Knowledge of baseline

The clock

The purpose of the tutorial is to learn about the possibilities that the microcontroller offers to generate the clock.

So far, we've seen circuits that use the internal clock, derived from a precision RC oscillator built into the chip. This system is extremely convenient and practical in most applications, as there is no need for a particular frequency value or even an accuracy better than the 1% offered by the integrated oscillator.

It should also be noted that the internal clock, usually set at 4MHz, in various chips has:

- the possibility of taking on other values
- the possibility of being brought back to the outside on a special pin

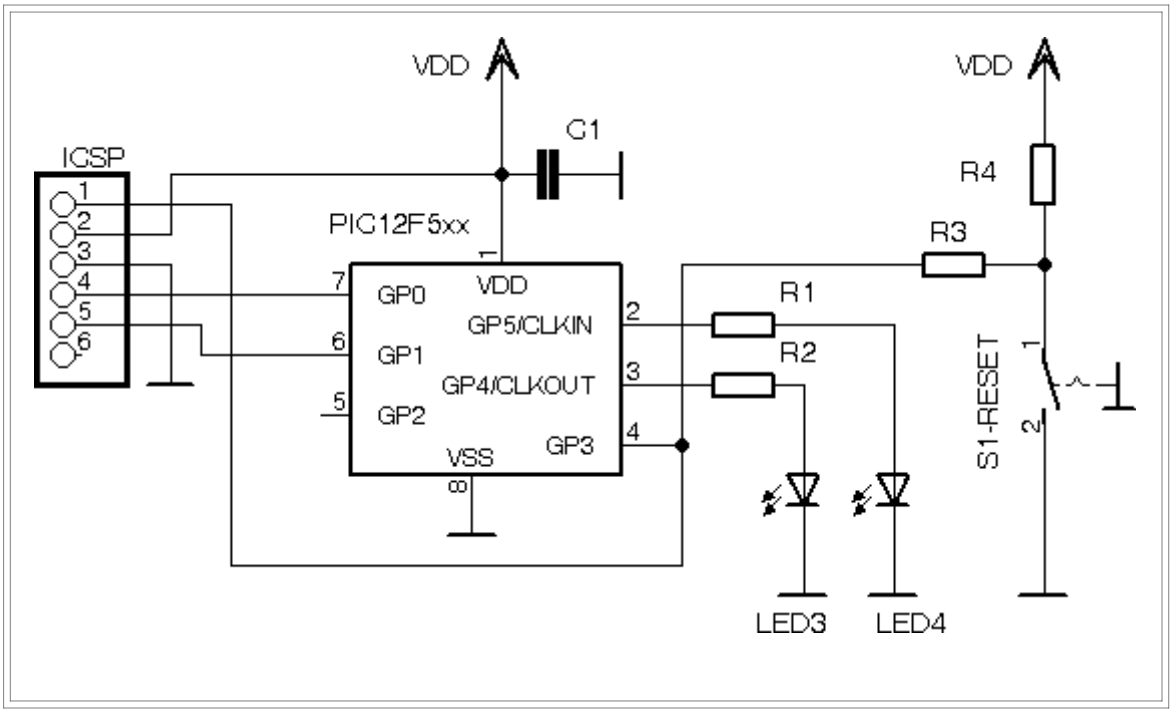
Let's start by checking these two possibilities.
First of all, it must be said that the **Baseline PICs** are the "simplest" components and are not equipped with particular sophistication; only some chips have the indicated options, while in the **Midrange** and **PIC18F families** there are also a dozen alternative frequency values that can be generated internally, as well as the possibility of multiple oscillators.
Now, limiting ourselves to the Baselines, we can observe that the chips have internal clocks, some with the option of a double-speed one and the possibility of externally deriving the generated frequency, in the form of ***Fosc/4***:

PIC	Internal RC	FOSC /4
10F200/2/4/6	4MHz	Yes
10F220/2	4-8MHz	Yes
12F508/9	4MHz	no
12F510	4-8MHz	no
12F519	4-8MHz	no
16F505	4MHz	Yes
16F506	4-8MHz	Yes
16F526	4-8MHz	Yes

We can, first of all, check the action of the double clock on the execution of the program.

4 and 8 MHz CLOCKS

Let's go back to a circuit among those already seen, that of exercise 2 for the 12F519:



The 12F508/509 and 16F505 will not be able to be used, as they do not have dual internal clocks.

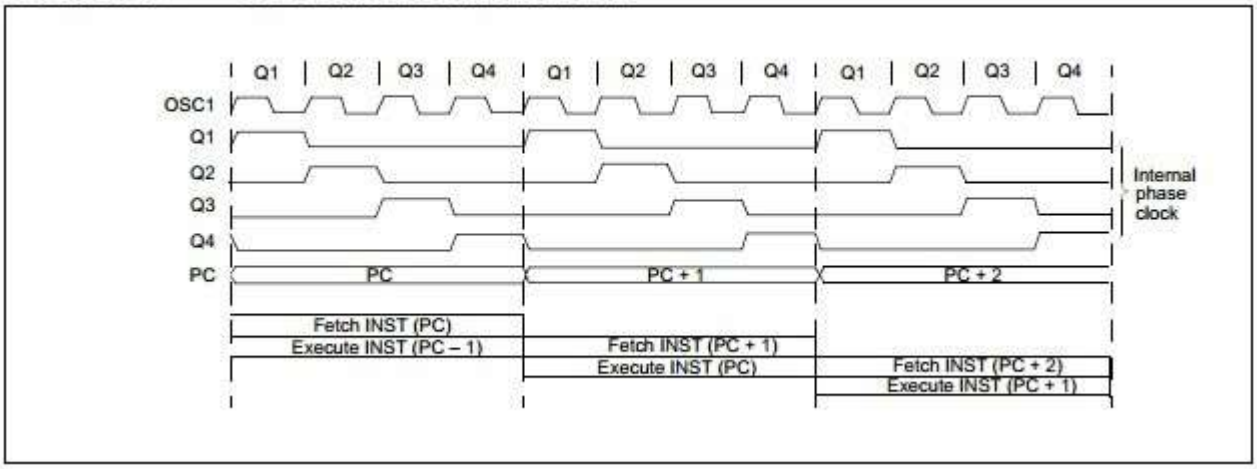
Now let's retrieve the program used to make the two LEDs flash from exercise 2A.

We have seen that the speed of the flashing depends on the time routine. And this, in turn, based on instruction cycles, depends on the time it takes the processor to execute an instruction. Let's exemplify in a table:

Clock	FOSC/4	Education Cycle
4MHz	1MHz	1us
8MHz	2MHz	500ns

This is due to the fact that the execution of an instruction requires 4 clock pulses

FIGURE 3-3: CLOCK/INSTRUCTION CYCLE



primary, in 4 phases (Q1, 2, 3 and 4) that make up a cycle-instruction.

If the clock doubles, going from 4 to 8MHz, it halves the execution time, which goes from 1us to 500ns. This means that, if we have made a time loop that at 4MHz takes 1 second to run, when the clock goes to 8MHz the loop will take only 1/2 second.

How do we change the clock?
If in the PIC18F and in the Enhanced Midrange the clock switching between the various possible values can be managed by the program, in the Baseline this parameter is part of the initial configuration settings and cannot be changed during the execution of the program.
We have to choose the desired clock value in the config.
So we'll have, for 4MHz:

```
;#########################################################################
;                          CONFIGURATION
;
; Oscillator internal 4MHz, no WDT, no CP, pin4=MCLR;
  __config _IntRC_OSC & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_ON
```

and for 8MHz:

```
;#########################################################################
;                          CONFIGURATION
;
; Oscillator intern 8MHz, no WDT, no CP, pin4=MCLR;
  __config _IntRC_OSC & _IOSCFS_8MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_ON
```

If we keep the source identical, selecting the clock at 8MHz, the flashing of the LEDs will double in frequency.

To get the same cadence, we need to modify the tempo subroutine:

```
;*****************************************************************
;                        CONFIGURATION
;
; Oscillator internal 8MHz, no WDT, no CP, pin4=MCLR;
  __config _IntRC_OSC & _IOSCFS_8MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_ON
```

If we apply the rest of the program without modification, we get a flashing frequency of the LEDs at twice the speed. To get the cadence of 1Hz, we have two possibilities: the first is to repeat the call to the tempo subroutine twice:

```
mainloop:
; Lights up
  LEDs LED_ON

; Standby 1s
  call    Delay1s     ; 1s @4MHz => 0.5s @8MHz
  call    Delay1s

; turns off
  LEDs

; Standby
  1s call Delay1s     ; 1s @4MHz => 0.5s @8MHz
  call    Delay1s

; loop
  goto    mainloop
```

The second option is to modify the subroutine, adapting it to the new clock:

```
;*****************************************************************
;================================================================
;=                          SUBROUTINES                         =
;================================================================
; Delay = 1 secondo @ Clock = 8 MHz
; Fosc/4 = 500ns , 1 second = 2000000 cycles
Delay1s_8:              ; 1999996 Cycles
  movlw   0x11       ; Initialize Counters
  movwf   d1
  movlw   0x5D
  movwf   d2
  movlw   0x05
  movwf   d3
Dly1s_80:
  decfsz d1, f
   goto   $+2
  decfsz  d2, f
```

```
   goto    $+2
  decfsz  d3, f
   goto    Dly1s_80
; End of Counting 1999996 Cycles - Now add the missing
  4 retlw 0          ; 4 cycles including call
```

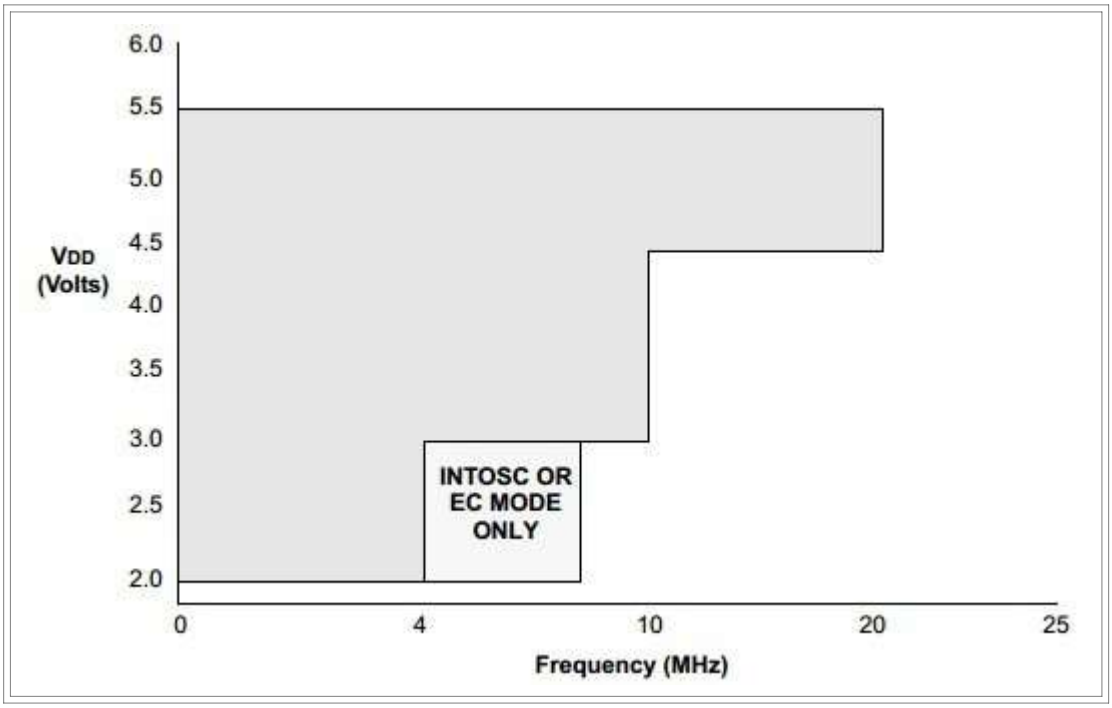
The algorithm, very similar to the previous one, is derived from the usual Delay ***Code Generator site***; we note that the difference lies exclusively in the pre-charge values of the counters to adjust them to the instruction cycle time that has gone from 1us to 500ns.

