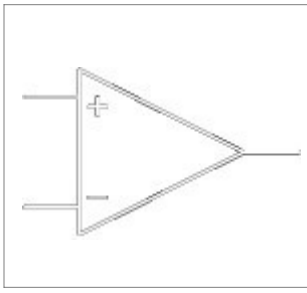
# Knowledge of baseline

## Comparator and Vref

**The object of the exercise is the knowledge of the characteristics and use of the Analog Comparator module and the reference voltage generation module.**

## Comparator

One of the function modules found on several PICs is the Analog Comparator. It should be noted that it is an analog comparator, i.e. it receives variable voltages at the inputs in the range between Vss and Vdd.
This is a common function in electronics.



The symbol, similar to that of an op-amp, indicates the presence of three terminals:

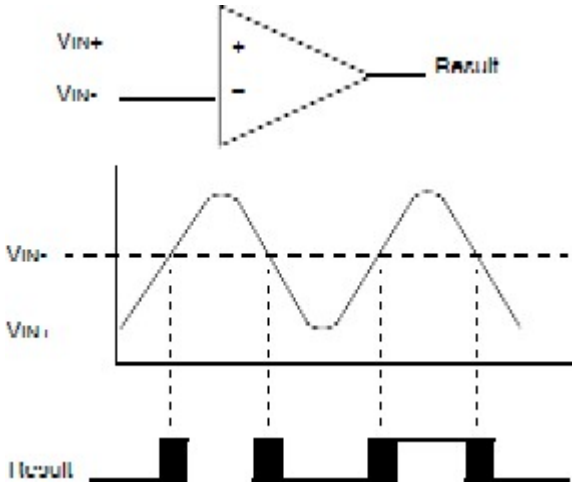
1. Non-inverting input vin+
2. Vin-inverting input
3. Vout Output

Reduced to the essentials, it works as follows:

if ***Vin+ > Vin- → Vout = H***

if ***Vin+ < Vin- → Vout = L***

That is, the output goes from low to high and vice versa depending on the level of voltages applied to the inputs.




The comparator compares the two voltages applied at the input and makes an output at two levels, high and low, compatible with the digital ones, depending on the outcome of the comparison. Basically, we have a digital response to an analog variation, which allows this element to be introduced into the data that the microcontroller can process.

For example, the image on the side exemplifies an application of the comparator: a variable signal is connected to one input, while a reference voltage is applied to the other. Whenever the first signal exceeds the reference value, the output goes to a high level, indicating the condition.

The circuit can also be realized by swapping the inputs and, consequently, the output condition. The uses of this function are many: discrimination of a signal with respect to a reference, over- or under-voltage detector, triggers for measurements of physical quantities, and so on.
Another possible application is to adapt to the digital input of the microcontroller a signal that has characteristics different from the expected TTL ones (ref. Tutorial 6). For example, a signal that is very slow in going from 0 to Vdd: outside the recognition thresholds accepted by the input pin, the applied voltage would give random output responses that would make the program's response to the input state equally random. If we apply this signal not to a digital pin, but to the input of the comparator and set a suitable trigger threshold, we will have a correct "digital" response to the analog variation.

## ICPs and Comparators

The comparator has numerous other possible applications, and its introduction as a function module in microcontrollers is quite common. As far as Baseline CIPs are concerned, we have this situation:

| PIC | Pin | Comp. |
|---|---|---|
| **10F204** | 6 | 1 |
| **10F206** | | 1 |
| **12F510** | 8 | 1 |
| **16F506** | 14 | 2 |
| **16F526** | | 2 |
| **16F527** | 20 | 2 |
| **16F570** | 28 | 2 |

For chips with a low pin count, this is a single unit. Chips with multiple pins have a pair of comparators.
The comparator terminals share other functions on some pins and are named by Microchip:

- **CxIN+** : Analog Input Vin+
- **CxIN-** : Analog Input Vin-
- **CxOUT** : Vout Digital Output

where **x** is the comparator referred to (so **C1IN+** will be the non-inverting input of comparator 1 and C2IN+ will be the non-inverting input of comparator 2). It should be noted that, in order to have the same labels in the treatment of Comparator1 for both single and double comparator chips, Microchip calls Comparator1 even though it is the only one available. This is not a bad idea !

**The Comparator is not a complicated function in itself, but it is from the point of view of control**, as the dedicated register, **CMxCON0**, allows a considerable range of possible configurations, which must be understood in order to effectively exploit the module. This requires a pretty extensive description.

The bits that make up the register are these:

| R-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 |
|---|---|---|---|---|---|---|---|
| C1OUT | $\overline{\text{C1OUTEN}}$ | C1POL | $\overline{\text{C1T0CS}}$ | C1ON | C1NREF | C1PREF | $\overline{\text{C1WU}}$ |
| bit 7 | | | | | | | bit 0 |

You will have **CM1CON0** if there is only one comparator and **CM1CON0** / **CM2CON0** if there are two. The registers are completely identical. Let's take a look at the individual actions of the various bits:

**Bit 7 C1OUT** is a flag that indicates the status of the comparator output.

| C1OUT | Function |
|---|---|
| 1 | **Vin+ > Vin-** |
| 0 | **Vin+ < Vin-** |

It should be considered that the comparator has a physical output that can be connected to the **C1OUT** pin, but it can work very well even if the output is not enabled (which leaves one pin free). So you need a flag that indicates the condition to allow the program to verify it.

**6 bit C1OUTEN** enables **C1OUT output** on the relative pin

| C1OUTEN | Function |
|---|---|
| 1 | **disabled - C1OUT pin available** |
| 0 | **output enabled on pin C1OUT** |

Allows you to have the comparator output available, if enabled; Otherwise, the PIN is released to the other shared functions. This is the default condition, so if we don't need the comparator output, we don't have to perform any maneuvers on this bit.

**5-bit C1POL** switches comparator output polarity

| C1POL | Function |
|---|---|
| 1 | **Not reversed** |
| 0 | **Reversed** |

This allows the output level to be reversed without physically swapping the Vin+ and Vin- pins with each other.

**Bit 4 C1T0CS** allows the comparator to be used as the input signal of the Timer0

| C1T0CS | Function |
|---|---|
| 1 | **the source is chosen by the T0CS bit** |
| 0 | **The source is the output of the comparator** |

With this bit we can connect the output of the comparator to the counting input of the Timer0, without making any external physical connection. The function is useful when you want to count pulses from external sensors.

**Bit 3 C1ON** turns the comparator on and off.

| C1ON | Function |
|---|---|
| 1 | **qualified** |
| 0 | **disabled** |

At the ROP, the comparator is enabled; If you don't use it, you need to disable it, both to reduce current consumption and to be able to access the other PIN functions.

**Bit 2 C1NREF** determines what the Vin- input is connected to.

| C1NREF | Function |
|---|---|
| 1 | **pin C1IN-** |
| 0 | **Internal Vref** |

It should be noted that the comparator, by default, is enabled and therefore engages the corresponding pins, as well as consuming a certain current. The bit allows you to turn off the

comparator if it is not in use.

**Bit 1 C1PREF** determines what the Vin+ input is connected to.

| C1PREF | Function |
|---|---|
| 1 | Pin C1IN+ |
| 0 | pin C1IN- |

This bit allows the connections to the inverting and non-inverting input pins to be swapped with each other, without modifying the external wiring.

**Bit 0 C1WU** controls wakeup for change of status of the comparator output.

| C1WU | Function |
|---|---|
| 1 | Wake up disabled |
| 0 | Wake up enabled |

By enabling the function, you will have an output (weake up) from the sleep condition as a consequence of the change in the status of the comparator output. Note that the output does not need to be connected to pin **C1OUT,** as it is checked internally.

By default, all bits are at 1; Then, if the audit log is not changed, the comparator is configured like this:

- Output Disabled
- Polarity not reversed
- Timer0 input chosen by T0CS
- Enabled Comparator
- Vin- = CIN-
- Vin+ = CIN+
- WakeUp Disabled

So, the comparator, by default, is enabled and engages the corresponding pins, as well as consuming a certain current. When the application does not use it, you will have to disable the comparator to have other functions on the pins.

It can be seen how, through the control register, the comparator can be configured in the most varied way, excluding the use of external links and increasing the possibilities of use.
The general logic of operation is that a reference voltage is applied to Vin-; in particular, bit 2 allows this input to be connected with an internal voltage source (Vref), which avoids the need for external references. This does not take away the possibility of using them or even using Vin+ for reference, reversing the state of the output. The bit1 allows you to swap the function of the two input pins, while the C1POL bit allows you to reverse the output, so you have maximum flexibility.

It should be noted that **the response of the comparator to changes in the setting of the control register is not immediate**, therefore, by varying the bits of this register during the execution of the program, it is useful to insert a wait of about ten milliseconds before considering the operational comparator in the new conditions set.

