CM1CON0 = 0b00101110;           // configure comparator 1:
//--1-----                   normal polarity (C1POL = 1)
//----1----                    turn 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 HI-TECH C, is to use a sequence of assignments, to set or clear the appropriate register bits. For example:

```
// configure comparator 1
C1PREF = 1;           // +ref is C1IN+
C1NREF = 1;           // -ref is C1IN-
C1POL = 1;           // normal polarity (C1IN+ > C1IN-)
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 instances, 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 variable 'C1OUT'. For example:

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

With the above configuration, the LED will turn on when the LDR is illuminated (you may need to adjust the pot to set the threshold appropriately for your ambient lighting).

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

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

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

or:

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

Finally, as explained in [baseline lesson 9](#), the internal RC oscillator on the PIC16F506 can run at either 4 MHz or 8 MHz, selectable by the IOSCFS bit in the processor configuration word.

To select 8 MHz operation using HI-TECH C, add the 'INTOSC8' symbol to the `__CONFIG()` macro.

For example:

```
// ext reset, no code protect, no watchdog, 8 MHz int clock
__CONFIG(MCLREN & UNPROTECT & WDTDIS & INTOSC8 & INTIO);
```

Complete program

Here is the complete inverted polarity version of the program, for HI-TECH C:

```
*****
*
* Description:    Lesson 5, 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 <htc.h>

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 8 MHz int clock
__CONFIG(MCLREN & UNPROTECT & WDTDIS & INTOSC8 & INTIO);

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

```

/***** MAIN PROGRAM *****/
void main()
{
    // Initialisation
    TRISC = ~(1<<nLED);           // configure LED pin (only) as an output

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

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

```

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);    // configure C1: 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); // configure C1: C1IN+ > C1IN-
// (disable C2)
```

Like PICC-Lite, 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 either:

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

or:

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

[Unlike HI-TECH PICC-Lite, CCS PCB allows the '~' operator to be used with bit variables.]

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

```
setup_comparator(CP1_B0_B1|CP1_INVERT); // configure C1: C1IN+ < C1IN-
```

Finally, if you wish to operate the internal RC oscillator at 8 MHz, add the 'IOSC8' symbol to the #fuses directive (not that there is really any reason to do so, in this example):

```
// ext reset, no code protect, no watchdog, 8MHz int clock
#fuses MCLR,NOPROTECT,NOWDT,INTRC_IO,IOSC8
```

Complete program

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

```
*****
*
*   Description:    Lesson 5, 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, 8MHz int clock
#fuses MCLR,NOPROTECT,NOWDT,INTRC_IO,IOSC8

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

```

/***** MAIN PROGRAM *****/
void main()
{
    // Initialisation
    setup_comparator(CP1_B0_B1|CP1_INVERT); // configure C1: C1IN+ < C1IN-

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

```

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	18	13	0
HI-TECH PICC-Lite	12	18	4
HI-TECH C PRO Lite	12	24	2
CCS PCB	7	28	4

The HI-TECH source code is longer, because separate statements are used to set or clear each bit in CM1CON0. Nevertheless, even the un-optimised HI-TECH C PRO compiler (running in ‘Lite’ mode) generates smaller code than the CCS compiler, with the PICC-Lite version being nearly as small as the hand-written assembler version.

Absolute Voltage Reference

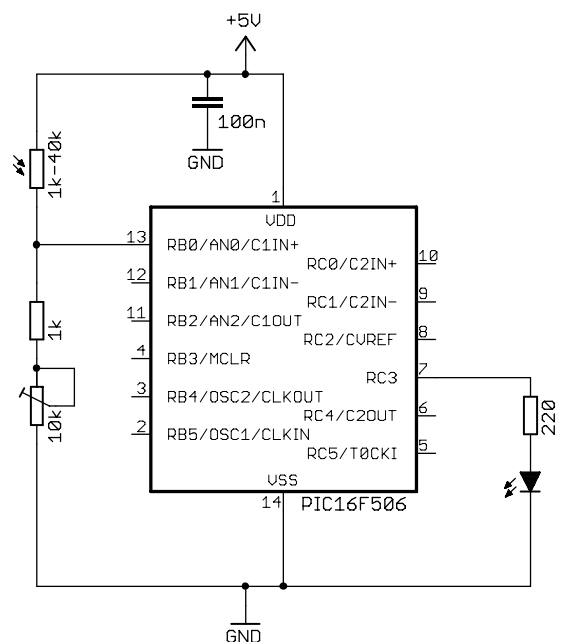
It is possible to assign an internal 0.6 V reference as the negative input for comparator 1.