Frequency and voltage Vdd

It is clear that the possibility of having a chip at a fixed frequency is limiting compared to the availability of more values, since a higher frequency allows for faster execution times in order to adapt to the needs of the controlled process (serial communications, measurements, AD conversions, etc.).


However, it should be well understood that it is not an obligation to send the chip at the maximum possible speed, as, for the execution of the program, a lower clock may be sufficient. This is related to the fact that energy consumption increases quadratically with increasing frequency; It follows that a low-power application will have to have a minimum clock and, in any case, lowering the frequency of the oscillator will save energy.

In addition, high working frequencies are possible only in direct proportion to the Vdd: PICs, in general, can work between 2 and 5.5V of Vdd, but the maximum possible frequency is linked to it, according to the following diagram:

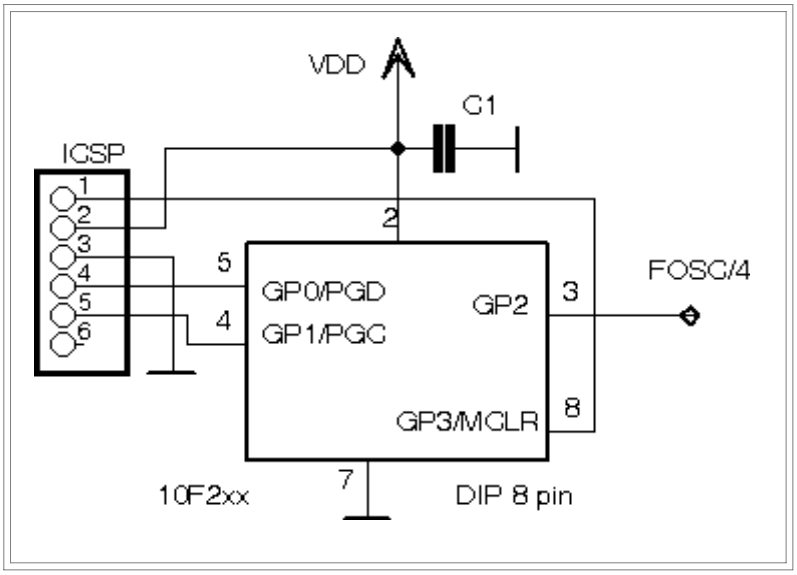


The table indicates the typical relationship between the Vdd and the working frequency: below 4.5V it is not possible to operate safely above 10MHz. Below 3V, the temperature should not exceed 4MHz. The characteristic indicated is that of fig.14-1 of the PIC16F526 data sheet, but it can be extended to all Baselines, and, with a few variations, to all PCIs.

FOSC/4

Verification of the **Fosc/4** output function is reserved for those who have an oscilloscope or at least a probe to locate logic levels.
In Baselines, some chips have this option, which consists of being able to send to a pin the signal of the internal clock divided by 4, i.e. the frequency of the instruction cycle.
This is not commonplace, but it can be used by external components, providing them with a clock that is synchronized with that of the processor.

We can use an extra simplified scheme:



That the microcontroller has nothing connected but ICSP and the power supply depends on the fact that what we care to check is the output of the clock frequency. This requires a pin to be dedicated to this purpose and is, therefore, a basic choice, to be made with an action on bit 0 of the **`OSSCAL register`**

REGISTER 4-3: OSCCAL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-0
CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	FOSC4
bit 7							bit 0

By bringing this bit to 1, the internally generated frequency will be applied to the **`GP2`** pin, which, therefore, cannot be used for any other purpose:

```
; Internal Oscillator Calibration and FOSC/4 Enable
          iorlw 1
```

```
          movwf OSCCAL
```

At **the POR,** as we have seen, the **W** register contains the calibration value of the oscillator. The **OR inclusive** `iorlw` brings the 0 bit of W to 1 and with this enables the function.

```
; Internal Oscillator Calibration and FOSC/4 Enable
          iorlw 1
          movwf OSCCAL
; Execution Block
          goto  $
;End of compilation
           END
```

Once the desired oscillator mode has been set, all that is needed: by applying the oscilloscope probe, or the logic probe, or a frequency meter to the **GP2** we will be able to detect the ***Fosc/4 signal***.
Incidentally, if you have a frequency meter and want to check the effect of the calibration, you can try excluding it altogether:

```
; FOSC/4 rating
          ;iorlw 1
          ;movwf  OSCCAL
           bsf OSCCAL,FOSC4
```

and entering other values other than the calibration value, such as the extremes of the adjustment range:

```
    ; Minimum calibration value
            movlw   b'10000000'
;FOSC/4 rating
            iorlw   1
            movwf   OSCCAL
```

and:

```
; Maximum calibration value
            movlw   b'01111110'
; FOSC/4 rating
            iorlw   1
            movwf   OSCCAL
```

This makes it possible to check the possible range of variation with respect to the nominal frequency.

You may also want to consider the possibility that you do not want to activate the Fosc/4 output, since it subtracts a pin from other functions. This condition should already be predisposed to by the calibration value of the oscillator set at the factory, which always has bit0 = 0.
If, however, we assume that the calibration value has been deleted or corrupted, there is a possibility that bit0 is at 1. Normally this does not happen, as the programming devices,

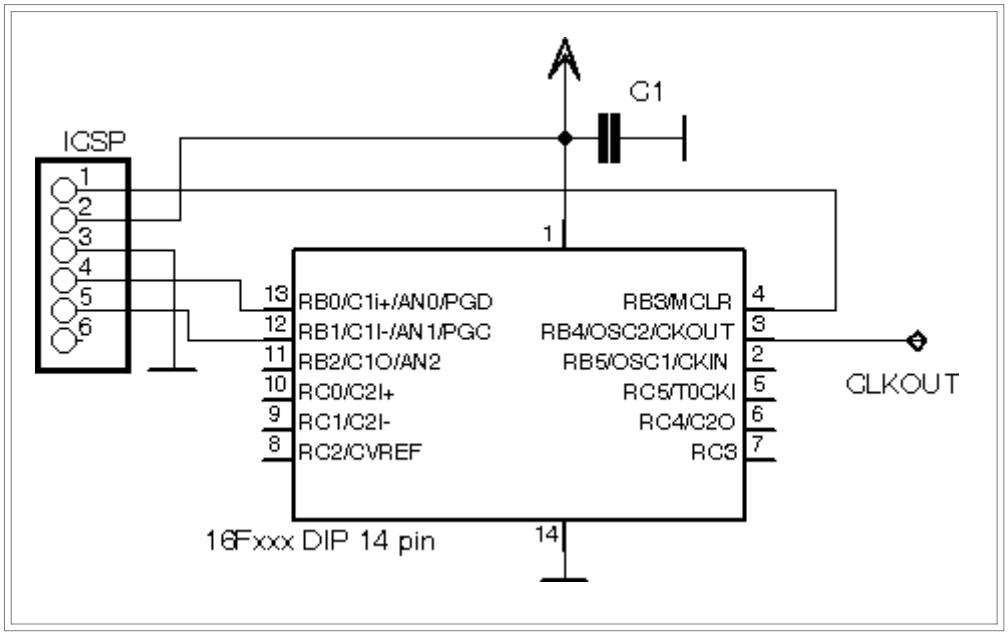
by connecting to the PIC, they report if the calibration value is incorrect and recovering the correct one is very simple, with the appropriate utility of the Pickit Programmers.
However, if you want to include an additional security in the program:

```
; Internal oscillator calibration with safety for
; disabling FOSC/4
          andlw b'11111110'     ; ensures bit0=0
          movwf OSCCAL
```

which allows you to safely have, in any case, Fosc/4 disabled.

As for the **12F Baseline**, none of them provide clock output, while it is possible in the 16F Baseline.

The option does not refer to ***FOSC/4***, but to the internal RC mode with clock output on the pin **OSC2 (CLKOUT)**, where, however, it does not depend on the calibration value, but on the initial config. Here, too, all you need is the power supply and a tool to check the output frequency.



It is always necessary to act in the config phase:

```
 __config  _FOSC_IntC_CLKOUT    ; IntRC with CLKOUT on OSC2
```
oppure
```
 __config  _IntRC_OSC_CLKOUT    ; IntRC with CLKOUT on OSC2
```

Here, too, we will be able to do the same checks seen previously

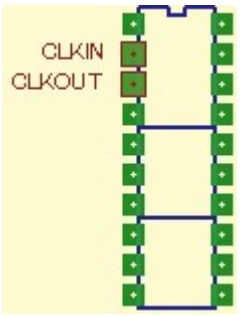
External Oscillators

In addition to the internal modes, most chips offer the ability to:

- Externally apply the clock frequency
- Use external components to have a crystal oscillator
- use external components to have an RC oscillator

PIC	Internal RC	FOSC/4	External Oscillator		
			Quartz - max.		
10F200/2/4/6	4MHz	Yes	-	-	-
10F220/2	4-8MHz	Yes	-	-	-
12F508/9	4MHz	no	4MHz	Yes	-
12F510	4-8MHz		8MHz		-
12F519	4-8MHz		8MHz		-
16F505	4MHz	Yes	20MHz		Yes
16F506	4-8MHz				
16F526	4-8MHz				

First of all, let's see that the PIC10F are excluded from these possibilities; the reason is simple: for an external oscillator you need a couple of pins to dedicate and in chips with only 4 I/Os available the possibility makes no sense.
On the other hand, chips from 8 pins and up can take advantage of an external oscillator (again at the expense of losing a couple of pins as I/O), but with a maximum frequency equal to that of the internal oscillator.
With 14-pin chips, the frequency of the external oscillator can be as high as 20MHz, even if the internal one is only 4 or 8.
It is evident that an oscillator made with the help of external components requires to have chip pins available to connect them to the internal circuit. In the Baselines these pins are identified as **`OSC1/OSC2`** or **`CLKIN/CLKOUT;`** since they **are stackable pin to pin**, the arrangement is as follows:



Package	CLKIN	CLKOUT
8 pins	Pin 2	Pin 3
14 pin		
20 pins		

Clock options and config

The possible options for the operation of the clock are described in the datasheets of the individual components, but it is possible to quickly have a list of them by consulting not the entire data sheet, but only the ***processorname.inc*** files related to the various chips. We have seen that this file is the list of the default labels of the main functions of the chip and that this file is readable with any editor. In section **_CONFIG** we find the oscillator modes as the first items. For example, for **16F526** we have this situation:

```
;----- CONFIG Options -----------------
_FOSC_LP            EQU H'0FF8' ; LP oscillator and 18 ms DRT
_LP_OSC             EQU H'0FF8' ; LP oscillator and 18 ms DRT
_FOSC_XT            EQU H'0FF9' ; XT oscillator and 18 ms DRT
_XT_OSC             EQU H'0FF9' ; XT oscillator and 18 ms DRT
_FOSC_HS            EQU H'0FFA' ; HS oscillator and 18 ms DRT
_HS_OSC             EQU H'0FFA' ; HS oscillator and 18 ms DRT
_FOSC_EC            EQU H'0FFB' ; EC oscillator w/ RB4 on RB4/OSC2/CLKOUT 1ms DRT
_EC_OSC             EQU H'0FFB' ; EC oscillator w/ RB4 on RB4/OSC2/CLKOUT 1ms DRT
_FOSC_INTRC_RB4     EQU H'0FFC' ; INTRC w/ RB4 on RB4/OSC2/CLKOUT 1ms DRT
_IntRC_OSC_RB4      EQU H'0FFC' ; INTRC with RB4 on RB4/OSC2/CLKOUT 1 ms DRT
_FOSC_INTRC_CLKOUT  EQU H'0FFD' ; INTRC with CLKOUT on RB4/OSC2/CLKOUT 1 ms DRT
_IntRC_OSC_CLKOUT   EQU H'0FFD' ; INTRC with CLKOUT on RB4/OSC2/CLKOUT 1 ms DRT
_FOSC_ExtRC_RB4     EQU H'0FFE' ; EXTRC with RB4 on RB4/OSC2/CLKOUT 1 ms DRT
_ExtRC_OSC_RB4      EQU H'0FFE' ; EXTRC with RB4 on RB4/OSC2/CLKOUT 1 ms DRT
_FOSC_ExtRC_CLKOUT  EQU H'0FFF' ; EXTRC with CLKOUT on RB4/OSC2/CLKOUT 1 ms DRT
_ExtRC_OSC_CLKOUT   EQU H'0FFF' ; EXTRC with CLKOUT on RB4/OSC2/CLKOUT 1 ms DRT
```