# A note for PIC10F204/6.

We have mentioned that Microchip has chosen to call 1 both the single comparator of the 12F510 and the channel 1 of the double comparator of the 16F and this allows the same labels to be used.
Unfortunately, for the 10F we are at the usual, fanciful differentiation from the "standard", so the control register looks like this (sic!):

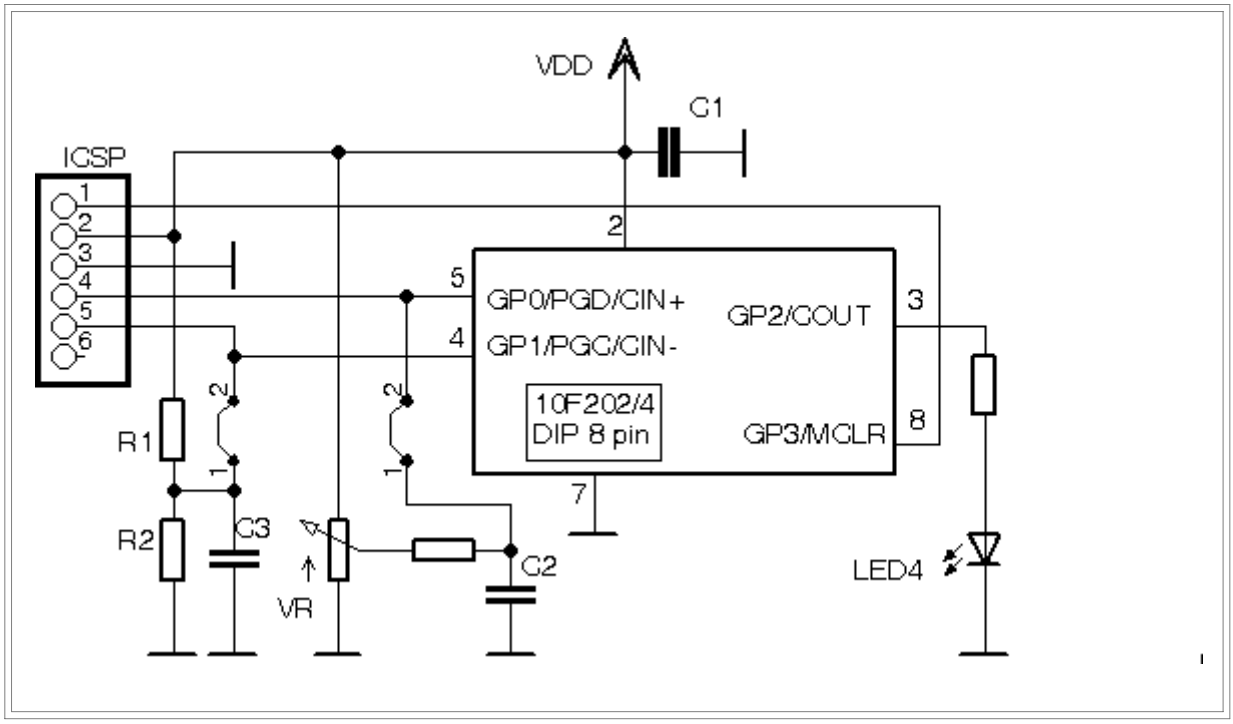
| R-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 |
|---|---|---|---|---|---|---|---|
| CMPOUT | $\overline{\text{COUTEN}}$ | POL | $\overline{\text{CMPT0CS}}$ | CMPON | CNREF | CPREF | $\overline{\text{CWU}}$ |
| bit 7 | | | | | | | bit 0 |

In any case, the bit functions are the same as those seen above. Based

on what has been said, we can see some practical applications.

# A basic application for the comparator

We can see the comparator at work with a **10F204/6**. Let's make this circuit:



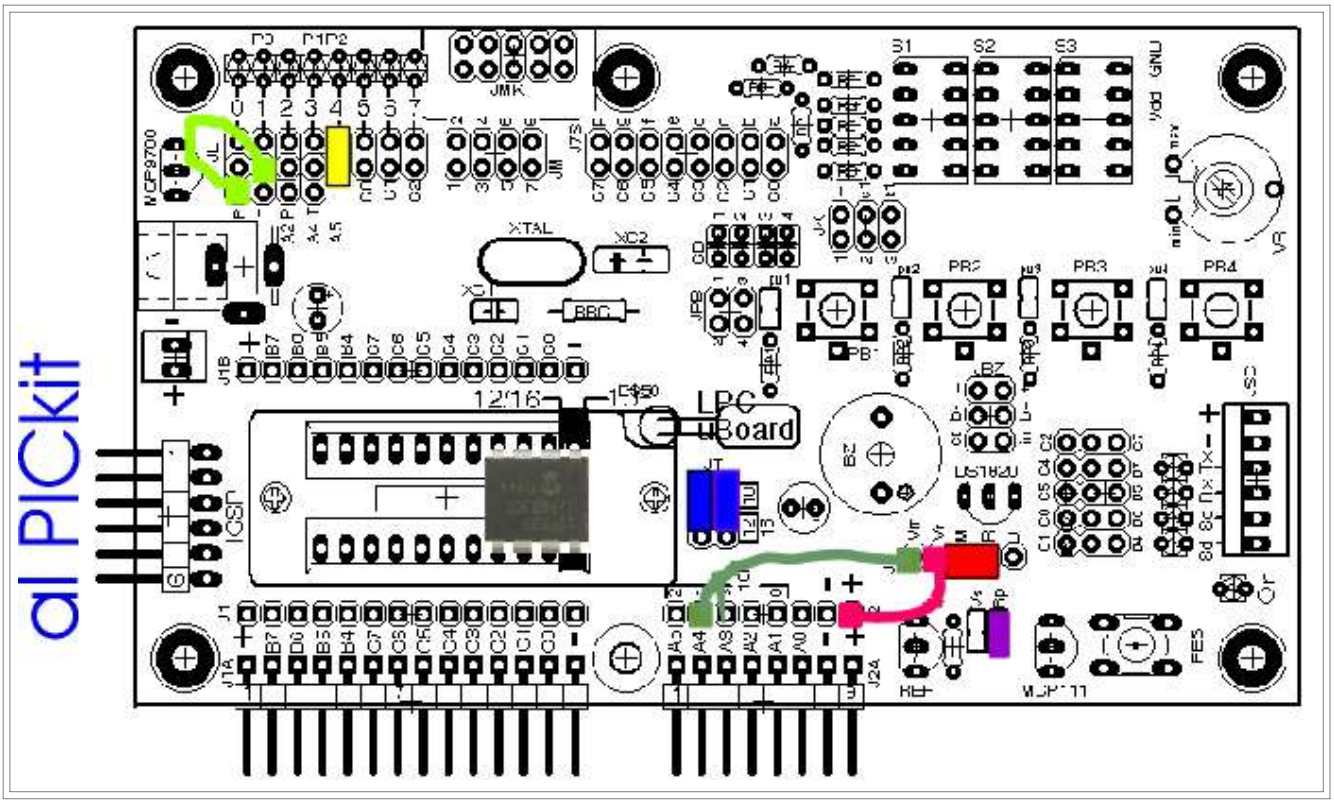

The VR potentiometer slider is connected to the **CIN+** input; the voltage varies between Vdd and Vss. The **CIN-** input is connected to a divider, where ***R1*** = ***R2***; therefore the voltage applied as a reference is half of the Vdd. On the LPCuB there are two 5k resistors, but these are not values
Critical; They are just enough to be equal to determine a point at half tension. However, different values can also be used if different reference voltages are required.


**Note the jumpers that connect to GP0 and GP1:** since these pins are also those of the programming connection, **the jumpers must be removed** during this phase, so that the resistors do not prevent it, improperly loading the serial signals.
They will be reconnected after programming the chip.

An LED is connected to **GP2** to indicate switching.
Capacitors ***C2*** and ***C3*** (100nF) are present on the LPCuB with the purpose of stabilizing voltages against small disturbance variations induced by neighboring circuits. They are not indispensable in a breadboard construction.

The operation is the basic one: by turning the potentiometer slider, we apply a variable voltage between 0 and 5V to the CIN+ input. As long as this voltage is lower than the 2.5V reference, the LED will be off and will light up when the threshold is exceeded.

The connections on the LPCuB are, as usual, very simple:




The "red" flying jumper connects the upper end of the divider (provided with the use of the reference voltage produced by the integrated amplifier that can be installed on the VREF socket) to the Vdd.
The "dark green" flying jumper connects the center point of the divider to the CIN-/GP1. The "dark vrede" flying jumper connects the VR slider with CIN+/GP0.
These two jumpers must be disconnected when programming the chip.
The "yellow" jumper connects GP2 to LED4.
The Reset jumpers are indifferent, since the button is not used.

## The program

We have an initial choice only between **10F204** and **10F206**, the only 10F2xx to have a comparator, after which we initialize the I/O:

```
; I/O initializations on reset

; GP2 as output
movlw b'11111011'TRIS
            GPIO

; disable T0CKI to have free GP2
; OPTION def '11111111'
;             1------- GPWU disabled
;             -1------ GPPU disabled
;             --0----- clock internal
;             ---1 ----done
;             ----1--- prescaler al timer
;             -----111 1:256 AM
```

```
     movlw b'11011111'
     OPTION


; Enable comparator, with no output and with normal inputs
; no wakeup, no Timer input
; CMCON def '11111111'
;            1 -------cmpout
;            -1------ Output disabled
;            --1----- Normal polarity
;            ---1---- T0CS per Timer0 in
;            ----1--- Enable Comparator
;            -----1-- vref- = CIN-
;            ------1- vref+ = CIN+
;            -------1 no wakeup
    movlw  b'11111111'
    movwf  CMCON0
```

We observe that the output of the comparator is disabled: **GP2** will control the LED through the program instructions.
Now we can enter the comparator status test and control the LED accordingly:

```
; Comparator Test
clp btfsc CMCON0, CMPOUT
     bsf LED
    btfss CMCON0, CMPOUT
     bcf LED
    Goto CLP             ; indefinite loop
```

The source is in the ***13A_10F204.asm file***.


**Once the chip is programmed, we disconnect the Pickit**, since the DAT and CLK lines of ICSP will be affected by analog signals.

We reconnect the jumpers that had to be removed, give voltage and, turning the knob of the potentiometer we will see the LED turn on and off depending on the position of the cursor.
We can, with a tester, check the voltages at pins GP0 and GP1.
If we program the **POL bit** to **COMCON0** to 0, the output will be reversed.

# The Independent Comparator

We can now make an interesting test. Let's leave the hardware as it is and make a tweak to the software:

```
; Enable comparator, with no output and with normal inputs
; no wakeup, no Timer input
; CMCON def '11111111'
;             1------------ cmpout
;              -0------ Output Enabled
;              --1----- Normal polarity
;             ---1---- T0CS per Timer0 in
;             ----1--- Enable Comparator
;             -----1-- vref- = CIN-
;             ------1- vref+ = CIN+
;             -------1 no wakeup
    movlw   b'10111111'
    movwf   CMCON0

    goto    $           ; Execution Block
```

After enabling **the output of the comparator**, we immediately block the execution with a loop on itself. The processor does nothing else.
Having programmed the chip with the previous precautions, **we turn the knob of the potentiometer and we will see the LED turn on and off as before !**

This is easy to explain: the comparator is indeed an element integrated into the microcontroller, but it depends on it only for the settings derived from **CMCON0;** for its operation it does not use anything else from the processor, not even the primary clock (so much so that it can also work in sleep).
Then, **once we have assigned inputs and outputs to physical pins, the processor does not interfere with the comparator until the bits of the control register are redefined**. Since we have enabled the output on the **GP2 pin**, the LED will now depend not on the digital function of **GP2**, which is controlled by the program, but directly on **the COUT** of the comparator.