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.

It also means that the RB1 pin is now available for I/O.

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

Since the voltage threshold is now 0.6 V instead of 2.5 V, the potentiometer has to be adjusted to compensate, so that light level changes will continue to affect the comparator output.



HI-TECH C PRO or PICC-Lite

To clear ADCON0 (disabling all analog inputs³, including AN2, to make it possible for the output of comparator 1 to be placed on the C1OUT pin) using PICC-Lite, we can write:

```
ADCON0 = 0; // disable analog inputs
```

Comparator 1 can then be configured by:

```
// configure comparator 1
C1PREF = 1; // +ref is C1IN+
C1NREF = 0; // -ref is 0.6V internal ref
C1POL = 1; // normal polarity (C1IN+ > 0.6V)
C1OUTEN = 0; // enable C1OUT (for hysteresis feedback)
C1ON = 1; // turn comparator on
```

or:

```
CM1CON0 = 0b00101010; // configure comparator 1:
// -0----- enable C1OUT pin (*C1OUTEN = 0)
// --1----- normal polarity (C1POL = 1)
// ----1--- turn 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 output is not inverted, and that the external output is enabled by clearing $\overline{\text{C1OUTEN}}$.

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
```

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); // configure C1:
// C1IN+ > 0.6V, C1OUT enabled
```

Comparisons

Here is the resource usage for the “comparator 1 external output” assembler and C example programs.

Comp1_LED-intref_hyst

Assembler / Compiler	Source code (lines)	Program memory (words)	Data memory (bytes)
Microchip MPASM	19	14	0
HI-TECH PICC-Lite	14	20	4
HI-TECH C PRO Lite	14	30	2
CCS PCB	8	31	4

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

Each program is a little longer than before, because of the additional instructions needed to disable the analog inputs. But otherwise the patterns remain the same; the trade-off between source code length and implementation efficiency remains obvious.

Wake-up on Comparator Change

We saw in [baseline lesson 9](#) that the comparator modules in the PIC16F509 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 $\overline{C1WU}$ bit in the CM1CON0 register.

To determine whether a reset was due to a wake-up on comparator change, test that 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 wake-up on change 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 lesson 9](#), the previous circuit was used (with no hysteresis, so the 22 kΩ resistor can be omitted), 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.

HI-TECH PICC-Lite

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

```
if (!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 “C1WU = 0;” to enable the wake-up on change function, and “C1OUTEN = 1;” to explicitly disable the external output (since it is disabled by default, it's not strictly necessary to include it here – again, it just seems clearer):

```
// configure comparator 1
C1PREF = 1;           // +ref is C1IN+
C1NREF = 0;           // -ref is 0.6V internal ref
C1POL = 1;           // normal polarity (C1IN+ > 0.6V)
C1OUTEN = 1;         // disable C1OUT pin
C1WU = 0;            // enable wake-up on change
C1ON = 1;            // turn comparator on
```

Or, this could have been written as:

```
CM1CON0 = 0b01101010; // configure comparator 1:
// -1-----          disable C1OUT pin (*C1OUTEN = 1)
// --1-----          normal polarity (C1POL = 1)
// ----1---           turn 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. There is a risk that the compiler’s optimiser will decide that, because this statement appears to do nothing, no instructions need to be generated for it. But in this case, both HI-TECH C compilers generate a “`movf CM1CON0, w`” instruction, which reads `CM1CON0`; exactly what we want.

Finally, since this code is to be run at 4 MHz (matching the ‘`XTAL_FREQ`’ definition, used by the delay routines), the ‘`INTOSC4`’ symbol is included in the `__CONFIG()` macro, instead of ‘`INTOSC8`’.

Complete program

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

```

/*****
*
*   Description:      Lesson 5, 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, no hysteresis)
*
*****/
*
*   Pin assignments:
*       CLIN+ = voltage to be measured (e.g. pot output or LDR)
*       RC3   = indicator LED
*
*****/

#include <htc.h>

#define XTAL_FREQ    4MHZ    // oscillator frequency for delay functions
#include "stdmacros-PCL.h"  // DelayS() - delay in seconds
                           // DelayMs() - delay in ms

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
__CONFIG(MCLREN & UNPROTECT & WDTDIS & INTIO & INTOSC4);