In summary, the following ways are available:

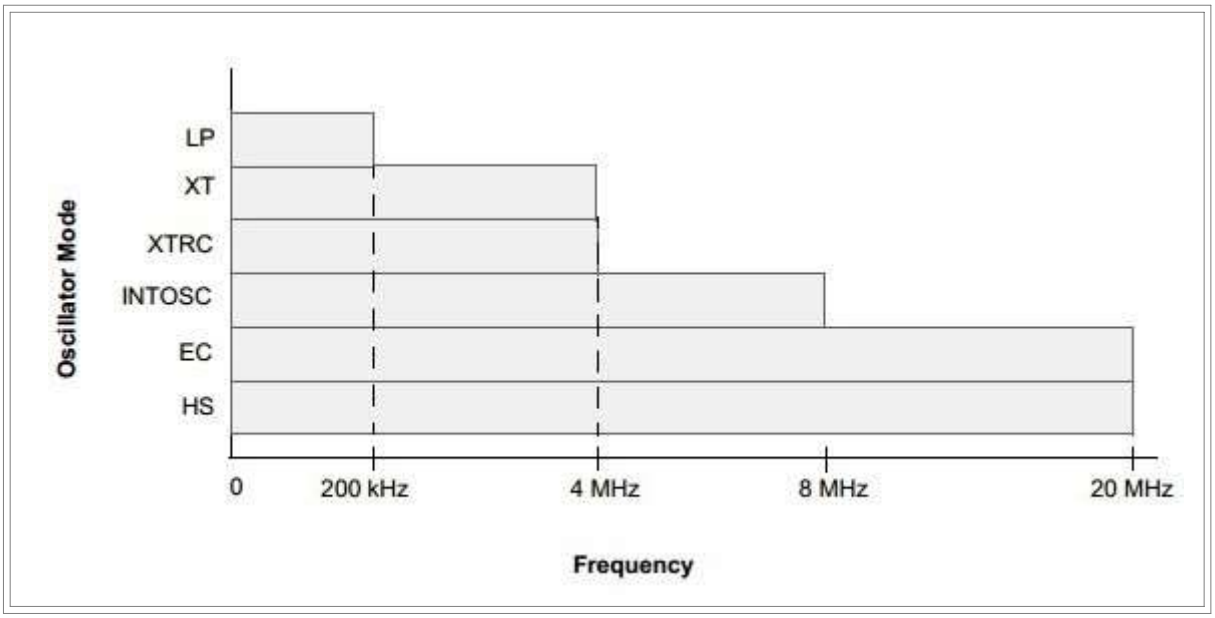
- **LP, XT, HS** - External Crystal Oscillator
- **EC** - External clock source
- **ExtRC** - External RC oscillator
- **IntRC** - Internal oscillator

The last two modes can have a Fosc/4 output.
We can draw up a table for the chips we consider in these tutorials:

PIC	Quartz	Locations		Interiors	
		RC CLKOUT		RC	RC CLKOUT

10F2xx							x	FOSC/ 4
12F508/9	x	x			x		x	
12F510	x	x			x		x	
12F519	x	x			x		x	
16F505	x	x	x	x	x	x	x	x
16F506	x	x	x	x	x	x	x	x
16F526	x	x	x	x	x	x	x	x

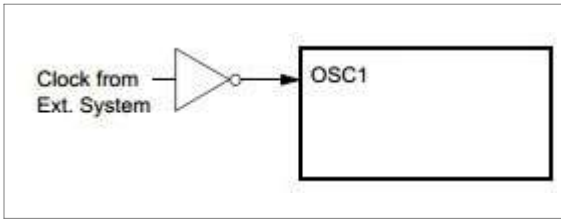
The datasheets offer us a summary graph of the relationship between the oscillator's operating mode and the achievable frequency, which is worth reporting:



Now let's look at the features of the individual options.

External Clock – EC

We can apply an externally generated frequency to the **`CLKIN/OSC1 pin`**.

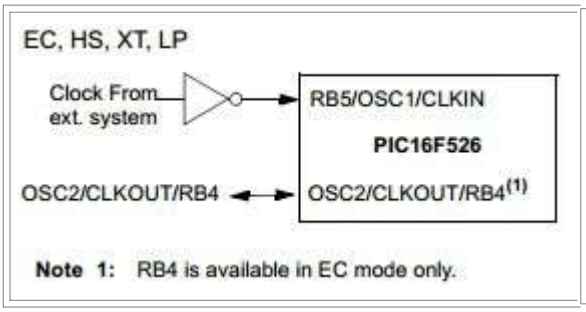


In this case, the pin is set up by the config to receive the external signal that must have characteristics compatible with the microcontroller (TTL levels, rise and fall times indicated in the data sheet).

Only some chips have this option, which is chosen by entering the following entry in the config:

```
 __config _FOSC_EC          ; EC oscillator with RB4 on RB4/OSC2/CLKOUT and 1 ms
DRT
```
oppure
```
 __config _EC_OSC           ; EC oscillator with RB4 on RB4/OSC2/CLKOUT and 1 ms
DRT
```

For Baselines, the **EC** (***External Clock***) option corresponds to the fact that the **`OSC1/CLKIN pin`** becomes the input for the external frequency and cannot assume any other function.

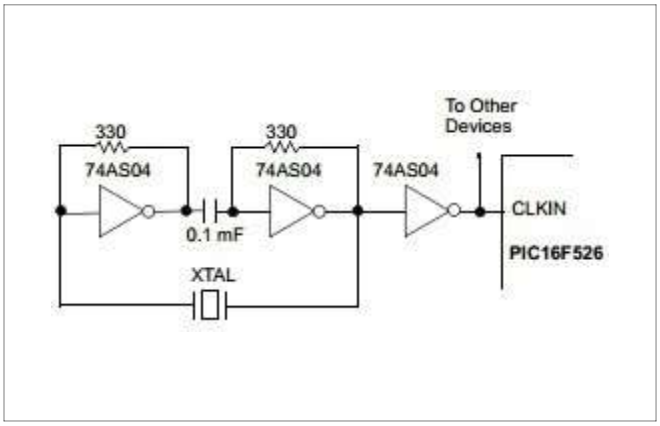


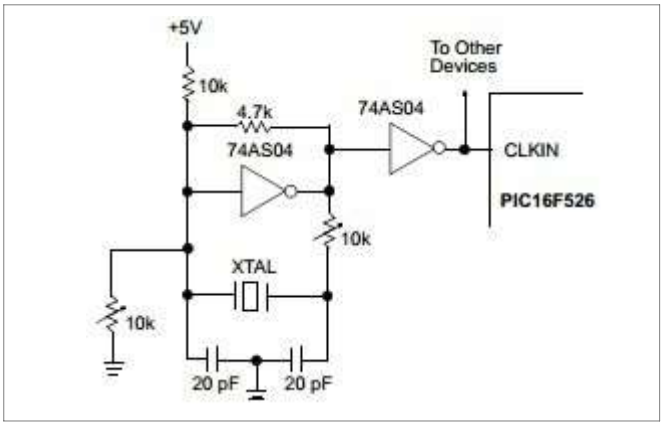
Instead, the **`OSC2/CLKOUT pin`** is assignable as the I/O. In the image on the side, referring to the 16F526, but valid for the other 16F Baseline, it is **`RB4`**.

This mode is not commonly used, but it is applicable where a clock generator already exists in the system, for example when the PIC is part of a complex board together with other processors.

Or when a specific frequency is needed that is produced by a special oscillator, of which there are many models with characteristics of precision and stability over time and with different temperatures and sizes.
The use of these components is usually dictated by the need for high precision and stability over time and with temperature.

In the event that you do not use an integrated oscillator, but you make it with gates, you need to use TTL (LS, AS, etc.) or CMOS HCT to have the right signal trend.



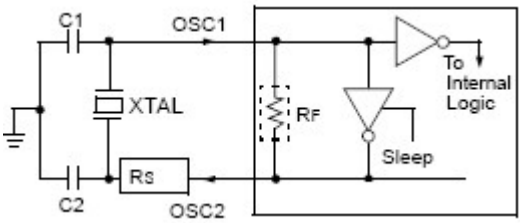


Compared to the built-in clock, an external clock source can serve multiple processors or other circuits. Compared to 4/8MHz, the external clock, for the PIC16F5xx, can increase to 16-20MHz, with the corresponding increase in performance: if at 4MHz an instruction is executed in 1 us, at 20MHz it takes only 200ns.

The component datasheet specifies the maximum applicable frequency as a function of the supply voltage, which, as we have seen, is proportional.

External Clock - LP, XT, HS

We can make a crystal-controlled oscillator (quartz or ceramic oscillator), from a few kHz to a couple of tens of MHz, by adding some external components:



The internal structure of the gates corresponding to the **OSC1/OSC2** pins is modified at the config in order to create an oscillator that uses a quartz crystal or a ceramic element as a stability and frequency element. It is necessary to add the pair of ***C1/C2*** capacitors: usually these are NP0 ceramics typically of 18-22pF.
The resistor ***Rs*** is added using low power crystals to limit the current.

The Rf ***resistor***, whose purpose is to bring the gate into the linear operating zone, is integrated and has a value of about 10Mohm. Usually you don't need an external one.
Note that this option engages both **OSC1/OSC2** pins, so they cannot be used for any other purpose. There are three ways to configure the oscillator gate:

Way	Frequency	C1/C2
LP	32kHz	15-18pF
XT	200kHz-4MHz	15-68pF
HS	4-20MHz	15-47pF

The **LP** (***Low Power***) option is chosen by inserting the following voice in the config:

```
__config ___FOSC_LP    ; LP oscillator
oppure
__config  _LP_OSC      ; LP oscillator
```

for crystals up to 200kHz. Typically these are **32768Hz** quartzes used to generate accurate times, clocks, data loggers, and the like. These kinds of crystals operate with a power of 1-2 microwatts, while most HF quartzes can dissipate from 100 to 500 microwatts. For this reason, when it is necessary to limit power dissipation: an excess of current may not prevent oscillation, but this occurs on high harmonics and is unstable and random, as well as

to the fact that driving with excessive power shortens the life of the glass. From here the Rs may be necessary, mainly for ultra low power elements of the kind used in watchmaking.
In practice, the choice of components for the LP oscillator can be laborious and it may not be immediate to find a suitable crystal for operation on OSC1/OSC2 in LP, especially among the recoveries from the disassembly of various boards; moreover, the generated frequency is linked to the value of the capacitors much more decisively than for the HS and XT modes, which are not too critical.
For LP it ranges from 6-12pF to a hundred, and it may be necessary to have different values for C1 and C2 to get the right oscillation frequency.
Even if the generated frequency depends a lot on the capacitors, using inadequate values, if quartz is suitable, the operation can hardly be missing; with values of the order of 33-100pF, the oscillation can be more stable, but the start-up time of the oscillator can become noticeably high (even seconds!).

The **XT** (***XTal***) option requires:

```
 __config  _FOSC_XT     ; XT oscillator
oppure
 __config  _XT_OSC      ; XT oscillator
```

and should be used with 200kHz to 4MHz ceramic crystals or resonators. Compared to the previous mode, the energy available to the oscillator changes, which is increased. Capacitors, between 18 and 47pF, are not particularly critical, but good quality elements are needed.

The HS **(**High Speed***) option requires***:

```
 __config  _FOSC_HS     ; HS oscillator
oppure
 __config  _HS_OSC      ; HS oscillator
```

This is the way to adopt for crystals from 4MHz upwards, where the energy required by the oscillator is maximum (the aforementioned need for more energy as the frequency increases).
It should be noted that the **12F5xx** does not support this mode: their maximum operating frequency, in any case, is indicated at 4MHz. On the other hand, the **16F5xx** PICs have HS mode, since they can reach 16-20MHz.

Commonly, **XT** and **HS** are the most commonly used: the choice of one or the other depends on the type of crystal used and must be defined, together with the values of the capacitors, to obtain a stable oscillation throughout the range of supply voltages and temperatures to which the microcontroller will be subjected. It is possible that the same crystal can work both ways, but probably one of the two will give a better waveform and stability.