 Note that it is not necessary to change the **GP2** assignment in **the TRIS,** since the setting to enable the output of the comparator overrides it.

This information can be found in the component data sheet and, clearly, in Table 5.1:

**TABLE 5-1: ORDER OF PRECEDENCE FOR PIN FUNCTIONS**

| Priority | GP0 | GP1 | GP2 | GP3 |
|---|---|---|---|---|
| 1 | CIN+ | CIN- | FOSC4 | I/MCLR |
| 2 | TRIS GPIO | TRIS GPIO | COUT | — |
| 3 | — | — | T0CKI | — |
| 4 | — | — | TRIS GPIO | — |

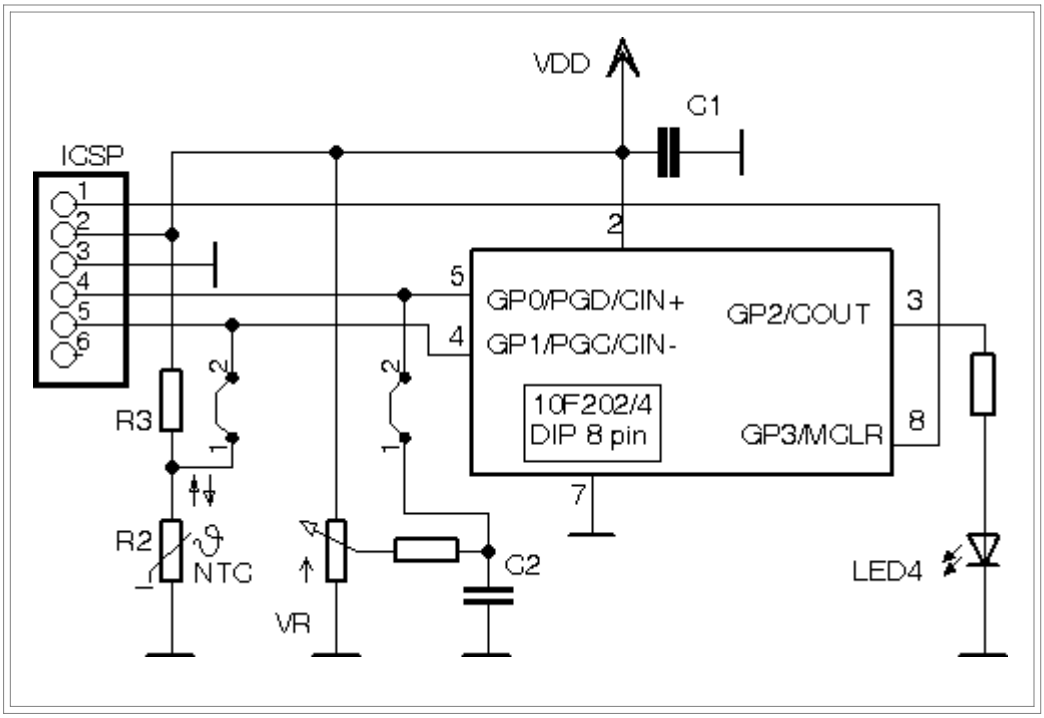
From the table we can see that 4 different functions contribute to the pin, whose priority is:

- **FOSC4**
- **COUT**
- **T0CKI**
- **TRIS GPIO**

As a result, setting the comparator output as active takes precedence over any other setting except that of **FOSC4** (regulated by the 0 bit of **OSCCAL).**

# Comparator & Sensors

Variable voltage can be generated by other sources other than a potentiometer, such as temperature, brightness, pressure sensors, etc.
On our LPCuB we can insert a thermistor in the diagram above, i.e. a resistor whose value varies with temperature:



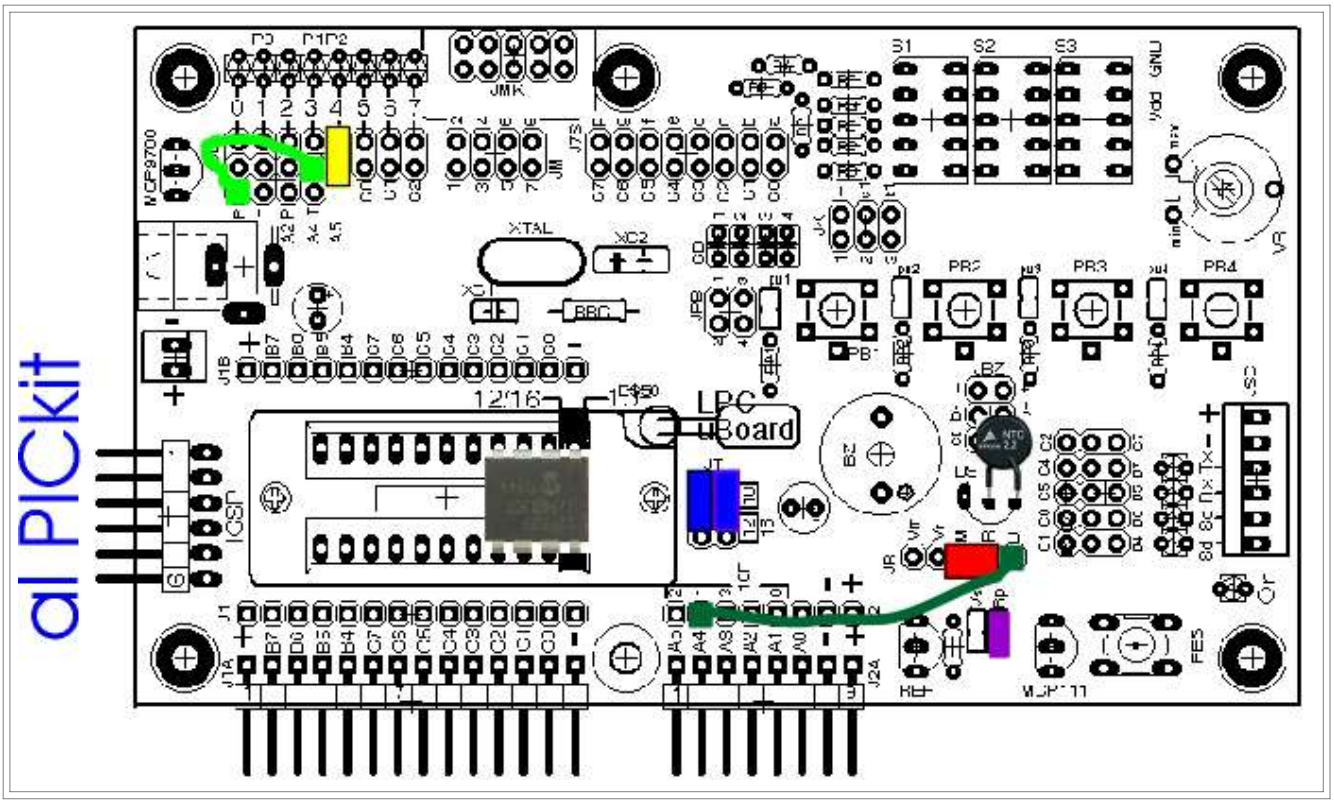

The **CIN-** input is connected to the junction point between a pull-up resistor R3 and an NTC to ground; the voltage at that point will vary with the value of the NTC, which depends on the temperature.
The **CIN+ input** is connected to the potentiometer.

On LPCuB, R3 is 4k7, so R2 will be a common 4k7 or 10k NTC. As a result, if we use a 4k7 @20°C NTC, the voltage on CIN- will be about half of the Vdd. If we insert a 10k one we will have about 3.3V. This voltage will be measurable between the Vss and any of the pins connected to GP0.

Turn the knob of the potentiometer so that we have a few millivolts less: the LED will be off for the reference relative to the temperature of the NTC at that moment.
In fact, on CIN- the voltage will depend on the temperature: when this rises, the NTC value drops and therefore the voltage drops. When this voltage is higher than that fixed with the potentiometer, Vin+>Vin- and then Vout=H. If we heat the NTC, even just holding it in our hands, when Vin- is less than Vin+, the LED will turn off.
This is how we created a thermostat with a variable

threshold. On the LPCuB, the connections are always

simple:




The NTC is inserted on the right pins of the socket indicated in silkscreen as DS1820. The "dark green" flying jumper connects the NTC with the GP1/CIN-
The "light green" flying jumper connects the potentiometer slider with GP0/CIN+.
The "yellow" jumper connects the GP2 with the LED4.

We keep the same program as before, and, as before:


1. **We disconnect the green jumpers while programming the chip**.

2. **Once the chip is programmed, we disconnect the Pickit**, since the DAT and CLK lines of ICSP will be affected by analog signals.

We reconnect the jumpers that had to be removed, turn on the power and, heating the NTC, we will see the LED light up. When it returns to room temperature, the LED will turn off. By reversing the POL bit, the action of the LED is reversed. You will be able to apply the same variations as before.

We can use any other resistive element that varies with the variation of a physical quantity, for example a photo resistance. In this case, the fact that the resistance decreases as the brightness increases, will be exploited. The resistance photo will be inserted as in the previous example and the program will be the same. Obviously, instead of the LED there could be a relay to control a load depending on the temperature or brightness.

# Other ICPs

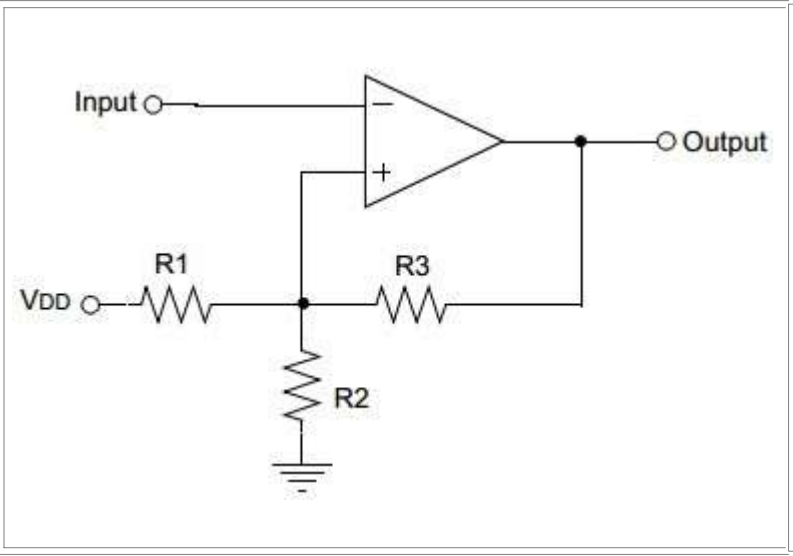
The same experiments can be carried out with the same program also for the other Baselines equipped with a comparator, using channel 1 and taking care to vary the labels relating to the control register that becomes **CM1CON0** and identifying the pins corresponding to C1IN+ / C1IN- / C1OUT.