// Pin assignments
#define LED          RC3      // display LED
#define nLED        3        // (port bit 3)

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

```

```

// initialise ports
LED = 0; // start with LED off
TRISC = ~(1<<nLED); // configure LED pin (only) as an output

// check for wake-up on comparator change
if (!CWUF)
{
    // power-on reset has occurred:
    // configure comparator 1
    C1PREF = 1; // +ref is C1IN+
    C1NREF = 0; // -ref is 0.6V internal ref
    C1POL = 1; // normal polarity (C1IN+ > 0.6V)
    C1OUTEN = 1; // no C1OUT
    C1WU = 0; // enable wake-up on change
    C1ON = 1; // turn comparator on
    // delay 10 ms to allow comparator to settle
    DelayMs(10);
}
else
{
    // wake-up on comparator change occurred:
    // turn on LED for 1 second
    LED = 1;
    DelayS(1);
}

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

```

HI-TECH C PRO

Because HI-TECH C PRO provides built-in delay functions, which are a little different from the example delay functions provides with PICC-Lite, we need to change the oscillator frequency definition from:

```
#define XTAL_FREQ 4MHZ // oscillator frequency for delay functions
```

to:

```
#define _XTAL_FREQ 4000000 // oscillator frequency for delay functions
```

and change the 100 ms delay from 'DelayMs(10)' to '__delay_ms(10)'.

We also need to include the appropriate header file, defining the 'DelayS()' macro (from [lesson 2](#)):

```
#include "stdmacros-HTC.h" // DelayS() - delay in seconds
```

instead of:

```
#include "stdmacros-PCL.h" // DelayS() - delay in seconds
```

so that the appropriate millisecond delay function will be called.

CCS PCB

In [lesson 3](#), 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 13, 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 as we did with HI-TECH C:

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

To configure comparator 1 for wake-up on change, we can use:

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

We cannot use the same method as we did with HI-TECH C 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 expose 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 “short” (single bit), although “char” (one byte, or 8 bits) is also appropriate – the generated code size is the same for both.

Finally, since this code is to be run at 4 MHz, the ‘IOSC4’ symbol is included in the #fuses directive, instead of ‘IOSC8’.

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 5, 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, no hysteresis)
*
*****/
*
*   Pin assignments:
*       C1IN+ = voltage to be measured (e.g. pot output or LDR)
*       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()
{
    short    temp;              // temp variable for reading C1

    // check for wake-up on comparator change
    if (!CWUF)
    {
        // power-on reset has occurred:
        // configure Comparator 1: C1IN+ > 0.6V, no C1OUT, wake-up on change
        setup_comparator(CP1_B0_VREF|CP1_WAKEUP);
        // delay 10 ms to allow comparator to settle
        delay_ms(10);
    }
    else
    {
        // wake-up on comparator change occurred:
        // turn on LED for 1 second
        output_high(LED);
        delay_ms(1000);
    }

    // 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	30	41	3
HI-TECH PICC-Lite	22	77	10
HI-TECH C PRO Lite	22	63	4
CCS PCB	16	63	7

In this case, not only is the CCS source code the shortest, but the CCS compiler generates more efficient code than PICC-Lite – mainly because the built-in delay functions available in HI-TECH C PRO and CCS PCB are more efficient than the example delay code provided with PICC-Lite.

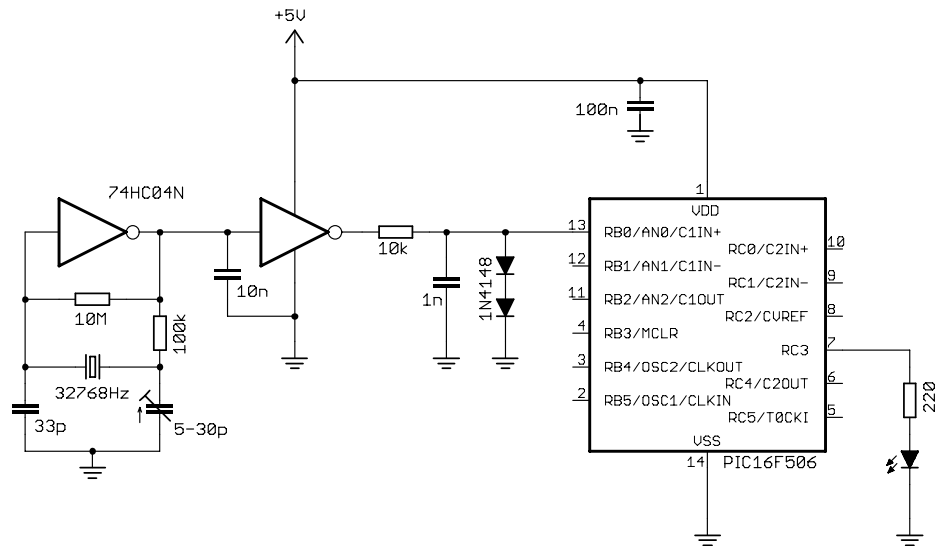
Nevertheless, both C compilers generate code around 50% larger than the assembler version, while the assembler source is almost twice as long as the CCS source code.

Clocking Timer0

As explained in [baseline 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 on the right was used. The external clock module from [baseline 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 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 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 $\overline{C1T0CS}$ 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).

HI-TECH C PRO or PICC-Lite

As we have seen before, to configure Timer0 for counter mode, using HI-TECH C, we can use:

```
OPTION = T0CS | 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, you could write:

```
// configure comparator 1
C1PREF = 1;                       // +ref is C1IN+
C1NREF = 0;                       // -ref is 0.6V internal ref
C1POL = 1;                       // normal polarity (C1IN+ > 0.6V)
C1T0CS = 0;                       // enable TMR0 clock source
C1ON = 1;                         // turn comparator on
```

We can then copy TMR0<7> to the LED output, with:

```
LED = (TMR0 & 1<<7) ? 1 : 0;
```

Complete program

Here is how the code for the “comparator 1 timer output demo” program fits together, using HI-TECH C:

```

/*****
*
* Description: Lesson 5, 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 <htc.h>

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
__CONFIG(MCLREN & UNPROTECT & WDTDIS & INTIO & INTOSC4);

// Pin assignments
#define LED RC3 // flashing 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 timer
    OPTION = T0CS | 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
    C1PREF = 1; // +ref is C1IN+
    C1NREF = 0; // -ref is 0.6V internal ref
    C1POL = 1; // noninverted output (C1IN+ > 0.6V)
    C1T0CS = 0; // enable TMR0 clock source
    C1ON = 1; // turn comparator on

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

```

CCS PCB

We saw in [lesson 2](#) 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, you need to include the 'CP1_TIMER0' symbol in the parameter passed to the setup_comparator() function:

```
// configure Comparator 1: C1IN+ > 0.6V, TMR out enabled
setup_comparator(CP1_B0_VREF|CP1_TIMER0);
```

Finally, bit 7 of TMR0 can be copied to the LED output by:

```
output_bit(LED, get_timer0() & 1<<7);
```

Complete program

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

```

/*****
*
* Description: Lesson 5, 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 <16F506.h>

/***** 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 // flashing LED on RC3

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

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

    // configure Comparator 1: C1IN+ > 0.6 V, TMR out enabled
    setup_comparator(CP1_B0_VREF|CP1_TIMER0);

```

```

// Main loop
while (TRUE)
{
    // TMR0<7> cycles at 1Hz, so continually copy to LED
    output_bit(LED, get_timer0() & 1<<7);
}

```

Comparisons

Here is the resource usage summary for the “comparator 1 timer output demo” assembler and C examples.

Comp1_TMR

Assembler / Compiler	Source code (lines)	Program memory (words)	Data memory (bytes)
Microchip MPASM	23	17	1
HI-TECH PICC-Lite	14	21	4
HI-TECH C PRO Lite	14	27	2
CCS PCB	8	33	5

The CCS source code is impressively short, at only eight lines long, although the program code generated by the CCS compiler is by far the largest – nearly twice the size of the assembler program. In comparison, the PICC-Lite compiler generated code which is only slightly larger than the hand-written assembler version.

Comparator 2 and the Programmable Voltage Reference

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

- 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 $0.72 \times V_{DD}$.

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 $CV_{REF} = VR<3:0>/24 \times V_{DD}$.

VRR = 0 selects the high range, where $CV_{REF} = V_{DD}/4 + VR<3:0>/32 \times V_{DD}$.

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 ($0.250 \times V_{DD}$, $0.500 \times V_{DD}$ and $0.625 \times V_{DD}$ 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 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.