A few notes:

- Once an oscillator mode has been set with external components, they will need to be wired as intended, otherwise the microcontroller will not work.

- If you choose a mode in the config that is not suitable for the components used, it is likely that the oscillator will not start or that it will operate randomly and at frequencies other than the nominal one

- High stability is only achieved by using quality components. The capacitors must be ceramic with a neutral temperature coefficient (NP0), the resistors must be metal layer and the crystal of the type suitable for this use, avoiding badly desoldered recovery elements (as an excess of temperature on the pins causes the performance of the crystal to decay). In any case, the frequency generated will not be more accurate or stable than the components used.

- the config **entries** related to the oscillator modes are unfortunately not unique for all PICs, so much so that Microchip has added various aliases in the ***.inc*** file. This does not prevent them from being insufficient, making it necessary to read the file to find the right labels.

As we can see from the tutorials, most simple and even less simple applications work correctly with the internal oscillator. The data sheets report an accuracy of the order of 1%, which is more than adequate even for fairly critical applications, such as asynchronous serial transmission, while the stability with temperature is adequate within common variations.
In addition, the calibration of the internal oscillator allows you to vary the generated frequency within a certain range and allows both to obtain values slightly different from the fundamental, and to adjust the value as a result of strong temperature variations.

 The fact that many examples on the web show the crystal and capacitors as a typical element of a circuit with PIC is essentially due to the fact that **the chip does not have an internal clock, such as obsolete PICs** (e.g. 16F84, 16F876/877). However, sometimes, it is only due to the lack of knowledge of possible alternative oscillator configurations and, above all, to the fact that the less obsolete PICs all have the option of one more frequencies available through the internal oscillator. Also with regard to precision, it must be said that a crystal is only useful in time-sensitive applications, i.e. those that require high precision in timing. From a more general point of view, unless you need an extremely precise/stable frequency or the chip has no internal oscillator, using an external 4MHz crystal or, worse, an external RC when the internal oscillator exists is completely useless and sacrifices otherwise usable pins for I/O.

External oscillators should be used when:

- The chip does not have an internal oscillator (obsolete chips)
- The high stability of quartz, which is greater than that of the internal oscillator, is required, especially when the temperature changes.
- precise frequencies of values other than those achievable by the internal oscillator are required, e.g. 32KHz for RTC or 2.476MHz for serial transmissions, etc.

 A further note for those who want to make measurements with the oscilloscope on the crystal oscillator: the pin to be tested is **`OSC2/CLKOUT`**, which is the one with the lowest impedance and where the oscilloscope probe or frequency meter creates the least noise. Applying the probe to **`OSC1`** will charge the oscillator with the impedance and capacitance of the same.

Microchip has issued some documents on oscillators:

- **AN826** - Crystal Oscillator Basics and Crystal Selection for rfPICTM and PICmicro® Device

- **AN949** - Making Your Oscillator Work
- **DD31002a** - Oscillator

And here **you can find other pages on the subject**.

In any case, by using proper components and decent wiring, we have never encountered any problems in the implementation of crystal oscillators.
If you have problems in this area, before accusing the chip or anything else:

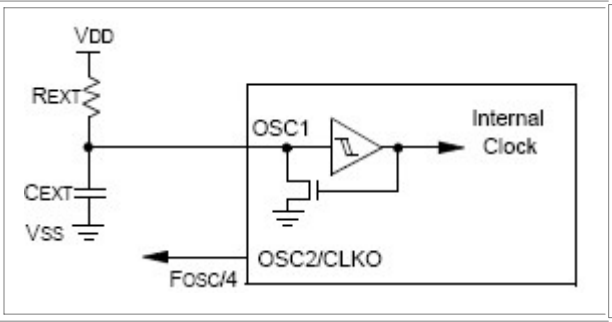
1. **Check the components and the correctness of the connections, especially if you are using breadboards that may have damaged insertion contacts**
2. **Check the correct setting of the oscillator chosen in the config**.

If everything is fine here, the cause is certainly to be found in the crystal that is of an inadequate type (for example, elements that are designed to work overtone, such as quartz for transmission, or crystals for low power inadequate for use with microcontrollers) or is damaged, as happens with recovered elements.
Capacitors rarely pose problems; When using new, ceramic or mutated, NP0 or low-drift ceramic components, there are never any problems in initiating the oscillation, even if the values are not perfectly centered can lead to some small shift in frequency.

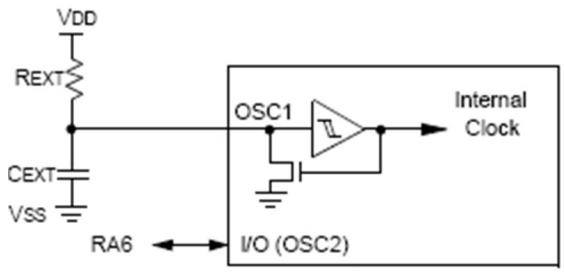
In relation to the possibility of frequency adjustment, it is possible to use a capacitive trimmer together with ***C1*** to adjust its capacitance according to the characteristics of the quartz.

External Clock - RC

An external oscillator can also be made using an **RC network**. We have two options:



The first allows you to have the external RC network, connected to **`OSC1`**, and to **`OSC2`** to obtain the Fosc/**4 output**.
Only the 16F5xx Baselines have this option.

	The second allows you to have the external RC network, connected to **OSC1**, and to **OSC2** have the pin digital I/O function available. This mode is available on both 12F and 16F Baselines.

This mode should only be used in one case:

- when **a frequency is needed that is not available internally**,
- but **of which neither precision nor stability matters** (*timing insensitive application* - non-timing-sensitive applications)
- and the design request is the minimum cost

In fact, a certain precision and stability can only be obtained by using high quality components for R and C: if precision and stable resistors with temperature variations are easily available, the same cannot be said of capacitors. And if the research in the project is related to low cost, this is certainly not achieved by using 0.1% metal layer resistors and mica capacitors.

On the other hand, even if the external RC can theoretically be used up to the maximum working frequency of the chip, this mode is advisable and necessary when you want clocks **with low frequencies**, not possible with crystals, for example 1 or 2kHz.
In this sense, a fundamental consideration must be made:

 in fact, PICs are "**static**" devices, meaning that **they do not need a minimum clock to be able to function**; The clock starts from 0, i.e., it can be interrupted, leaving the execution of the program suspended. This is the key to the operation of the **sleep mode**, where the stop to the processor takes place precisely by stopping the clock.

It is obvious that a low clock causes a proportionally long instruction cycle time: for example, 4MHz of clock gives an execution time of 1us, but 4kHz brings the execution time to 1ms per instruction.
On the other hand, **power consumption is drastically reduced**: for PIC16F506, for example, the supply current is rated at 1.4mA @ 5V with a clock of 20MHz and goes to only 11uA @ 32KHz, with the possibility of lowering the Vdd voltage to 2V (datasheet, param. D010).

From a construction point of view, the specifications indicate that the ***Rext*** must be between 3 and 100Kohm, while Cext can also be omitted using the parasitic capacitances of the circuits, but this choice is completely discouraged for obvious reasons. So ***Cext*** will be able to start from 20pF and up. The "***Electrical Characteristic***" section of the datasheet indicates the specifications of the application.

The RC oscillator is based on the charge of the Cext capacitor through the Rest resistor. When the voltage of about 0.75xVdd is reached, an internal transistor rapidly discharges the capacitor and the voltage drops to about 0.25Vdd, obtaining the typical triangle wave, which will then be squared by the following circuits (see diagrams in the table above). It is possible to calculate the period of oscillation that can be obtained from a combination RC with the formula:

$$T = (R*C)\ ln\ [\ Vdd\ /\ (Vdd-Vih)]$$

where is the switching voltage of the Schmitt trigger placed at the input of the OSC1 pin and of the value of about 0.9Vdd (param. D043).
For example, for ***Vdd=5V, R=10k, and C=22pF*** we have:

$$T = (10k*22pF)\ ln\ [\ 5\ /\ (5-4.5)] = 506ns$$

So the frequency will be:

$$F = 1/\ T = 1\ /\ 506ns = 1.976MHz$$

and: $FOSC/4 = 494kHz$

If we consider only the 5V power supply, we can use an abbreviated formulation:

$$T = R\ *C\ *\ 2.3$$

Obviously, the actual value obtained depends on the tolerance of the R and C components used and on the construction characteristics and tolerances of the chip. In addition, their variation with temperature must be considered.

The **External RC** option requires:

```
__config  _FOSC_ExtRC_RB4   ; EXTRC with I/O function on OSC2
oppure
__config  _ExtRC_OSC_RB4    ; EXTRC with I/O function on OSC2
```

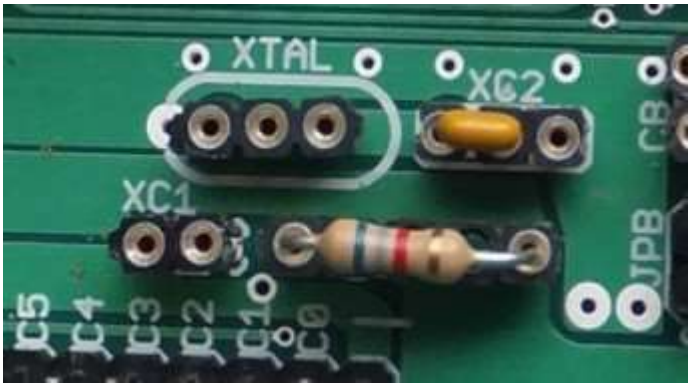
if you want digital I/O available. For chips where ***Fosc/4 output is possible***:

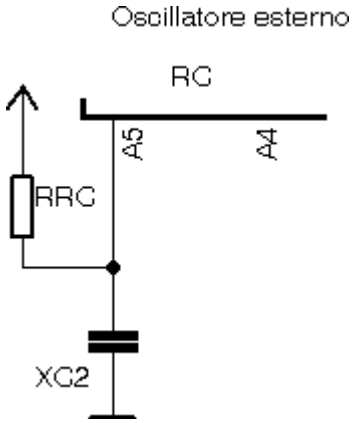
```
__config  _FOSC_ExtRC_CLKOUT   ; EXTRC with CLKOUT on OSC2
oppure
__config  _ExtRC_OSC_CLKOUT    ; EXTRC with CLKOUT on OSC2
```

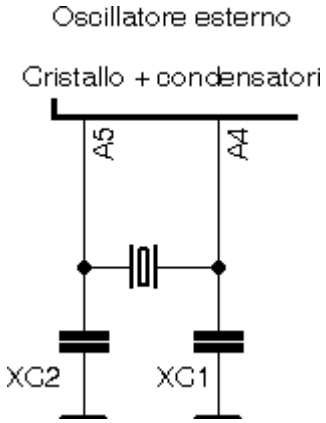
Again, the **config** entries related to these oscillator modes are unfortunately not unique for all PICs, so much so that Microchip has added various aliases to the ***.inc*** file. This does not prevent them from being insufficient, making it necessary to read the file to find the right labels.

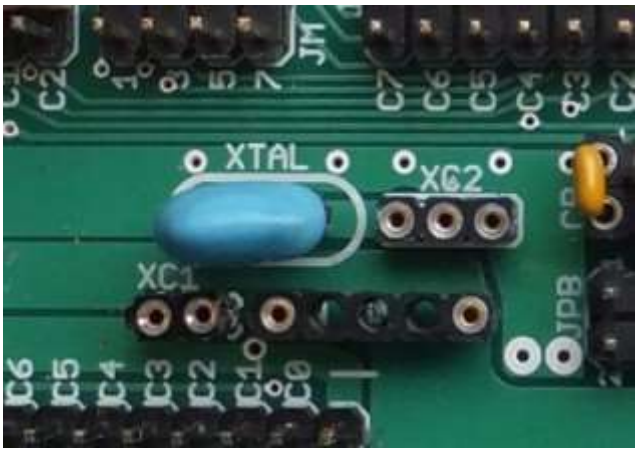
In practice

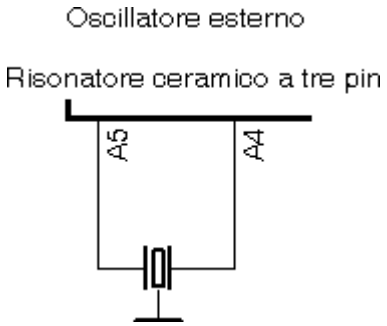
The development board has the possibility to experiment with all the possible variations of the external oscillator as there is an area dedicated to the components:











Note that the three-pin ceramic resonator integrates the capacitors, so there is no need to add them.
The choice of the ceramic oscillator instead of quartz is dictated by the lower cost of this component and its functionality, albeit with less precision/stability.