# The reference voltage

We have said that, by means of the **CNREF bit** (**C1NREF**) we can connect it to an internal voltage, produced by a precision band-gap generator with 0.6V output.
This voltage, bringing the control bit to 0, becomes the reference of the comparator and is valid for the entire range of Vdd admissible for the operation of the chip (2-5.5V).

# Hysteresis

When the input voltages of the comparator have values very close together, it is enough for a small disturbance to be superimposed on one of them to produce undesirable switching. To avoid this, you need to insert a reaction, adding some hysteresis to the switching. If this is greater than the magnitude of the disturbance, it will have no effect. This results in a similar behaviour to the Schmitt trigger.




It is a matter of inserting a resistor that brings a part of the output voltage back to the positive input.

In order to calculate the values of the resistances, it is necessary to undertake a certain number of calculations, which require a certain knowledge of electricity and its basic laws.

However, we present the subject both as an exercise and because it is possible that the opportunity arises to

To calculate the value of the resistances, we first determine the maximum and minimum threshold voltage (***Vth*** / ***Vtl***) to obtain:

$$Vavg = (Vdd*Vtl) / ( Vdd - Vth + Vtl)$$

which, for Thevenin, becomes:

$$Vavg = (Vdd*R2) / [ Vdd (R1+R2)] = Vdd (R2 / (R1+R2))$$

It must be taken into account that a resistor equivalent to R1+R2 is located in parallel between Vdd and Vss and therefore consumes a certain current and dissipates a certain power, which is not appreciated by battery-powered systems. To limit this, it is advisable that the two resistors have a high value. However, at the same time, there should be a maximum of ten k to limit the value of R3, which cannot become of the order of many hundreds of kohms, otherwise offset voltages will be produced due to the comparator bias currents (which, although small, exist).
The equivalent resistance of R1 R2 is:

$$Req = (R1\ R2) / (R1+R2)$$

while the reaction ratio is given by:

$$Dr = (Vth - Vtl) / Vdd$$

From these two equations we can calculate R3:

$$R3 = Req\ [(1 / Dr) -1]$$

For example, with a Vdd of 5V, we want to discriminate signals below 2.5V (Vtl) and above 3V (Vth); All intermediate voltages must be excluded.
We:

$$Dr = (Vth - Vtl) / Vdd = (3 - 2.5) / 5 = 0.1$$

We can choose two common values for R1 and R2: R1= 8k2 and R2=10, which result in a Vavg of 2.75V, in the middle of the band to be rejected.
We:

$$Req = (R1\ R2) / (R1+R2) = (8.2 * 10) / (8.2 + 10) = 4k5$$

Whereby:

$$R3 = Req\ [(1 / Dr) -1] = 4k5\ ((1/0.1)-1)) = 40.5k$$

This value is not readily available and can be considered a value from trade series, such as 39K, or 39K2 or 40k2 from precision series. This will lead to some approximation of the threshold values. Using the common 39k we can calculate where they move, obtaining Vth=2.98V and Vtl=2.46V, which are adequate for the application.
It should be kept in mind that calculations of this kind involve the tolerances of the components. In formulas, we consider the parameters to be fulfilled, without error. In practice, we are dealing with the tolerance of the Vdd (1-5%), that of the resistors (1% - there are resistors of better precision, but the cost becomes terribly high) and the constructive ones of the comparator. Therefore, deviations from the calculated values are always to be expected; The important thing is that, taking into account the errors, the deviations from the actual final values deviate from the calculated theoretical values by such an amount that the system still allows the system to function correctly.

Again, we observe how the comparator, analog function, is very independent from the rest of the digital circuitry, so much so that it can operate as if it were a "classic" component, such as LM311, LM339, etc.

If you want to test this application as well, you will need to use components external to the application. **LPCuB**, which can be flying or assembled on a small board.
From the point of view of the program, the **COUT output (C1OUT)** must be enabled to pick up the reaction.
CPOL **(**C1POL) must also **be at level 1**, otherwise the feedback will be negative and will not result in the desired hysteresis.

# Comparator and Timer0

Through the **C1T0CS bit** we can connect the T0CKI input of the Tmer0 to the output of the comparator.
This connection takes place entirely internally, in this way:

- if **C1T0CS** is at level 1 (default), the Timer0 count input is determined by the T0CS bit of the option register, i.e. it can be connected to FOSC/4 (the counter increments with the internal clock) or to the T0CKI pin (the counter increments according to the pulses on the pin. In this case, the pin takes over this function and cannot be used otherwise.

- if **C1T0CS** is programmed at level 0, the comparator output is connected to T0CKI, regardless of the choice made on T0CS. Under these conditions, the timer counter increments as a result of the comparator switches.

There is a problem to be noted: if T0CKI is selected by T0CS=1, the relevant bit of TRIS is set to 1, i.e. as input, regardless of whether 0 is written to the direction register. If we select **C1T0CS=0** , the pin that shares T0CKI cannot be used as an output, since its TRIS is locked at 1.

What is this option used for? It can be used for an input signal that is counted by the timer, for example, by creating a precise trigger point and cleaning the signal of noise; or by squaring an input signal with a sinusoidal waveform or otherwise different from a digital signal.
One application can be the detection of the passage to zero of an alternating voltage, the evaluation of the duration of a period, the passage counter and the like.

# The second channel of the comparator is the Cvref module

If the single comparator integrated in **12F510** is completely analogous (label aside) to the one we have just seen and so are channels 1 of the double comparator **of 16F506/526**, the second channel of the latter has a substantial difference: **it has a different reference voltage module from that of channel 1**.

In practice, for the single channel (or channel1), by means of the **CNREF bit** (**C1NREF**), we can connect Vin- to an internal voltage, produced by a precision band-gap generator with a fixed output at 0.6V. This generator is the same one used by the ADC (analog-to-digital converter) module: it is precise, but it produces only one voltage: if we want other references, we must have them externally.

On the other hand, in the Baselines equipped with 2 comparators, a module called **CVREF** has been added, which can produce 32 different voltages. This allows channel 2 of the comparator to be used in a wide range of applications without the need to have an external voltage, or a divider, as a reference for comparison.

The **CVREF** module is an analog element independent of the processor, from which it receives commands through a special register, **VRCON:**

| R/W-0 | R/W-0 | R/W-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| VREN | VROE | VRR | — | VR3 | VR2 | VR1 | VR0 |
| bit 7 | | | | | | | bit 0 |

**Bit 7 VREN** Activates/disables the reference voltage generator.

| VREN | Function |
|---|---|
| 1 | **qualified** |
| 0 | **disabled** |

By default, this bit is zero, so the generator is disabled. To get the reference voltage, you need to enable the bit from the program.

**6-bit VROE** Enable voltage output on pin

| VROE | Function |
|---|---|
| 1 | **Output Enabled** |
| 0 | **output disabled - CVREF pin available** |

By default, this bit is zero, so output is disabled. To have the reference voltage on the **CVREF** pin, you need to enable the program bit. In this case, the pin is not available for other uses.

**Bit 5 VRR** switches the two available voltage ranges

| C1POL | Function |
|---|---|
| 1 | **CVREF = (VR3:0/24) * Vdd** |
| 0 | **CVREF = (VR3:0/32) * Vdd** |

This allows you to have 32 different voltages in two ranges of 16 each, values selected from the state of the following bits.

**3:0 bits VR3:0** select the voltage in the range defined by the previous bit. If we have a ***Vdd=5V***, the reference will vary according to this table:

| VRR = 1 | | | VRR = 0 | | |
|---|---|---|---|---|---|
| VR3:0 | moltiplicatore | Valore @Vdd=5V | VR3:0 | moltiplicatore | Valore @Vdd=5V |
| 0000 | 0.000 | 0V | 0000 | 0.250 | 1.25V |
| 0001 | 0.042 | 0.21V | 0001 | 0.281 | 1.41V |
| 0010 | 0.083 | 0.42V | 0010 | 0.313 | 1.56V |
| 0011 | 0.125 | 0.62V | 0011 | 0.344 | 1.72V |
| 0100 | 0.167 | 0.83V | 0100 | 0.375 | 1.88V |
| 0101 | 0.208 | 1.04V | 0101 | 0.406 | 2.03V |
| 0110 | 0.250 | 1.25V | 0110 | 0.438 | 2.19V |
| 0111 | 0.292 | 1.46V | 0111 | 0.469 | 2.34V |
| 1000 | 0.333 | 1.67V | 1000 | 0.500 | 2.50V |
| 1001 | 0.375 | 1.87V | 1001 | 0.531 | 2.66V |
| 1010 | 0.417 | 2.08V | 1010 | 0.563 | 2.81V |
| 1011 | 0.458 | 2.29V | 1011 | 0.594 | 2.97V |
| 1100 | 0.500 | 2.5V | 1100 | 0.625 | 3.13V |
| 1101 | 0.542 | 2.71V | 1101 | 0.656 | 3.28V |
| 1110 | 0.583 | 2.92V | 1110 | 0.688 | 3.44V |
| 1111 | 0.625 | 3.12V | 1111 | 0.719 | 3.59V |

The accuracy of the multiplication factor with respect to Vdd is ±1/2 LSB.

 It should be noted that, as seen in the table, the values that can be obtained from the VREF **are a function of the Vdd and vary with it**. Thus, their "accuracy" of the reference voltage is relative to that of the Vdd.