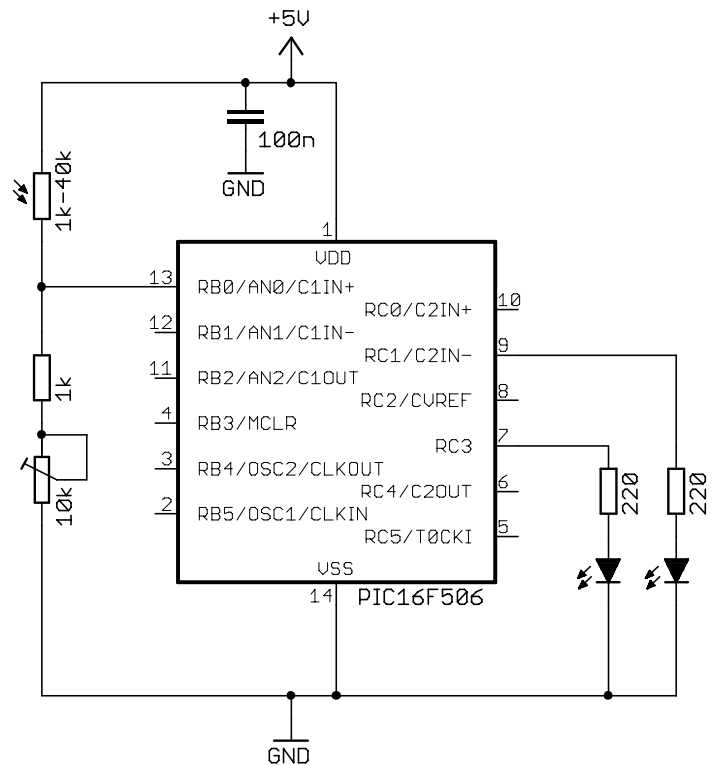
The C1IN+ input is used with an LDR to detect light levels, as before.

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.

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 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!



HI-TECH C PRO or PICC-Lite

Comparator 2 can be configured, using HI-TECH C, by a block of statements assigning values to each bit in CM2CON0, in much the same way as we have been configuring comparator 1:

```
// configure Comparator 2
C2PREF1 = 0;           // +ref is C1IN+
C2PREF2 = 1;
C2NREF = 0;           // -ref is CVref
C2POL = 1;           // normal polarity (C1IN+ > CVref)
C2ON = 1;           // turn comparator on
```

Or you could simply assign a value to CM2CON0:

```
CM2CON0 = 0b00111000; // configure comparator 2:
//--1----- // normal polarity (C2POL = 1)
//---1--0- // +ref is C1IN+ (C2PREF1 = 0, C2PREF2 = 1)
//-----0-- // -ref is CVref (C2NREF = 0)
//----1--- // turn comparator on (C2ON = 1)
// -> C2OUT = 1 if C1IN+ > CVref,
```

HI-TECH C makes the individual control bits in the VRCON register available as variables, so we can write, for example:

```
VRR = 1;           // select low range
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.

Since these are the least-significant four bits of VRCON, we need to ensure that, when we assign a value to these bits, the most-significant four bits are not affected. This can be done by *masking*, as follows:

```
VRCON = VRCON & 0xF0 | 5; // VR = 5 -> CVref = 0.208*Vdd = 1.04 V
```

By ANDing VRCON with F0h (or 11110000 in binary), the upper four bits are preserved, while the lower four bits are cleared. A 4-bit value can then be ORed with the result, assigning that value to the lower four bits. This type of expression is commonly used to assign a value to a *bit field* within a register.

Alternatively, you may prefer to simply assign a value to VRCON:

```
VRCON = 0b10100101; // configure programmable voltage reference:
//1----- enable voltage reference (VREN = 1)
//--1-0101 CVref = 0.208*Vdd (VRR = 1, VR = 5)
// -> CVref = 1.04 V
```

Both of these examples selected a reference of $0.208 \times VDD$, giving $CVREF = 1.04 \text{ V}$ with a 5 V supply.

To generate a reference voltage close to 2.0 V, we can use:

```
VRR = 0;           // select high range
VRCON = VRCON & 0xF0 | 5; // VR = 5 -> CVref = 0.406*Vdd = 2.03 V
VREN = 1;         // turn voltage reference on
```

or:

```
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
```

Note that, by coincidence, the only difference between the settings for the two voltages is $VRR = 1$ to select 1.04 V and $VRR = 0$ to select 2.03 V.

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

```
// configure voltage reference
VRCON = VRCON & 0xF0 | 5; // VR = 5 -> CVref = 0.208*Vdd (1.04V) if VRR = 1
//                               or 0.406*Vdd (2.03V) if VRR = 0
VREN = 1; // turn voltage reference on
```

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

```
VRR = 1; // select low CVref (1.04 V)
```

and:

```
VRR = 0; // select high CVref (2.03 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 μs .