If you want to experiment with these oscillators, you will need to:

- Insert the components required by each schematic
- Consider that one or two pins cannot be used for any other purpose
- and that in the source the config must be modified according to the chosen mode

If we operate on supports other than **LPCuB** , it will be sufficient to make the indicated circuits, taking care to verify that the connections are secure, especially on breadboards.

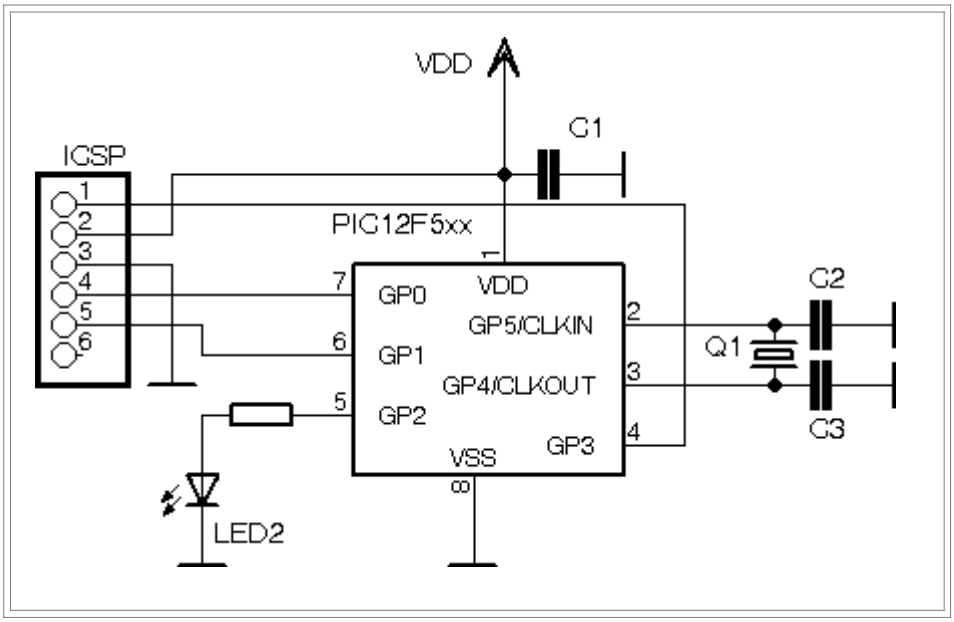
An application example - LP/XT/HS mode

We can basically see the effect of clocking on the execution of the program by flashing an LED. For external oscillator mode, the components must be connected as shown above, occupying the **OSC1/OSC2** pins. This means that the pins cannot be used as I/O. This means that for 8-pin chips (12F5xx) you can lose two out of 6 digital I/O, while for multi-pin PICs the loss of 2 I/Os is less sensitive. Especially:

PIC	CLKIN	CLKOUT	Pins that cannot be used as I/O					
			IntRC	IntRC clockout	ExtRC	ExtRC clcokout	EC	LP/XT/HS
10F2xx	-	-	-	GP2	-	-	-	-
12F5xx	GP5	GP4	-	-	GP5	-	-	GP4-GP5
16F5xx	RB5	RB4	-	RB4	RB5	RB4-RB5	RB5	RB4_RB5

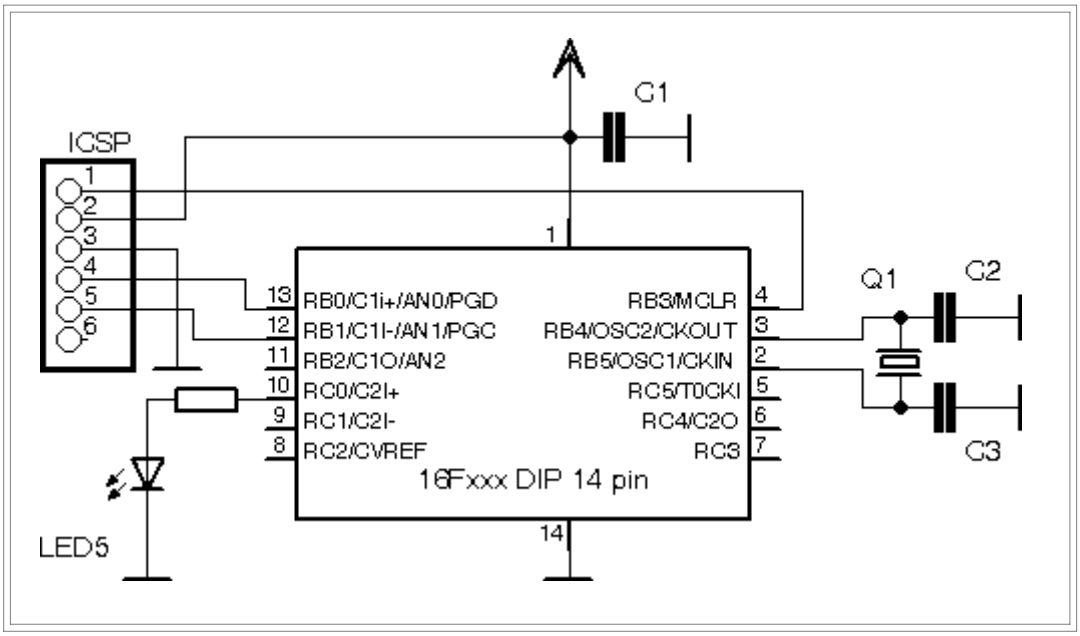
- indicates a mode that is not available for that chip. In the 10F2xx chips, only the internal RC mode is possible, with the eventual ***Fosc/4*** output seen above.

You can experiment with external crystal oscillators with the following schemes:



Quartz up to about 4MHz (or 8MHz, depending on the model) can be used, although the chip works without problems with higher frequencies, but it would be "overclocking", a condition that is not possible.

certified by the
manufacturer.
For the 16F5xx:

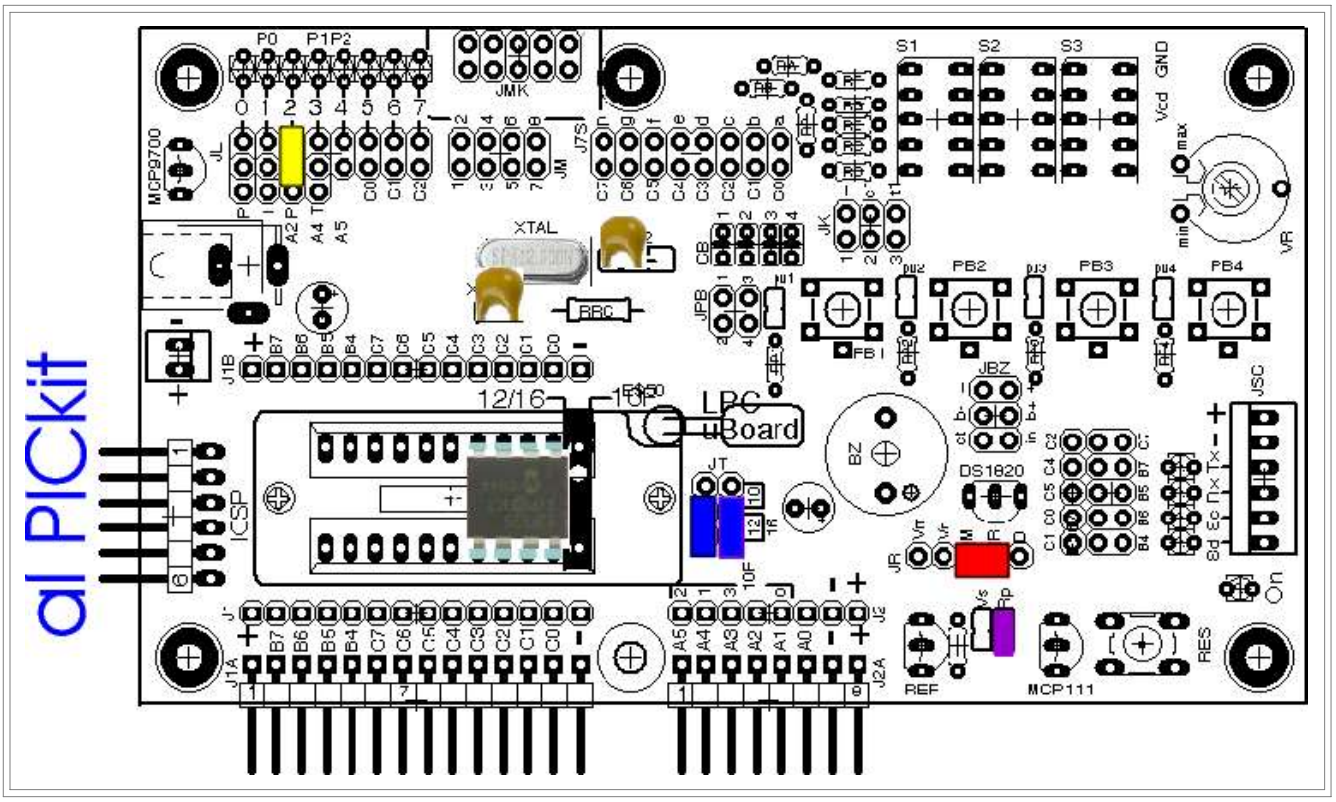


Quartz up to 20MHz can be used. Crystals in HC49 case or in any case suitable for use with such oscillators are fine. Capacitors can be 15-27pF quality ceramics. We can also use two-pin or three-pin ceramic resonators; In the latter case, external capacitors are not required.

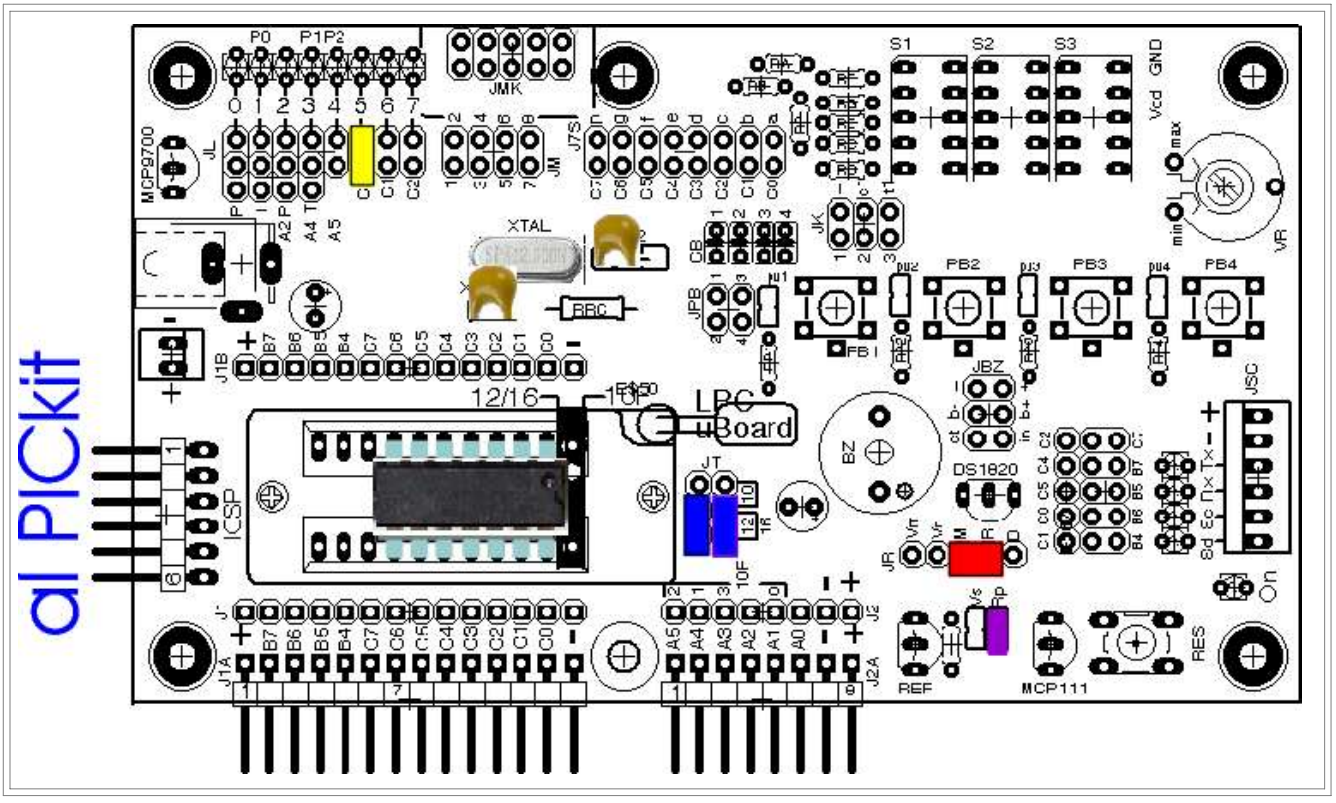
Crystal: 1-4(20)MHz XT or **HS mode**	Crystal: 32-200kHz **LP mode**	2-pin ceramic resonator XT **or** HS **mode**	3-pin ceramic resonator XT **or** HS **mode**
			