It will be necessary, if precise absolute values are needed, to have an adjustment of the Vdd of equal precision or to implement a self-calibration system or to use not absolute values, but parameterized ones; This point becomes important in the case of non-stabilized power supply, such as that provided by batteries. Moreover, it should be noted that the VREF is adjusted in steps and that it does not cover all possible values; hence, a certain tolerance is necessary in any case.
In addition:

 It should be noted that, obviously, **the Vdd must be greater than the reference voltage to be obtained**.
In particular, the datasheet recommends that you do not use the module to generate voltages that are

if the Vdd is less than 2.7V.

Again, it is necessary to know that, by varying the VR bits, the response of the generator is not immediate, but requires at least 10us for the voltage to stabilize. So, if we need to vary this voltage in the program, we will need to introduce a small delay, for example in this form:

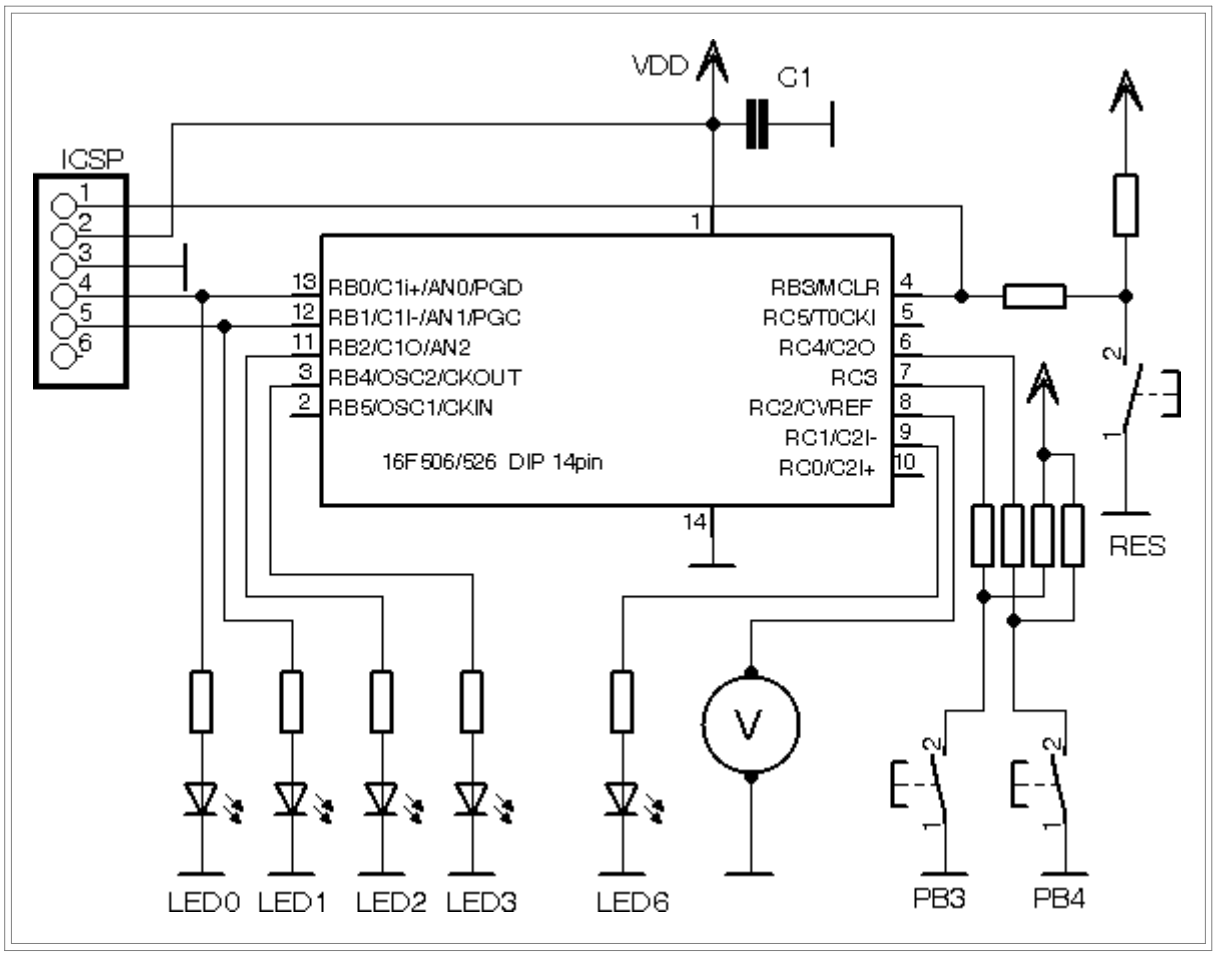
```
; delay 10us @4MHz
DLY10US  MACRO
    goto  $+1     ; 5 x 2us
    goto  $+1
    goto  $+1
    goto  $+1
    goto $+1
       ENDM
```

Given the low number of instructions that make up the algorithm, the use of a subroutine is not particularly convenient, also considering the two-tier limit of the Baseline stack.

---

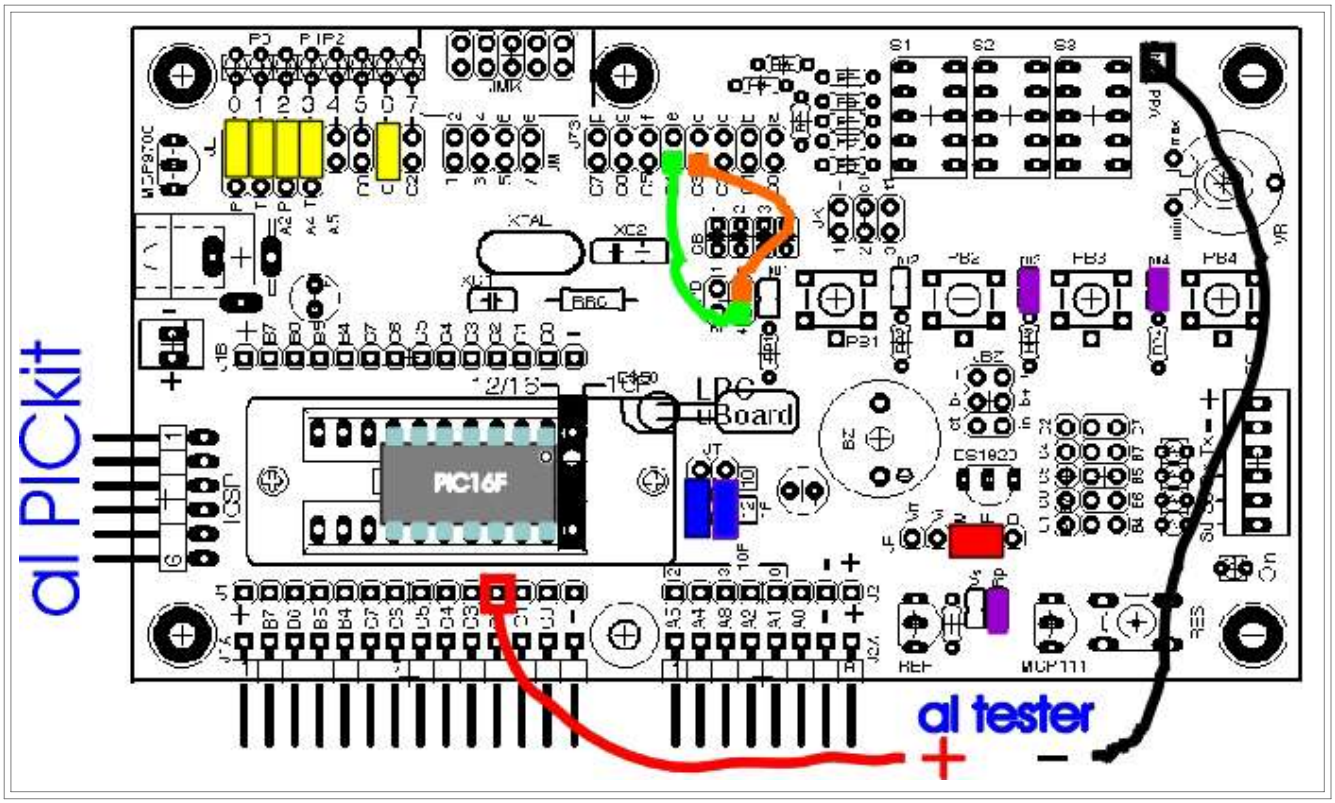
## A test for Vref

We can carry out a simple test for the possibilities of the voltage generator by sending this voltage to the output pin.
We use one button to switch between the next steps of the range and another button to change the range.
Four LEDs will indicate the binary value set in the VR bits, while an additional LED will indicate the set range. We use 16F506 or 16F526.




Since we want the CVREF output to be able to measure it, the RC2 pin is not usable for anything else. You will need a tester or voltmeter to measure the voltage generated on the RC2/CVREF pin.

About the QL200:




The "green" and "red" flying jumpers connect PB3 to RC3 and PB4 to RC4 respectively. The function of the program is as follows:

| PB3 | PB4 |
|---|---|
| VR Advancement | VRR gearbox |

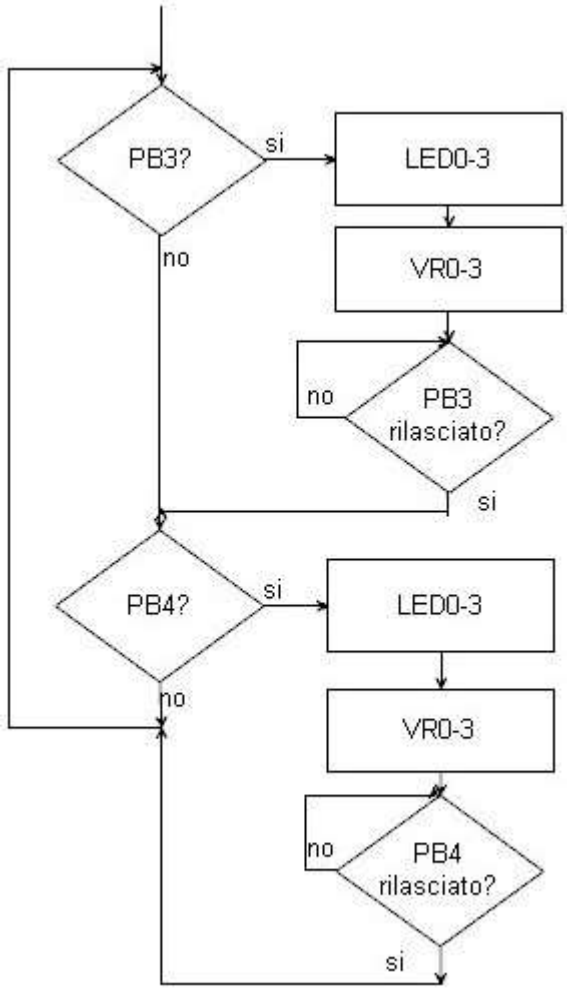
"Purple" jumpers enable pull-ups.
The "yellow" jumpers connect the LEDs to RB0,1,2,4 and RC1. The logic imposed by the program will be this:

| LED4 | LED1 | LED2 | LED3 | LED6 |
|---|---|---|---|---|
| VR0 | VR1 | VR2 | VR3 | VRR |

The tester will be connected between a pin of RC2 and a ground pin, on the 10 or 20V full scale range.