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 'DelayUs ()' macro defined in the "delay.h" header file supplied with PICC-Lite, or the '___delay_us ()' macro built into HI-TECH C PRO.

Complete program

Here is how the above code fragments fit together, to form the complete "comparator 2 and programmable voltage reference demo" program for PICC-Lite:

```

/*****
*
* Description: Lesson 5, example 4
*
* Demonstrates use of Comparator 2 and programmable voltage reference
*
* Turns on Low LED when C1IN+ < 1.0 V (low light level)
* or High LED when C1IN+ > 2.0 V (high light level)
*
*****/
*
* Pin assignments:
* C1IN+ - voltage to be measured (LDR/resistor divider)
* RC3 - "Low" LED
* RC1 - "High" LED
*
*****/

#include <htc.h>

#define XTAL_FREQ 4MHZ // oscillator frequency for delay functions
#include "delay.h" // DelayUs() - delay in us

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
___CONFIG(MCLRREN & UNPROTECT & WDTDIS & INTOSC4 & INTIO);

// Pin assignments
#define LO RC3 // "Low" LED
#define nLO 3 // (port bit 3)

#define HI RC1 // "High" LED
#define nHI 1 // (port bit 1)

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

    // configure ports
    TRISC = ~(1<<nLO|1<<nHI); // configure PORTC LED pins as outputs

    // configure Comparator 2
    C2PREF1 = 0; // +ref is C1IN+
    C2PREF2 = 1;
    C2NREF = 0; // -ref is CVref
    C2POL = 1; // normal polarity (C1IN+ > CVref)
    C2ON = 1; // turn comparator on
}

```

```

// configure voltage reference
VRCON = VRCON & 0xF0 | 5; // VR = 5 -> CVref = 0.208*Vdd (1.04V) if VRR = 1
//                                     or 0.406*Vdd (2.03V) if VRR = 0
VREN = 1; // turn voltage reference on

// Main loop
for (;;)
{
    // Test for low illumination
    VRR = 1; // select low CVref (1.04 V)
    DelayUs(10); // delay to allow CVref to settle
    LO = !C2OUT; // display inverse of comparator output
                // (C1IN+ < CVref) on Low LED

    // Test for high illumination
    VRR = 0; // select high CVref (2.03 V)
    DelayUs(10); // delay to allow CVref to settle
    HI = C2OUT; // display comparator output
               // (C1IN+ > CVref) on High LED
}
}

```

CCS PCB

As was alluded to above, 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_B0_VREF); // configure C2: C1IN+ > CVref
// (disable C1)

```

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:

```
// set low input threshold
setup_vref(VREF_LOW | 5);           // CVref = 0.208*Vdd = 1.04 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 HI-TECH C 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, to form the complete CCS PCB version of the "comparator 2 and programmable voltage reference demo" program:

```
*****
*
*   Description:      Lesson 5, example 4
*
*   Demonstrates use of Comparator 2 and programmable voltage reference
*
*   Turns on Low LED when C1IN+ < 1.0 V (low light level)
*                   or High LED when C1IN+ > 2.0 V (high light level)
*
*****
*
*   Pin assignments:
*   C1IN+ - voltage to be measured (LDR/resistor divider)
*   RC3   - "Low" LED
*   RC1   - "High" LED
*
*****/

#include <16F506.h>

#define _XTAL_FREQ 4000000 // oscillator frequency for delay_ms()

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
#define MCLR 1
#define NOPROTECT 1
#define NOWDT 1
#define INTRC_IO 1
#define IOSC4 1

// Pin assignments
#define LO PIN_C3 // "Low" LED
#define HI PIN_C1 // "High" LED

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

```

{
  // Initialisation

  // configure comparators: C1 disabled, C2 = C1IN+ > CVref
  setup_comparator(NC_NC|CP2_B0_VREF);