If you use salvaged components, make sure that they are suitable for use. In particular, crystals for series oscillators, crystals for harmonic oscillators, ceramic filters from MF and the like are not suitable.
The choice of **LP mode** is mandatory for **32kHz low power quartzes** used in watchmaking; for very miniaturized models it may be necessary to add the ***Rs*** as shown in the previous diagram. For other quartz, usually for frequencies below 4MHz, the **XT mode is fine**, but if the crystal is "hard" it must be set **HS**, even for lower frequencies, especially with ceramic oscillators. **HS** should be chosen for crystals above 4MHz.

Regarding the arrangement on the **LPCuB development board**, for the DIP-8 pins:



and for DIP-14 pins:

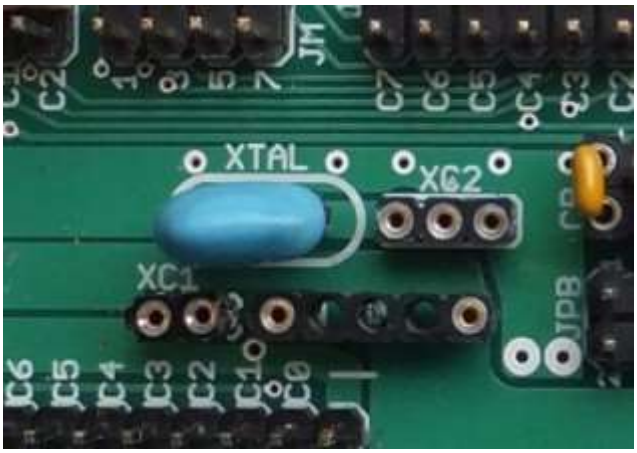


In real life, a magnification of the area of the oscillator components:

The components of the oscillator are inserted on the appropriate sockets with tulip contacts.

Care must be taken, if recovering components are used, to ensure that the terminals are perfectly clean of solder residues, so as not to damage the socket contacts.

The ***XC2'***s socket allows you to install 2.54" or 5.08" pitch capacitors. Notice in the photo the positioning of the capacitor with the tightest pitch.



In the case of using a **three-pin** ceramic oscillator, it is located on the ***XTAL*** socket, while capacitors are not required.

The program

The differentiation between oscillator modes is only in the config. For example, for **XT mode**:

```
; FOR 12F519
; Internal oscillator, no WDT, no CP, GP3 = MCLR
;
 __config _XT_OSC & _WDTE_OFF & _CP_OFF & _CPDF_OFF & _MCLR_ON

; PER 12F508/509
; Internal oscillator, no WDT, no CP, GP3 = MCLR
;
 __config _XT_OSC & _WDT_OFF & _CP_OFF & _MCLRE_ON

; PER 16F505/526
```

```
 #ifdef___16F526
; Internal oscillator, 4MHz, no WDT, no CP, MCLR
 __config _XT_OSC & _WDTE_OFF & _CP_OFF & _CPDF_OFF & _MCLRE_ON
 #endif
 #ifdef___16F505
; Oscillatore interno, 4MHz, no WDT, no CP, MCLR
 __config _XT_OSC & _WDTE_OFF & _CP_OFF & _MCLRE_ON
 #endif
```

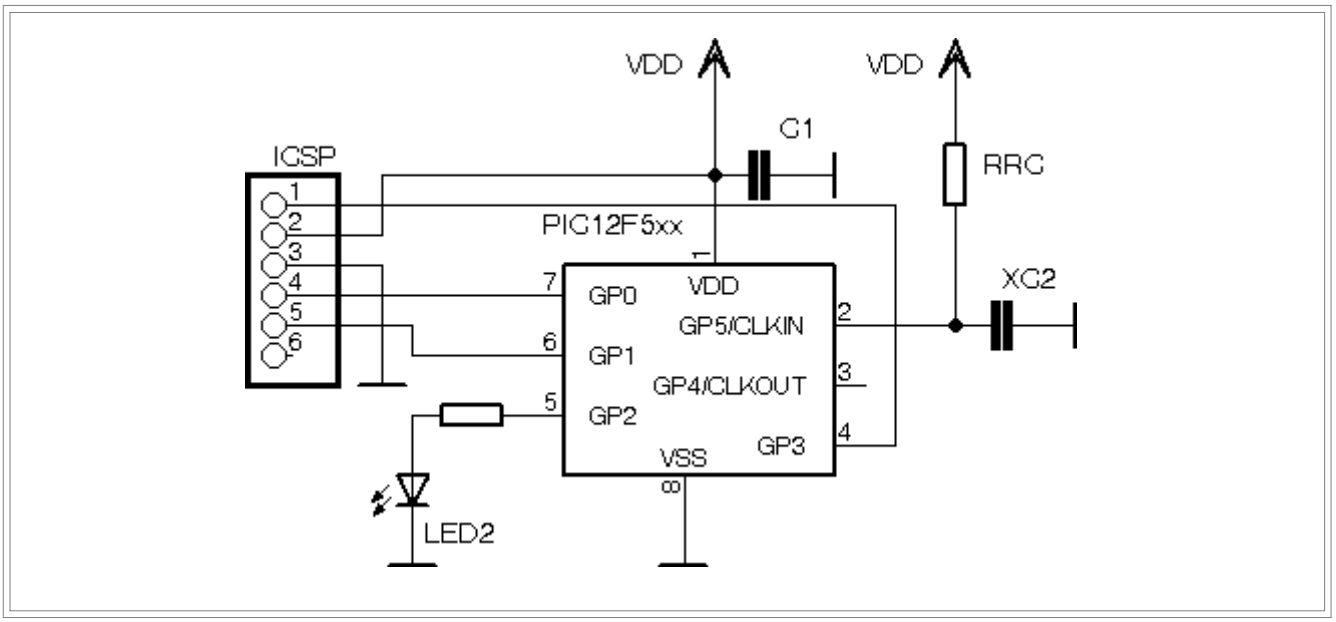
For LP mode , the label _LP_OSC will be used.
For HS mode (only possible on 16F5xx) _HS_OSC will be used.

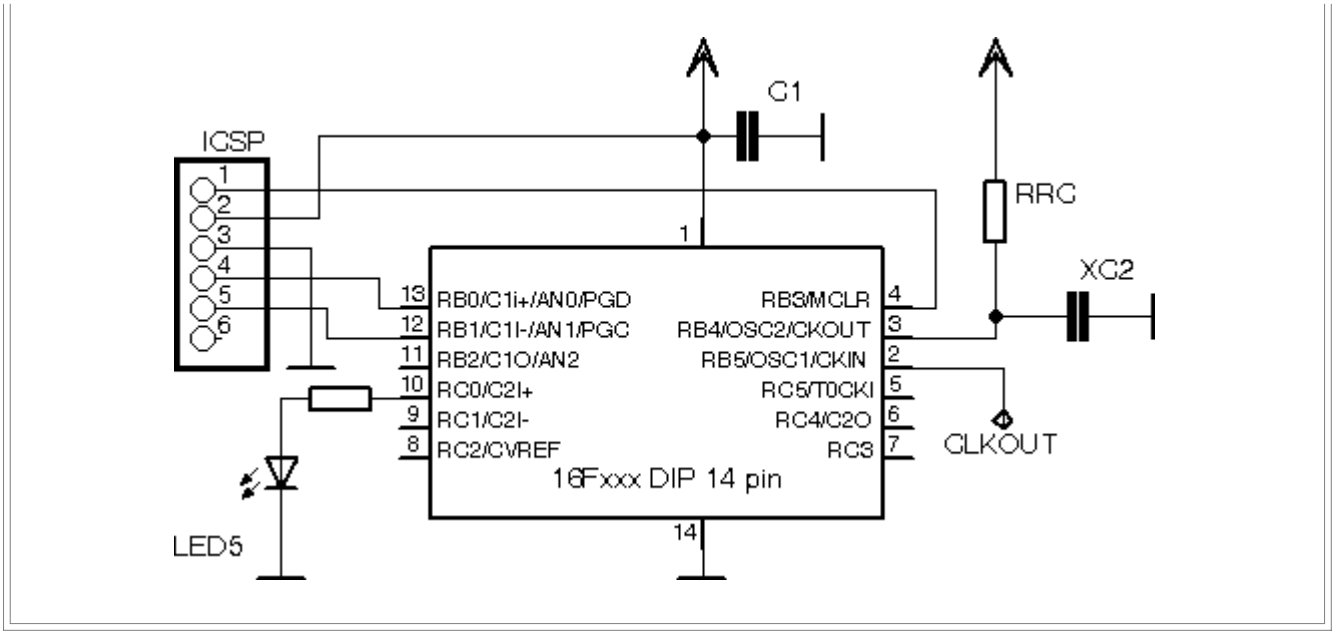
You will not be able to use the **OSC1/OSC2 pins**.

In addition, any time routines will need to be adjusted to the new clock frequency.

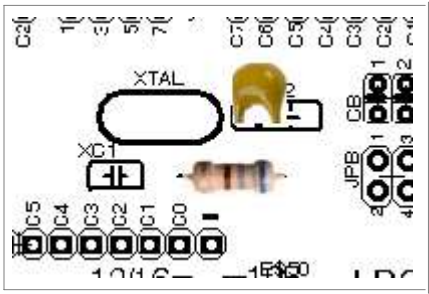
ExtRC Mode

We can experiment with this oscillator as well.





In the case of 14pin PICs we can enable the ExtRC-Clkout mode and check on the pin **`RB5/OSC2`** the output of the generated frequency/4.



The arrangement of the components is identical to the previous one, with the difference that the ***Rext*** (**RRC**) and the ***Cext*** (**XC2**) must be inserted

The ***XC2'***s socket allows you to install 2.54" or 5.08" pitch capacitors. Notice in the drawing the positioning of the capacitor with the narrowest pitch.

Using a 33k resistor, possibly of good quality (metal layer, 1 or 2%) and a 33pF capacitor (ceramic multilayer or plastic film) we should obtain an oscillation frequency of:

T = 2.3 x 33 x 33 = 2504 ns -> 399.5Hz

which corresponds to an instruction cycle of just under 10ms.

Of course, you can use any other combination, adapting the software to the choice made.

The program

Also here the variations consist exclusively in the configuration in which the desired mode for the oscillator is placed, using the _ExtRC_OSC label for the 12F5xx and 16F5xx.

In addition, for the 16F5xx only we can also use **`_ExtRC_OSC_RB4EN`** obtaining the output of the ***Fosc/4*** on the **`OSC2 pin`**.

Obviously, you need to adjust your time routines.