## The program

To carry out an application of a certain complexity, it is first necessary to identify the actions that are necessary.




Here we can consider several logical blocks:

- Button management
- LED management
- VREF module management

There are no particular time problems and the different actions do not have critical timings, but for the management of the buttons it is necessary to provide for the debounce, in order to avoid false advances of the selections.

The LEDs are lit to correspond to level 1 of the bits they represent.

The **VREF** module is activated as a consequence of pressing the buttons: starting from an initial 0 imposed on the **VR** and **VRR** bits, each press of **PB3** increases the value of the **VR** as if it were a binary counter.

Each press of **PB4** changes the **VRR level**.
The Reset button is active and deletes all settings, returning the program to the beginning.

From an instruction perspective, you need to define I/O as inputs and outputs depending on the destination.
To access the digital functions, it is necessary to exclude the analog English of the ADC, We also assign the prescaler to the Timer0, to the extent that we will need for the timings of the program, which, being of the order of milliseconds, require the maximum ratio:

```
;  Disable analog inputs to have digital inputs
clrf        ADCON0

; disable T0CKI
; OPTION def 11111111
;            1------- GPWU disabled
;            -1------ GPPU disabled
;            --0----- clock interno
;            ---1 ----done
;            ----0--- prescaler al timer
;            -----111 1:256 AM
    movlw b'11010111'
    OPTION

; Initialize I/O
    clrf    PORTB           ; pre clear latch
```

```
    CLRF    PORTC
    movlw   b'00101000'     ; RB4,2,1,0 come out
    Tris    PORTB
    movlw   b'00111111'     ; RC1 out, RC3.4 input
    tris    PORTC
```

Then you have to initialize the comparators, disabling them; we only need the CVREF module:

```
; disabilita comparatori
; al POR      11111111 C1OUT
;             -1------ no COUT
;             --1----- Normal polarity
;             ---1---- TOCS
;             ----0--- disabled
;             -----1-- C1IN-
;             ------1- C1IN+
;             -------1 ! No wakeup
    movlw   b'11110111'
    movwf   CM1CON0
; to the      11111111
  ROP
;             1------- C2OUT
;             -1------ no COUT
;             --1----- Normal polarity
;             ---1---- TOCS
;             ----0--- disabled
;             -----1-- C2IN-
;             ------1- C2IN+
;             -------1 ! No wakeup
    movlw   b'11110111'
    movwf   CM2CON0
```

and we can also initialize the Vref module to have the output on the pin:

```
; Enable VREF for External Output
; ROP         00000000
;             1---------- Disabled
;             -1------ CVREF out
;             --1----- low range
;             ---0-------nn
;             ----0000 minimum value
    movlw   b'11100000'
    movwf   VRCON
```

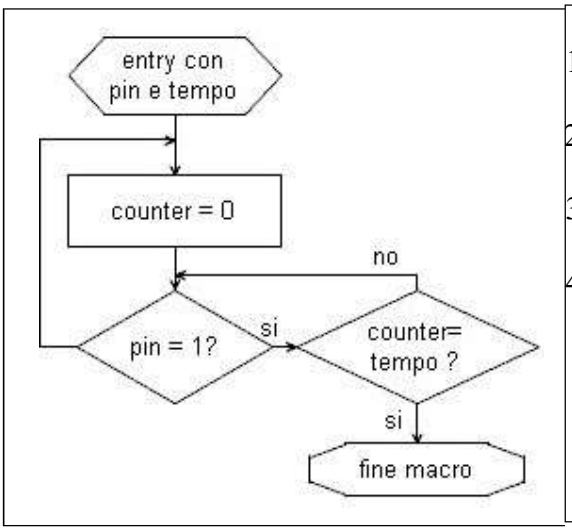
As for the number of lines, initialization is complex, but only because these are composed of comments describing the functions of the various bits of the control registers. This is useful until you have "the hand", then they can be eliminated, leaving only the instruction ones, which, in the end, are very few.

The management of the debounce is delegated to an algorithm that uses Timer0 and inserted in the source as a macro.
The choice of the macro over the subroutine has the disadvantage of occupying more program memory, but of reducing the risk of exceeding with the subroutines and exceeding the capacity of the stack (which, remember, is only 2 levels).

```
;*****************************************************************
; WAITFORH
; Pin wait at high level, with debounce
; PIC Baseline con FOSC = 4MHz
; You must have configured Timer0 with 1:256
prescaler WAITFORH   MACRO  port,pin,dbtime
     local   step, w0, w1
     variable  step = dbtime * .1000 / .256
   pagesel $           ; select current page

W0 CLRF    TMR0        ; Start time count from 0
W1 BTFSS   port,pin    ; pin = 1 ?
    Goto   w0          ; No - try again, resetting the time count
   movf    TMR0,w      ; Yes - Check if it has remained at 1 for the
                       time
   xorlw   Step        ; The set time
   skpz                ; If yes, end macro
    Goto   w1          ; If not, start counting the time all over
                       again
           ENDM
```

This is a parameterized macro; This allows you to insert labels for variables or constants in the macro call, which will be used by the Assembler in the build.
Here, we have as parameters the pin to be checked and the waiting time, in ms.
To understand the details, let's draw the flow chart that allows us a better understanding of the algorithm.




The way it works is simple:
1. the Timer0 register is reset. The count starts at 0.
2. As long as the required pin is not 1, the counter is continuously reset to zero
3. When the pin goes to 1, the timer counts with the clock
4. If the pin returns to 0, for a bounce, the cycle is restarted from step 1

Only if the pin has been at level 1 for a time equal to the time indicated, the macro is terminated.
It is, therefore, a "blocking" procedure, i.e. with a polling that stops every other operation until the one in progress has been successful.

Now let's look at the details:

- **WAITFORH MACRO pin, time** The definition of the macro label is accompanied by the parameters that specify the pin to be tested and the waiting time, which must be expressed in micro seconds.

- `local step, w0, w1` we insert a statement that has the opposite function of `GLOBAL,` i.e. it limits the value of the labels in question to the local scope only. This allows the same labels to be reused in other areas without having compilation errors.

If we don't use this directive, the internal labels **w0** and **w1**, which are related to addresses in program memory, are stored in the compilation list and if they appear to be linked, elsewhere in the source, to other addresses, the compiler will not be able to terminate its action, issuing an error message for duplicate label. This solution is an alternative to using the **$ sign**:

```
WAITFORH   MACRO port,pin,dbtime
     local   step
     variable  step = dbtime * .1000 / .256
   pagesel $           ; Select Current Page for Jumps
   clrf    TMR0        ; Start time count from 0
   btfss   port,pin    ; pin = 1 ?
    goto   $-2         ; No - try again, resetting the time count
   movf    TMR0,w      ; Yes - Check if it has remained at 1 for the
   xorlw   step       time
   skpz               ; The set time
    goto   $-6         ; If yes, end macro
           ENDM
```

Here the advantage of **$** comes from not having to enter labels; the compiler will calculate the value of **$-2** and
**$-6** and will handle them without creating labels in the list and thus avoiding the possible duplication error. The problem with using **$** is that there is a wide possibility of making mistakes in the calculation of the positions to be moved to the Program Counter; in Baselines the opcodes are 1 byte, but not all: **goto** and **call** are 2. Also, if there is a macro (or a pseudo opcode, as in this case) in the jump area, its real length must be taken into account, even if it appears as a line in the source. For example, in this case we have **skpz** which is equivalent to **btfss STATUS,Z**, which is always a byte. But if we had used **bz k** , this equates to :

```
BTFSC   STATUS,z
 GOTO   k
```

That is, TWO lines are compiled, for a total of THREE bytes to be skipped, even if the line in the source is only one! Only by keeping these risks in mind, **$** can be used and it goes without saying that the greater the width of the jump in number of lines, the greater the possibility of error.
Another point to consider is the following: using label, the destination of a jump is obvious, but with the **$** it is no longer obvious: the source loses its comprehensibility. Finally, it should be considered that the result will not be portable on processors where the instructions are encoded on a different number of bytes.
Labels, in the end, are more cost-effective

- `variable step = dbtime * .1000 / .256` this determines the value to be loaded in TMR0 to count at the time **time**. If, for example, we enter the parameter **dbtime** = 10, the compiler will compute the expression as:
  **step = 10 * 1000 / 256 = 39.06**
  which will be rounded up to 39 (27h). This is the value that **TMR0 will reach** starting from 0, clocked at 4MHz (1us instruction cycle) and with the prescaler equal to 1:256, after 9.984us. In the count there is a certain approximation, but this does not matter in the application.

Using the Assembler functions, there is no need, therefore, to resort to the various "*Timer0 Calculators*", since you just have to enter the desired time and MPASM will calculate for you.

The limits that can be set range from 1 to 65 ms.

- `pageel $` This is necessary to make the macro usable even in relocatable compilations, bringing the page selection bit to the one where the macro ends up being during compilation, allowing gotos to operate correctly.

- The comparison between the desired value and the count register is made with the exclusive OR: if the two values are the same, the result is 0 and the **Z** flag is set.

The macro to check the low pin level is completely identical; Only the level test changes, with:

```
btfsc   port,pin    ; pin = 0 ?
```

The source is in the ***13A_vref.asm*** file, compileable for 16F506/526, with its project.