  // Main loop
  while (TRUE)
  {
    // Test for low illumination
    // set low input threshold
    setup_vref(VREF_LOW | 5);          // CVref = 0.208*Vdd = 1.04 V
    delay_us(10);                      // wait 10us to settle
    // turn on Low LED if C1IN+ < CVref
    output_bit(LO,~C2OUT);             // display inverse of C2 output

    // Test for high illumination
    // set high input threshold
    setup_vref(VREF_HIGH | 5);         // CVref = 0.406*Vdd = 2.03 V
    delay_us(10);                      // wait 10us to settle
    // turn on Low LED if C1IN+ > CVref
    output_bit(HI,C2OUT);              // display C2 output
  }
}

```

Comparisons

Here is the resource usage summary for the “comparator 2 and programmable voltage reference demo” assembler and C examples.

Comp2_2LEDs

Assembler / Compiler	Source code (lines)	Program memory (words)	Data memory (bytes)
Microchip MPASM	36	30	1
HI-TECH PICC-Lite	24	53	5
HI-TECH C PRO Lite	23	54	3
CCS PCB	14	48	4

Once again, the CCS source code is impressively short, at only fourteen lines long – little more than half the length of the HI-TECH source code. But all of the C compilers generate significantly larger code than the hand-written assembler version, in this example.

Using Both Comparators with the Programmable Voltage Reference

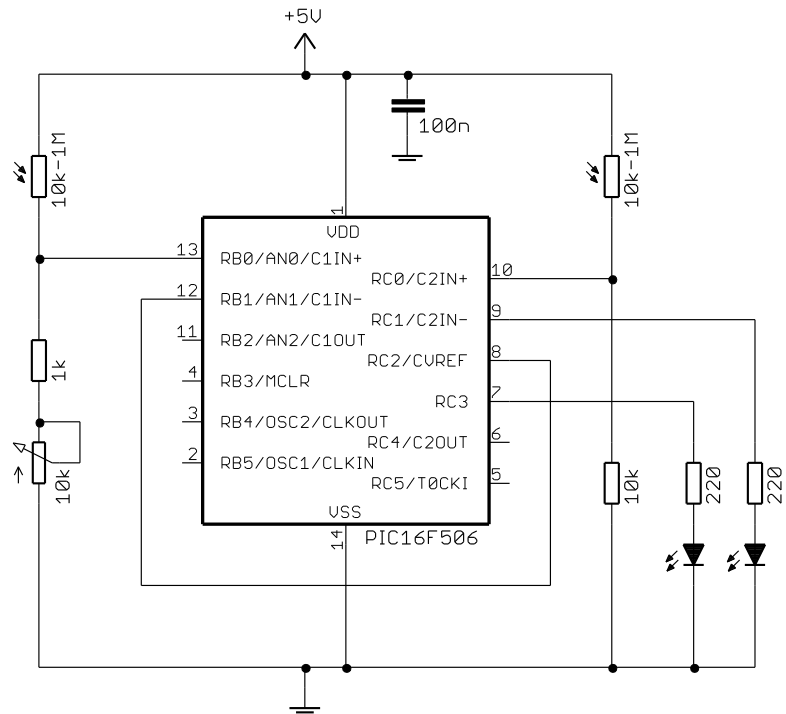
As a final example, suppose that we want to test two input signals (say, light level in two locations) by comparing them against a common reference. We would need to use two comparators, with an input signal connected to each, and a single threshold voltage level connected to both.

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-.



HI-TECH C PRO or PICC-Lite

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:

```
VRR = 1; // select low range
VRCON = VRCON & 0xF0 | 5; // VR = 5 -> CVref = 0.208*Vdd = 1.04 V
VROE = 1; // enable CVref output pin
VREN = 1; // turn voltage reference on
```

Complete program

Here is the complete HI-TECH C version of the “two inputs with a common programmed voltage reference” program:

```

/*****
* Description: Lesson 5, example 5
*
* Demonstrates use of comparators 1 and 2
* with the programmable voltage reference
*
* Turns on: LED 1 when C1IN+ > 1.0 V
* and LED 2 when C2IN+ > 1.0 V
*
*****
* Pin assignments:
* C1IN+ - input 1 (LDR/resistor divider)
* C1IN- - connected to CVref
* C2IN+ - input 2 (LDR/resistor divider)
* RC1 - indicator LED 1
* RC3 - indicator LED 2
*****/

```

```

#include <htc.h>

/***** CONFIGURATION *****/
// ext reset, no code protect, no watchdog, 4 MHz int clock
__CONFIG(MCLREN & UNPROTECT & WDTDIS & INTOSC4 & INTIO);