Conclusions

Let's summarize some definitions:

- **Fosc** : ***Frequency of the oscillator*** - the frequency of the oscillator, whatever it may be
- **Fosc/4** : the internal frequency relative to an instruction cycle, equal to 1/4 of the frequency of the oscillator
- **tcyc** : the execution time (cycle) of an instruction

As we can easily understand, the different types of clocks do not affect the instructions except for the different execution time. We remind you that this time is given by:

$$t = 1/FOSC/4$$

To get an idea of the variation:

Fosc [MHz]	FOSC /4 [MHz]	TC YC [US]
0.32768	0.08192	122.07
1	0.25	4
4	1	1
8	2	0.5
16	4	0.25
20	5	0.2

- A change in the clock mode implies the obligation (in addition to the possible need to add external components) of the correct initial configuration. Remember that the configuration is not part of the actual program, but it is essential in the source to fix the characteristics of the oscillator.
- The variation in clock frequency affects only those parts of the program that act through a calculation of time; increasing the ***FOSC*** decreases the execution time, and therefore it is necessary to adjust the time routines or actions that use instruction loops as time elements.

- Increasing the frequency of the oscillator, reducing the execution time of the instructions, allows for greater processor performance.

- The power consumption of the microcontroller is proportional to the frequency. Low-power applications will need to use low working frequencies.

- The working frequency of the microcontroller is proportional to the supply voltage: as the voltage decreases, the possibility of operating with high frequencies is also reduced.

As for the various oscillator modes, where possible, the choice must be made according to these considerations:

- **Internal oscillator** : practically suitable for all applications where the execution time admits a tolerance of around 1%. It has the double advantage of not needing external components and leaving two pins free as I/O.

- **External Crystal Oscillator** : Required only for processors that do not have an internal oscillator, in those applications where execution time requires greater precision and stability in time/temperature or a frequency is required that is not available through the internal oscillator. External components are required and the two pins of the oscillator are taken care of.

- **external oscillator** : to be used if there is already a clock source in the circuit or if it requires stability and precision that can be provided by integrated oscillators (TCXO and similar) or a frequency is required that is not available through the internal oscillator. It occupies a pin for the external clock input.

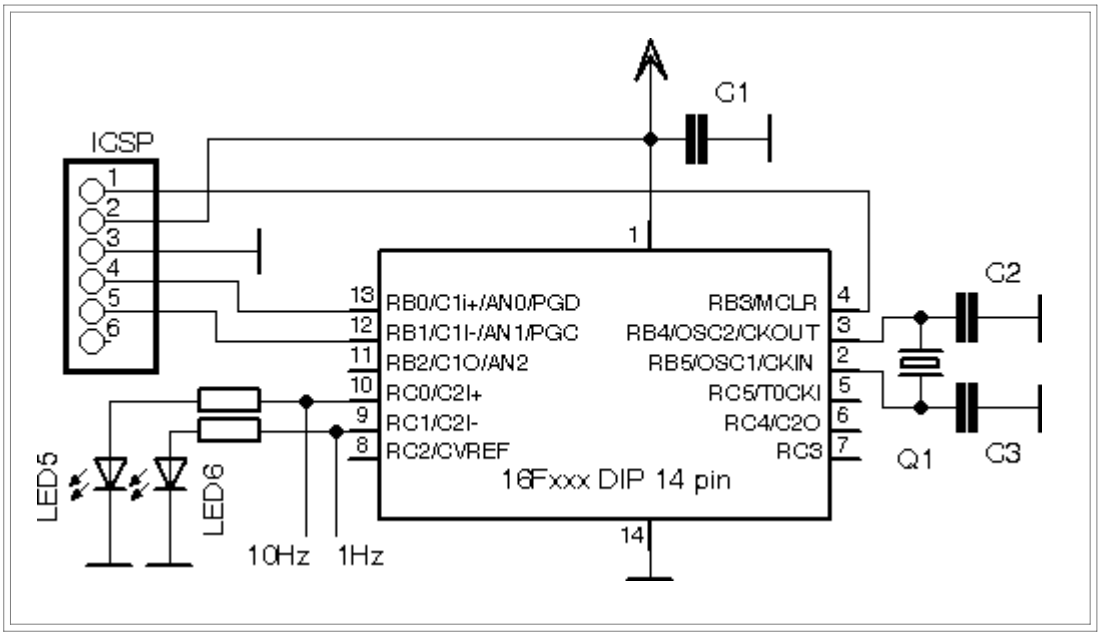
- **RC oscillator** : can be used when accuracy and clock stability are not decisive elements, but a frequency is needed that is not available through the internal oscillator. It occupies a pin for the external clock input.

In any case, the accuracy of the timings in the program depends on the accuracy of the FOSC.

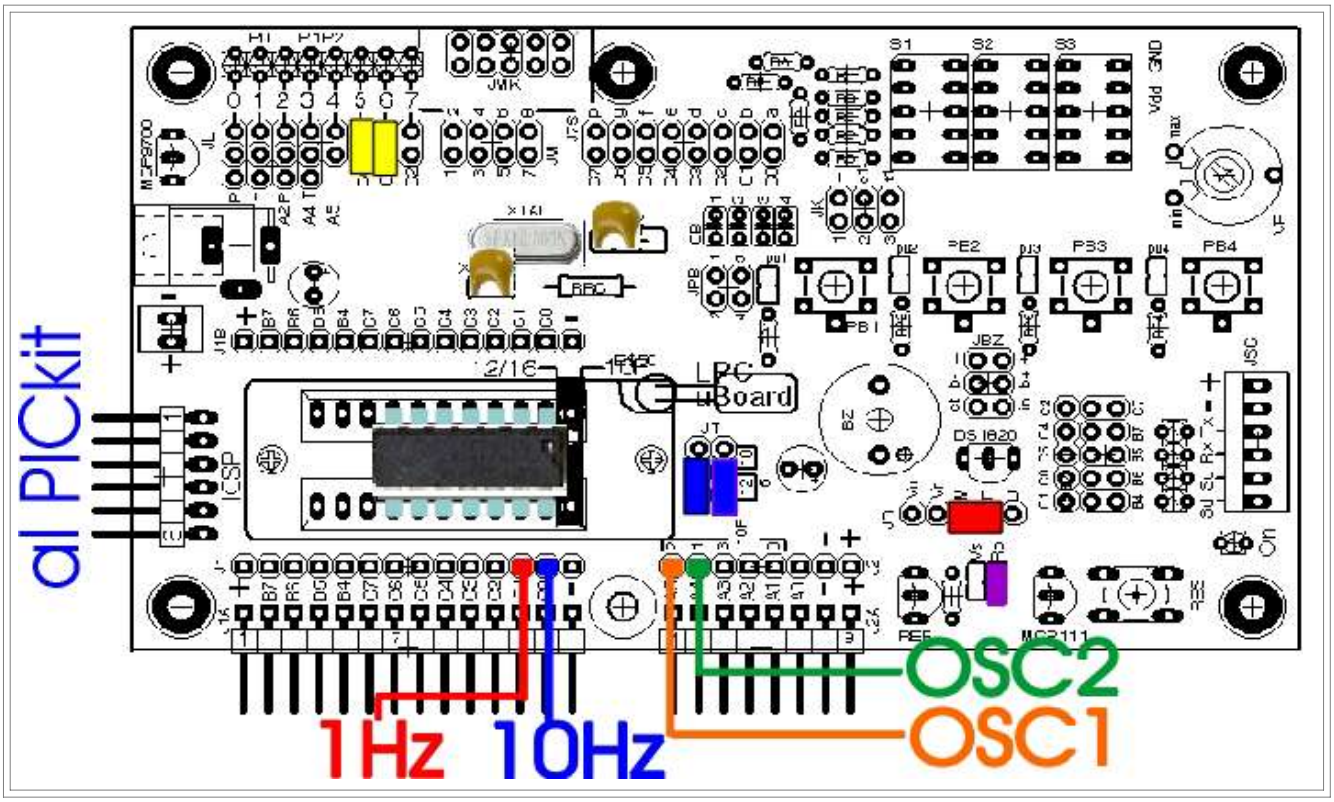
One application: generate 1Hz from 14.318MHz

The 1Hz frequency is useful in RTC applications. Usually a crystal is used in the oscillator to have both high accuracy and good stability over time and with temperature variations.
We use, therefore, an external quartz oscillator.
There are several possibilities to choose for the frequency value of the oscillator: a first one that uses quartz for watchmaking at 32768Hz and another with higher frequency values. Let's first look at the second possibility, perhaps less common than the first, but which still uses a quartz that is very easy to find and low cost.
Among the many values available, we can use the frequency of **14.318000MHz**. Why this seemingly strange value? Simply because it is a crystal of very common use, easily available and cheap, being present in the timebases of PC mainboards. These are typically HC-49 package elements with hole mount or SMD terminals. We will need the first option for a direct installation on the plinths of the **LPCuB**. In addition to the crystal, you need two 15-30pF ceramic capacitors; These are not critical, but they must be of good quality.

The output of the generated frequency of 1Hz is sent to a pin of the **PORTC**. Since the software allows it easily, an output of 10Hz has also been obtained. Two LEDs connected to the outputs indicate that the circuit is operating by flashing. The wiring diagram is similar to the one already seen:



and the **LPCuB**:



The "yellow" jumpers are intended to add LEDs on the outputs for a display of signals even without an oscilloscope.

The program

The source is ***9A_10-1Hz.asm***.
The frequency of 14.318 MHz results in a cycle time per instruction of about 0.28us With a loop of 178975 cycles, a delay of 50ms is obtained. At each time, the state of the 10Hz output pin is reversed, while every 10 the state of the 1Hz output pin is reversed, thus obtaining square waves with the required frequency.

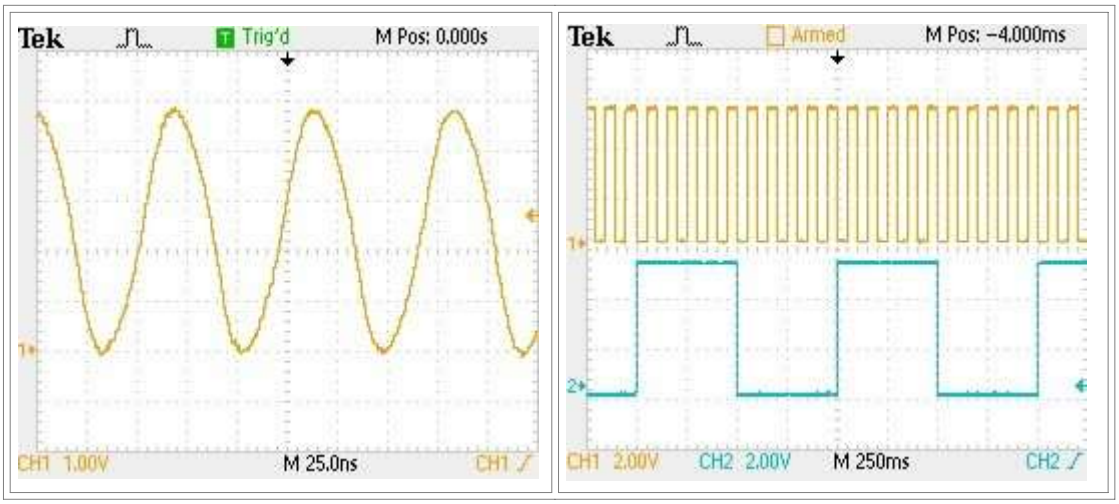
The usual initial phase of adaptation to the chosen processor, it is necessary to select the **HS** (High Speed) mode in the config for the correct operation of the oscillator. In particular, a **`procanalog parameter is defined`** that then allows the selection of the right actions to exclude analog peripherals in those chips that are supplied with them.

The allocation of RAM memory is carried out according to the **`UDATA directive`**.

It simply follows a chain of 10 iterations of the desired delay, which is in the form of a subroutine. At the end of each iteration, the pin value is inverted with an exclusive OR, in order to obtain a symmetrical square wave at the output, with a 50ms semi period. At the tenth iteration, the output is also switched to 1Hz.
Note the first timing which is 2us shorter than the others to compensate for the **`goto`** that closes the loop.