Macros are collected in a mini library that is included at the beginning of the source (***basemacro.asm***). The fact that it is a collection of macros of which only three are used in the execution of the program does not create any elongation of the source, with unnecessary memory consumption. Remember that macros must be declared before they can be used, since the compiler must have already evaluated them in order to replace their label with the instruction content. However, if a macro is declared, but then not used, nothing is written to the source. To avoid text overlaying, the `NOLIST` statement prevents the library contents from being printed in the ***.list file***. If we want to see macros appear in the listing (which is not the .hex object), we just need to exclude the directive.

In addition to the macros we have just seen, related to the buttons, we also use another one, which allows you to copy bits from one file to another without modifying the remaining bits:

```
;*****************************************************************
; CoPy Bits fron File to File - Copy bits from one file to another
; c_mask is the mask that identifies the bits to be copied:
; 1 to copy, 0 to not edit CPYBFF
        MACRO   c_mask,source,target
       movf  source,w
       xorwf target,w
       andlw c_mask
       xorwf target,f
        ENDM
```

It is still a macro with parameters, which in this case are, in order: the mask of the bits to be moved, the source file, the target file.
Operations are carried out with XOR and AND. The mask is a binary number where the bits at 1 correspond to those to be moved and the 0 bits to those not to be modified.

## A Note for the Perplexed

Reading the source, doubts may arise; Let's anticipate the solutions:

1.

```
; disable T0CKI from RB5
```

We don't use this pin, but since we still have to write the OPTION register to set the clock and prescaler of the timer, we add that too, just to keep that in mind.

2.

```
; increment counter for VRR bits
```

There are only 4 bits of VR, so you may think that it is necessary to stop the count when it exceeds 0Fh, starting from 0. However, we have a useful coincidence: the VR3:0 bits are the first 4. In the binary counter, the values follow one another as follows:

| bit7:4 | bit3:0 |
|---|---|
| .... | .... |
| 0000 | 1111 |
| 0001 | 0000 |
| 0001 | 0001 |
| .... | .... |
| 1111 | 0000 |
| 1111 | 0001 |
| 1111 | 0010 |
| .... | .... |
| 1111 | 1111 |
| 0000 | 0000 |

That is, considering only the first 4 low bits, the count always goes from 0 to F. Since the remaining bits are not considered in the move, there is no need to reset the counter periodically.

3.

We mentioned that changes to the VRCON registry require 10us of stabilization, which doesn't appear here. The reason is simple: they are not needed, in the sense that we do not use the voltage generated at the output except to measure it with the tester and not in an application where we expect responses from an external circuit.

4.

Finally, there is no debounce for closing the buttons because it is not needed: as soon as the contact is closed, the corresponding action is immediately performed. This takes a few micro seconds, so it is possible that, at the beginning of the release test, the button will appear open. But that doesn't matter since the algorithm we described above doesn't declare contact open until it's open for at least 10ms, thus canceling out the effect of short bounce pulses.

Once the chip is programmed and the tester is connected, we will get a reading of about 0V, while the LEDs will be off, except for LED6 which indicates **VRR = 1**.
By pressing **PB3**, we will change the VR bits and the LED3:0 will indicate their status. Pressing **PB4** will change **VRR**, changing the range.
The **RES button** returns the program to the beginning, resetting the positions reached.

# Comparators and Wake Up

Comparators can wake ***up*** the processor from its sleep state, which we will see described in more detail in Tutorial 15A.

To get the wake up from sleep you need to act on the ! **CWU** that activates this function, bringing it to 0 from the program (by default it is at 1, so the option is disabled).

Wake up, i.e. the exit from the sleep state, occurs when the output of the comparator changes state. There is no need to enable the COUT physical output, as the processor checks the internal status.
**The event is signaled by the CWUF bit of the STATUS, which goes to 1.**

**Similarly to an interrupt, once the wake up has been carried out, the causes must be erased** and this is done by rereading the control register of the comparator. All it takes is one line:

```
movf    CMCON0, w    ; cancella stato wakeup del comparatore
```

However, if we do not do this, the wake-up request message remains active and a return to sleep is terminated immediately.
This line will have to be added even before going to sleep, to avoid that the intervention flag is set and you find yourself in the previous condition, with the processor not remaining in sleep, a difficult situation to debug.

Where there are two comparators, there are also two activation bits, one in each control register (**C1WUF** and **C2WUF)**, but **the STATUS flag is unique**.
So, if we have enabled wake up from only one comparator there is no problem, but if we have enabled it from both, we need to be able to distinguish the source, for example by keeping a copy of the status of the comparators before going to sleep:

```
Whichcomp:
      movf    hold1,w          ; Retrieve Previous Value Comp1
      XORWF   CM1CON0,w        ; the same as the current one ?
      skpnz                    ; No - next test
       Goto   isCMP1           ; Yes - Manage Event
      movf    hold2,w          ; Retrieve Previous Value Comp2
      XORWF   CM2CON0,w        ; the same as the current one ?
      skpnz                    ; No - end of test
       Goto   isCMP2           ; Yes - Manage Event
```

Sleep mode drastically reduces processor power consumption and is therefore suitable for situations where you are waiting for an event. If this can be managed by the comparator, it is ideal as it continues to work even in sleep. We have seen that this is an element that does not depend on the processor except for the management of the control register and therefore, when sleep blocks the Program Counter, the comparator continues to work: when the expected event changes the state of the output, the processor is returned to normal activity.
This is suitable for applications such as data loggers that need to monitor a voltage: as long as it is within the desired parameters, the processor is idle, idle with low current consumption. When the voltage goes out of bounds, the comparator wakes up the processor. Once the event is detected, the program sends the micro in sleep waiting for the next one. The average power consumption is significantly reduced and this makes it possible to power batteries for long periods.

In the sleep and wake up tutorial we can see more details of how to manage this mode.

## Conclusions

The comparator is a device full of possible uses. It can be used individually or even both channels at the same time, together with the internal or external reference voltage, with the dynamic possibility of varying the configuration from the program. for complex applications, such as multi-level thresholds, window comparators, Sigma-Delta ADC converters, PWM, and signal conditioning.

Microchip's PICmicro® Comparator Tips 'n Tricks publication has a number of interesting examples.

# 13A_10F204.asm

```
;*****************************************************************
; 13A_10F204.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 13A
;                     Comparator Tests
;
;    PIC            : 10F204/6
;    Support        : MPASM
;    Version       A: V.20x-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :  Afg
;
;-----------------------------------------------------------------
;
;  Pin use :
;  ---------------
;     10F2004/6 @ 8 pin DIP      10F2004/6 @ 6-pin SOT-23
;
;                |__\/__|                  *______|
;           NC -|1    8|- GP3         GP0 -|1    6|- GP3
;          Vdd -|2    7|- Vss         Vss -|2    5|- Vdd
;          GP2 -|3    6|- NC          GP1 -|3    4|- GP2
;          GP1 -|4    5|- GP0               |______|
;               |______|
;
;                           DIP  SOT
;    NC                      1:  Nc
;    Vdd                     2:  5: ++
;    GP2/T0CKI/FOSC4/COUT    3:  4: Out  LED
;    GP1/ICSPCLK/CIN-        4:  3:
;    GP0/ICSPDAT/CIN+        5:  1:
;    NC                      6:  Nc
;    Vss                     7:  2: --
;    GP3/MCLR/VPP            8:  6: MCLR
;
;*****************************************************************
;=================================================================
;            DEFINITION OF PORT USE
;
; GPIO map
; |  3  |  2  |  1  |  0  |
; |-----|-----|-----|-----|
; |     | LED4|CIN- |CIN+ |
;
;#define        GPIO,GP0    ;
;#define        GPIO,GP1    ;
#define    LED  GPIO,GP2    ; LED4
;#define        GPIO,GP3    ;

;
;*****************************************************************
; ###############################################################
```

```
; Choice of #ifdef
 10F204 processor
        LIST      p=10F204
        #include <p10F204.inc>
 #endif
 #ifdef 10F206
        LIST      p=10F206
        #include <p10F206.inc>
 #endif
          Radix  DEC

; ##########################################################################
;                          CONFIGURATION
;
 ; No WDT, no CP, pin4=GP3
  __config  _WDT_ON & _CP_OFF & _MCLRE_ON

; ##########################################################################
;==========================================================================
;                           RESET ENTRY
;
; Reset Vector
        ORG     0x00

; internal oscillator calibration; FOSC4 force disabled in the
; Case of incorrect calibration value
      andlw   0xFE
      movwf   OSCCAL

; I/O initializations on reset
; GP2 as output
      movlw  b'11111011'
      TRIS   GPIO

; disable T0CKI for    have free GP2
; OPTION def '11111111'
;             1------- GPWU Disabled
;             -1------ GPPU disabled
;             --0----- Internal Clock
;             ---1---- done
;             ----1--- Prescaler to Timer
;             -----111 1:256
     movlw  b'11011111'
     OPTION

; Enable comparator, with no output and with normal inputs
; no wakeup, no timer input
; CMCON def  '11111111'
;             1 ------- cmpout
;             -1------ Output disabled
;             --1----- Normal polarity
;             ---1---- T0CS for Timer0 in
;             ----1--- Enable Comparator
;             -----1-- vref- = CIN-
;             ------1- vref+ = CIN+
;             -------1 no wakeup
     movlw  b'11111111' ; Output Disabled
```

```
;     movlw  b'10111111' ; Output Enabled
      movwf  CMCON0
;      Goto   $           ; Execution Block

; Comparator Test
CLP BTFSC   CMCON0, CMPOUT
      bsf    LED
     BTFSS   CMCON0, CMPOUT
      Bcf    LED

     Goto    CLP         ; Undefined loop
;     Goto   $            ; Execution Block

;*****************************************************************
       END
```