// Pin assignments
#define LED1    RC1           // indicator LED 1
#define nLED1   1            // (port bit 1)

#define LED2    RC3           // indicator LED 2
#define nLED2   3            // (port bit 3)

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

    // configure ports
    TRISC = ~(1<<nLED1|1<<nLED2); // configure PORTC LED pins as outputs

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

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

    // configure voltage reference
    VRR = 1;                   // select low range
    VRCON = VRCON & 0xF0 | 5; // VR = 5 -> CVref = 0.208*Vdd = 1.04 V
    VROE = 1;                  // enable CVref output pin
    VREN = 1;                   // turn voltage reference on

    // Main loop
    for (;;)
    {
        // display comparator outputs
        LED1 = C1OUT;           // turn on LED 1 if C1IN+ > CVref
        LED2 = C2OUT;           // turn on LED 2 if C2IN+ > CVref
    }
}

```

CCS PCB

As we saw above, 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 | 5 | VREF_A2); // CVref = 0.208*Vdd = 1.04 V,
// enable CVref output pin
```

Complete program

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

```

/*****
*   Description:      Lesson 5, example 5
*
*   Demonstrates use of comparators 1 and 2
*   with the programmable voltage reference
*
*   Turns on: LED 1 when C1IN+ > 1.0 V
*               and LED 2 when C2IN+ > 1.0 V
*
*****/
*   Pin assignments:
*       C1IN+ - input 1 (LDR/resistor divider)
*       C1IN- - connected to CVref
*       C2IN+ - input 2 (LDR/resistor divider)
*       RC1  - indicator LED 1
*       RC3  - indicator LED 2
*****/

#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 LED1    PIN_C1           // indicator LED 1
#define LED2    PIN_C3           // indicator LED 2

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

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

    // configure voltage reference
    setup_vref(VREF_LOW | 5 | VREF_A2); // CVref = 0.208*Vdd = 1.04 V,
                                        // enable CVref output pin

    // Main loop
    while (TRUE)
    {
        // display comparator outputs
        output_bit(LED1,C1OUT); // turn on LED 1 if C1IN+ > CVref
        output_bit(LED2,C2OUT); // turn on LED 2 if C2IN+ > CVref
    }
}

```

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	27	20	1
HI-TECH PICC-Lite	23	35	4
HI-TECH C PRO Lite	23	45	2
CCS PCB	10	45	5

The CCS source code continues to be impressively short, being than half the length of the HI-TECH source code – mainly because of the availability of built-in functions. But although PICC-Lite continues to generate the smallest code of the C compilers, it is still more than 50% larger than 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 HI-TECH or CCS C compilers.

As we have come to expect, source code written for the CCS compiler is consistently the shortest (as little as one third the length of the corresponding assembler source), due to the availability of built-in functions:

Source code (lines)

Assembler / Compiler	Ex 1b	Ex 1d	Ex 2	Ex 3	Ex 4	Ex 5
Microchip MPASM	18	19	30	23	36	27
HI-TECH PICC-Lite	12	14	22	14	24	23
HI-TECH C PRO Lite	12	14	22	14	23	23
CCS PCB	7	8	16	8	14	10

However, we have also seen that, at present, CCS support for comparator 2 (misleading symbols in header files) is a little “rough”.

Once again, all three C compilers generate code which is significantly larger than the corresponding hand-written assembler versions, with the CCS-generated code being up to twice the size of the assembler version:

Program memory (words)

Assembler / Compiler	Ex 1b	Ex 1d	Ex 2	Ex 3	Ex 4	Ex 5
Microchip MPASM	13	14	41	17	30	20
HI-TECH PICC-Lite	18	20	77	21	53	35
HI-TECH C PRO Lite	24	30	63	27	54	45
CCS PCB	28	31	63	33	48	45

Data memory (bytes)

Assembler / Compiler	Ex 1b	Ex 1d	Ex 2	Ex 3	Ex 4	Ex 5
Microchip MPASM	0	0	3	1	1	1
HI-TECH PICC-Lite	4	4	10	4	5	4
HI-TECH C PRO Lite	2	2	4	2	3	2
CCS PCB	4	4	7	5	4	5

The trade-off remains clear – programming effort (loosely correlated with source code length) versus efficient resource utilisation (program and data memory usage). There continues to be a place for both C and assembler...

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](#)).