# 13A_526.asm

```
;*****************************************************************
; 13A_526.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 15A
;                     Counter with 7-segment display output.
;                     The figure presented on the
;                     display every 500ms. Once the oevrflow is reached,
;                     lit the decimal point.
;    PIC            : 16F526 or 16F506
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :  Afg
;
;-----------------------------------------------------------------
;
;  Pin use :
;  ----------------
;       16F506/526 @ 14 pin
;
;                    |‾‾\/‾‾|
;              Vdd -|1    14|- Vss
;              RB5 -|2    13|- RB0
;              RB4 -|3    12|- RB1
;         RB3/MCLR -|4    11|- RB22
;              RC5 -|5    10|- RC0
;              RC4 -|6     9|- RC1
;              RC3 -|7     8|- RC2
;                   |______|
;
;    Vdd                      1: ++
;    RB5/OSC1/CLKIN           2:
;    RB4/OSC2/CLKOUT          3: Out  LED4
;    RB3/! MCLR/VPP           4: MCLR
;    RC5/T0CKI                5:
;    RC4/C2OUT                6: In   PB4
;    RC3                      7: In   PB3
;    RC2/CVref                8: CVREF
;    RC1/C2IN-                9: Out  LED6
;    RC0/C2IN+               10:
;    RB2/C1OUT/AN2           11: Out  LED2
;    RB1/C1IN-/AN1 /ICSPC    12: Out  LED1
;    RB0/C1IN+/AN0 /ICSPD    13: Out  LED0
;    Vss                     14: --
;
; ###################################################################
; Choice of #ifdef
  processor   16F526
        LIST      p=16F526         ; Processor Definition
        #include <p16F526.inc>
  #endif
  #ifdef  16F506
```

```
        LIST      p=16F506          ; Processor Definition
         #include <p16F506.inc>
  #endif

        Radix     DEC

; ##########################################################################
;                           CONFIGURATION
  #ifdef  16F526
; Internal Oscillator, 4MHz, No WDT, No CP, RB3=MCLR
 __config  _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_CPDF_OFF & _MCLRE_ON
  #endif
  #ifdef  16F506
; Internal oscillator, no WDT, no CP, MCLR
  __config  _IntRC_OSC_RB4EN  & _IOSCFS_OFF & _WDT_OFF & _CP_OFF &
_MCLRE_ON
  #endif


; ##########################################################################
;                            RAM
; general purpose RAM
        CBLOCK 0x10          ; start of RAM
      VRcntr area            ; counter for VR
        ENDC

 ;*************************************************************************
;=========================================================================
;            DEFINITION OF PORT USE
;
;P ORTC map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;|     | PB4 | PB3 |CVREF| LED6|     |
;
;#define          PORTC,0     ;
#define     LED6  PORTC,1     ; LED6 for VRR
;#define          PORTC,2     ; CVREF
#define     PB3   PORTC,3     ; #define
button      PB4   PORTC,4     ; button
;#definePORTC,5   ;     VRRLED
        EQU b'000010'

;P ORTB map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;|     |     |LED3 |LED2 |LED1 |LED0 |
;
#define     VRLED0 PORTB,0     ; LEDs for
VR #define  VRLED1 PORTB,1     ;
#define     VRLED2 PORTB,2     ;
;#define           PORTB,3     ; MCLR
#define     VRLED3 PORTB,4     ; LEDs for
VR
;#define           PORTB,5     ;
;
;*************************************************************************
```

```
;
;    Notes:
;
;*****************************************************************

; ###########################################################################
;                           MACRO
; use macro library for Baseline
  #include   C:\PIC\Library\Baseline\basemacro.asm
 LIST

; ###########################################################################
;                           CONSTANTS
;
dbtime  EQU  .10          ; Debounce time 10ms

; ###########################################################################
;                           RAM
    CBLOCK  0x010
  VRcntr                  ; counter for VR
    ENDC

; ###########################################################################
;                           RESET ENTRY
;
; Reset Vector
        ORG     0x00

; MOWF Internal Oscillator
        Calibration OSCCAL

; ###########################################################################
;                         MAIN PROGRAM
Main:
; Reset Initializations

; Disable analog inputs to have CLRF digital inputs  ADCON0

; disable T0CKI from RB5
; Internal clock, prescaler 1:256
         ;      b'11010111'
         ;        1-------     GPWU Disabled
         ;        -1------     GPPU disabled
         ;        --0-----     Internal Clock
         ;        ---1----     Falling
         ;        ----0---     prescaler to TMR0
         ;        -----111     1:256
        movlw   b'11010111'
        OPTION

; Initialize I/O
        CLRF    PORTB         ; Pre Clear Latch
        CLRF    PORTC
        movlw   b'00101000'   ; RB4,2,1,0 out
        three of a kind      PORTB
        movlw   b'00111101'   ; RC1 out
```

```
        Tris   PORTC

; Disable Comparators
; to the ROP 11111111
;            1 ------- C1OUT
;            -1------ no COUT
;            --1----- Normal polarity
;            ---1 ---- TOCS
;            ----0 --- disabled
;            -----1-- C1IN-
;            ------1- C1IN+
;            -------1 no wakeup
    movlw  b'11110111'
    movwf  CM1CON0

; to the     11111111
  ROP
;            1------- C2OUT
;            -1------ no COUT
;            --1----- Normal polarity
;            ---1---- TOCS
;            ----0--- disabled
;            -----1-- C2IN-
;            ------1- C2IN+
;            -------1 No wakeup
    movlw  b'11110111'
    movwf  CM2CON0

; Enable VREF for External Output
; to the ROP 00000000
;            1 ------- disabled
;            -1------ CVREF out
;            --1----- low range
;            ---0 ---- nn
;            ----0000 minimum movlw
    value  b'11100000'
    movwf  VRCON
    Bsf    LED6              ; VRR Status Replication
    CLRF   VRcntr            ; reset VR counter

Mainloop:
    BTFSS  PB3
     Goto  xPB3
    BTFSS  PB4
     Goto  xPB4
    goto   Mainloop

; pressure management
PB3 xPB3
; increment counter for incf vrr
    bits   VRcntr,f
    ; BTFSC VRcntr,4       ; bit 4=1 ? (>0Fh)
    ; CLRF  VRcntr         ; Yes - Reset

; copy VR bits to VRCON
    CPYBFF b'00001111',VRcntr,VRCON

; copy bits to PORTB
```

```
    CPYBFF b'00000111',VRcntr,PORTB
; fix bit3->PORTB4
    btfss  VRcntr,3
     Bcf   VRLED3
    btfsc  VRcntr,3
     Bsf   VRLED3
; waiting for release
    PB3 WAITFORH
    PB3,dbtime

; New Cycle
    Goto   Mainloop

; pressure management
PB4 xPB4
; movlw bit VR
    toggle VRR
    xorwf  VRCON,f

; toggle LED6
    movlw  VRRLED
    xorwf  PORTC,f
; waiting for release
    PB4 WAITFORH
    PB4,dbtime

; New Cycle
    Goto   Mainloop


;*****************************************************************
;                            THE END
        END
```

# basemacro.asm

```
;*****************************************************************
; basemacro.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Collection of MACROS
;
;    PIC            : Baseline - clock 4MHz
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :  Afg
;
;-----------------------------------------------------------------
 NOLIST
; Macros included *******************************
; Name         Function                    Use
;
; DECW      w = w - 1                   DECW
;
; CPYBFF    Copy bits from a file       CPYBFF source,target,c_mask
;           to the other
; MOVLF     Copy literal to file        MOVLF  literal,target
; MOVFF     Copy file to file           MOVFF  source,target
; MOV16FF   1st 16-bit copy             MOV16FF source,target
; DLY10US   10us delay @4MHz            DELAY10US
; WAITFORH debounce high                WAITFORH port,pin,dbtime
; WAITFORL low debounce                 WAITFORL port,pin,dbtime

; Bit and File Operations
;*****************************************************************
; Working with W
; Decrementa W.  C = 1 if w>0 after decw
decrement       MACRO
       addlw       -1
      ENDM

;*****************************************************************
; CoPy Bits fron File to File - Copy bits from one file to another
; c_mask is the mask that identifies the bits to be copied:
; 1 to copy, 0 to not edit CPYBFF
         MACRO   c_mask,source,target
       movf  source,w
       xorwf target,w
       andlw c_mask
       xorwf target,f
         ENDM

;*****************************************************************
; Load a literal into a file without changing W.
MOVLF  MACRO   literal,target
        xorlw   Literal
        movwf   Target
        XORLW   Literal
```

```
         XORWF   target,f
        ENDM

;*****************************************************************
; Move the contents of one file to another
MOVFF  MACRO   source,target
         movf    source,w
         movwf   ENDM
        target

;*****************************************************************
; Move content to 16bit
MOV16FF  MACRO  source,target
         movf    source,w
         movwf   Target
         movf    source+1,w
         movwf   target+1
        ENDM

; Utilities
;*****************************************************************
; DLY10US
; Waiting for 10us
; PIC Baseline with FOSC = 4MHz
; You must have configured Timer0 with 1:256 prescaler
DLY10US  MACRO
    Goto  $+1          ; 5 x 2us
    Goto  $+1
    Goto  $+1
    Goto  $+1
    Goto  $+1
         ENDM

;*****************************************************************
; WAITFORH
; Pin wait at high level, with debounce
; Release if port,pin remained at high level for t=dbtime
; PIC Baseline with FOSC = 4MHz
; You must have configured Timer0 with 1:256 WAITFORH
prescaler  MACRO  port,pin,dbtime
    local   Step, W0, W1
    variable  step = dbtime * .1000 / .256
    pageel $; Select Current Page for W0 Jumps CLRF  TMR0  ;
Start time count from 0
w1  BTFSS   port,pin    ; pin = 1 ?
     Goto   w0          ; No - Please try again, resetting the time
                        count
    movf    TMR0,w      ; yes - check if it has remained at 1 for the
                        time
    xorlw   Step        ; set up
    skpz                ; If yes, end macro
     Goto   w1          ; If not, resume the time count from 0
           ENDM

;*****************************************************************
; WAITFORL
; Pin wait at low level, with debounce
; Release if port,pin remained at low level for t=dbtime
```

```
; PIC Baseline with FOSC = 4MHz
```

```
; You must have configured Timer0 with 1:256 WAITFORL
prescaler  MACRO  port,pin,dbtime
    local   Step, W0, W1
    variable  step = dbtime * .1000 / .256
    pageel $; Select Current Page for W0 Jumps CLRF  TMR0  ;
Start time count from 0
w1  BTFSC   port,pin    ; pin = 0 ?
     Goto   w0          ; No - Please try again, resetting the time
                        count
    movf    TMR0,w      ; yes - check if it has remained at 1 for the
                        time
    xorlw   Step        ; set up
    skpz                ; If yes, end macro
     Goto   w1          ; If not, resume the time count from 0
           ENDM
```