Here's what the waveform of the quartz oscillator at pin OSC2 looks like and the frequencies in

1 and 10Hz output:

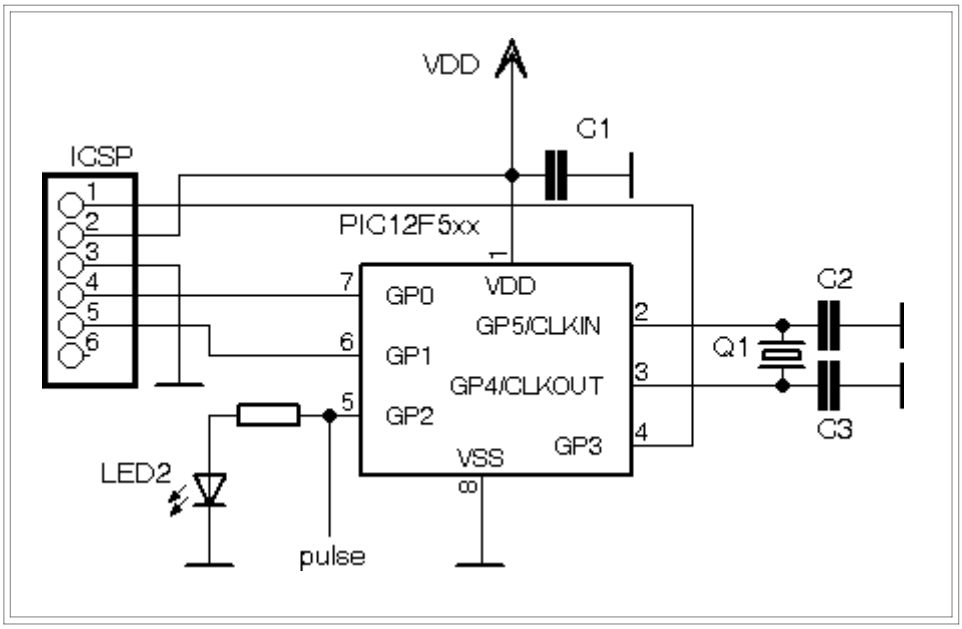


It is not necessary to calibrate the internal oscillator, which is not used (but which is still expressed in the source as a comment to remind you of the need on other occasions). The same goes for disabling T0CKI from RB5, which is defined in the config as the pin of the external oscillator.

Generate 1Hz from 32768Hz

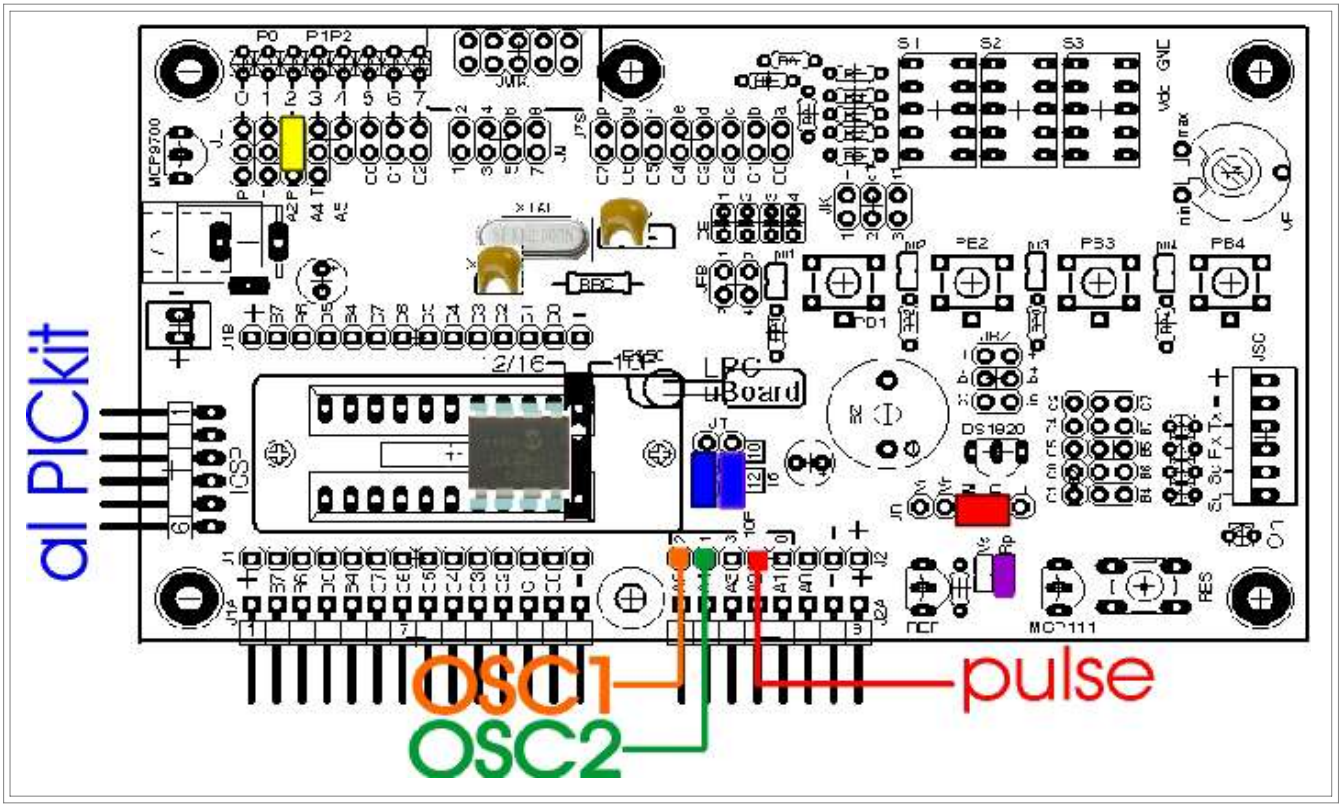
The 8-pin Baselines claim a maximum frequency of 4(8)MHz, so the above circuit is not applicable. A crystal of lower frequency is required. The same program would be adjustable for a 3.5795MHz crystal, which is 1/4 of 14.318MHz; however, this value is not so common, while 32768Hz is. These quartzes can be found from the recovery of old motherboards (where they perform their function as a support for the RTC) or from one of the many electric clocks Made in China, even if a new component, of quality and with a data sheet where you can find the nominal value of the capacitors, is certainly not a high expense.

The application scheme is similar to the previous one.



We output the signal to the GP2, where an LED allows you to display it.

For the hardware on the **LPCuB**:



The "yellow" jumper connects LED 2 to the GP2 output and will flash at the frequency produced. The oscillator waveforms can be taken from the indicated pins, using x10 probes. Quartz and capacitors occupy their respective sockets.

The program

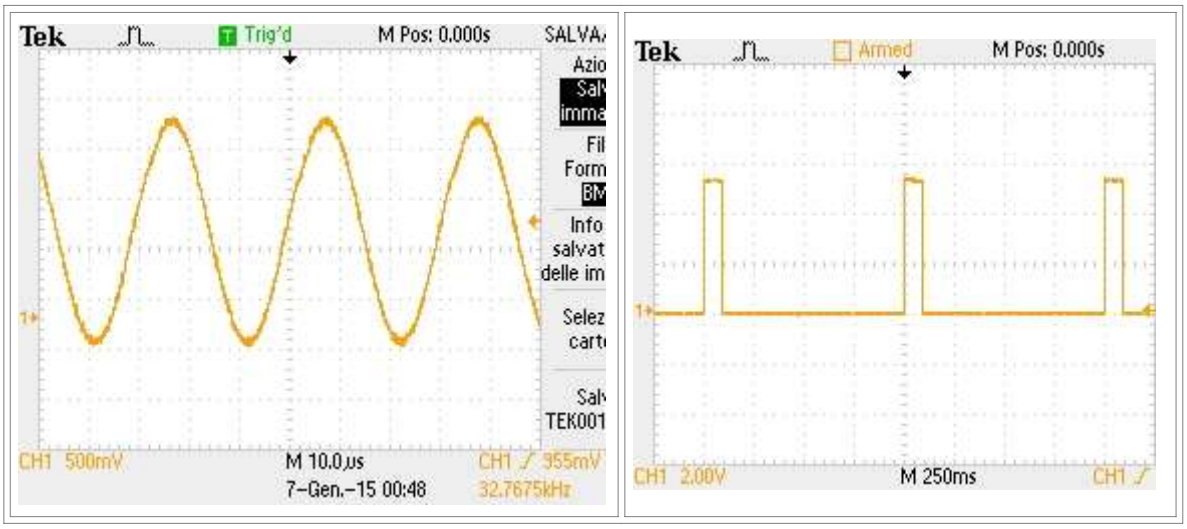
The source is ***9A_1Hz.asm***.

It is necessary to adjust the initial config: the LP (*Low Power*) oscillator mode must be set because it is a low power quartz; an error in the setting of a higher power may not prevent the oscillation, but this occurs in a disorderly way and on a frequency much higher than the nominal one, making the operation incorrect and random. The capacitors can be ceramic, again from 7 to 18pF depending on the crystal used. By inserting capacitors of higher capacitance, for example 47-100pF, oscillation is possible and, in general, more stable, but the start-up time can be considerable (even a couple of seconds!).

The program is based on the use of **Timer0** in free-running: the timer is not loaded and freely counts continuously between 0 and 255.
The value of the oscillation frequency is the key to the simplicity of the program: 32768 is a multiple of 2 (2^15); The logic of the processor is binary and the use of power numbers of 2 greatly simplifies operations.
With ***Fosc = 32768Hz*** we have ***Fosc/4 = 8192Hz***. This means that 8192 instruction cycles are performed for the time of 1 second.
With a prescaler of 32, the overflow cycle of the **Timer0** count will be in exactly 1 second (256 x 32 = 8192). At the end of each overflow, verified in polling, a 100ms high pulse is sent to the output pin. In the meantime, the timer continues to count and the program recycles waiting for a new overflow.

Polling is possible because a value in the TMR0 count register remains for a number of instruction cycles equal to the pre-divisor, allowing the program not to miss the desired event.
The pulse time is evaluated by verifying when **`TMR0`** , in its continuous count, reaches the value of 25, which corresponds to 97.6525ms. An LED connected to the output flashes briefly once per second, following the pulse pattern.
Note that no RAM rental is required.

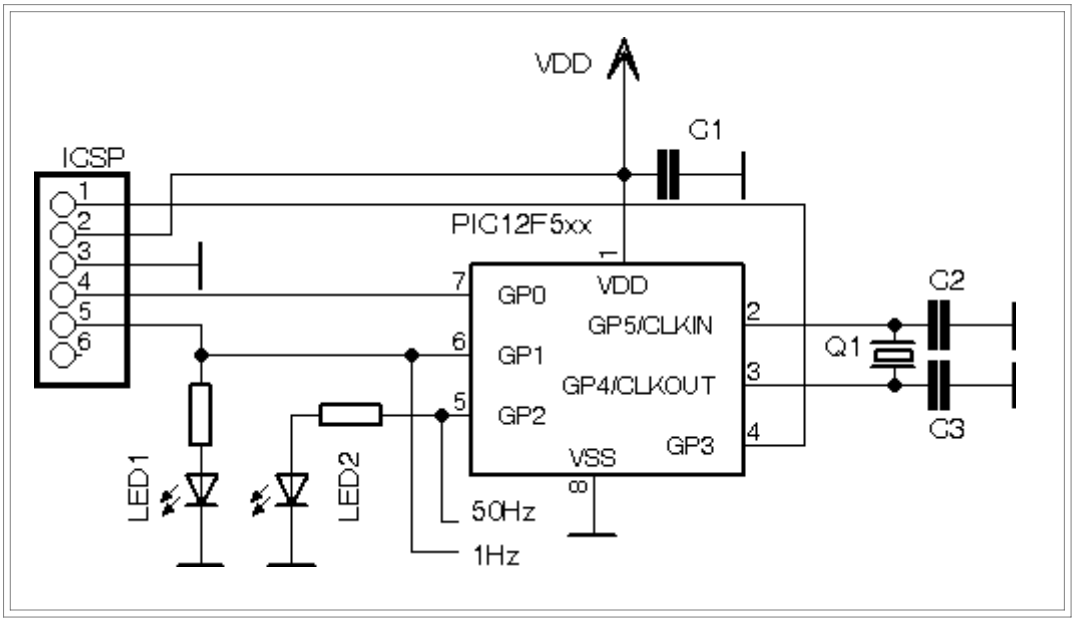
Here's what the waveform of the OSC2 pin quartz oscillator and output look like:



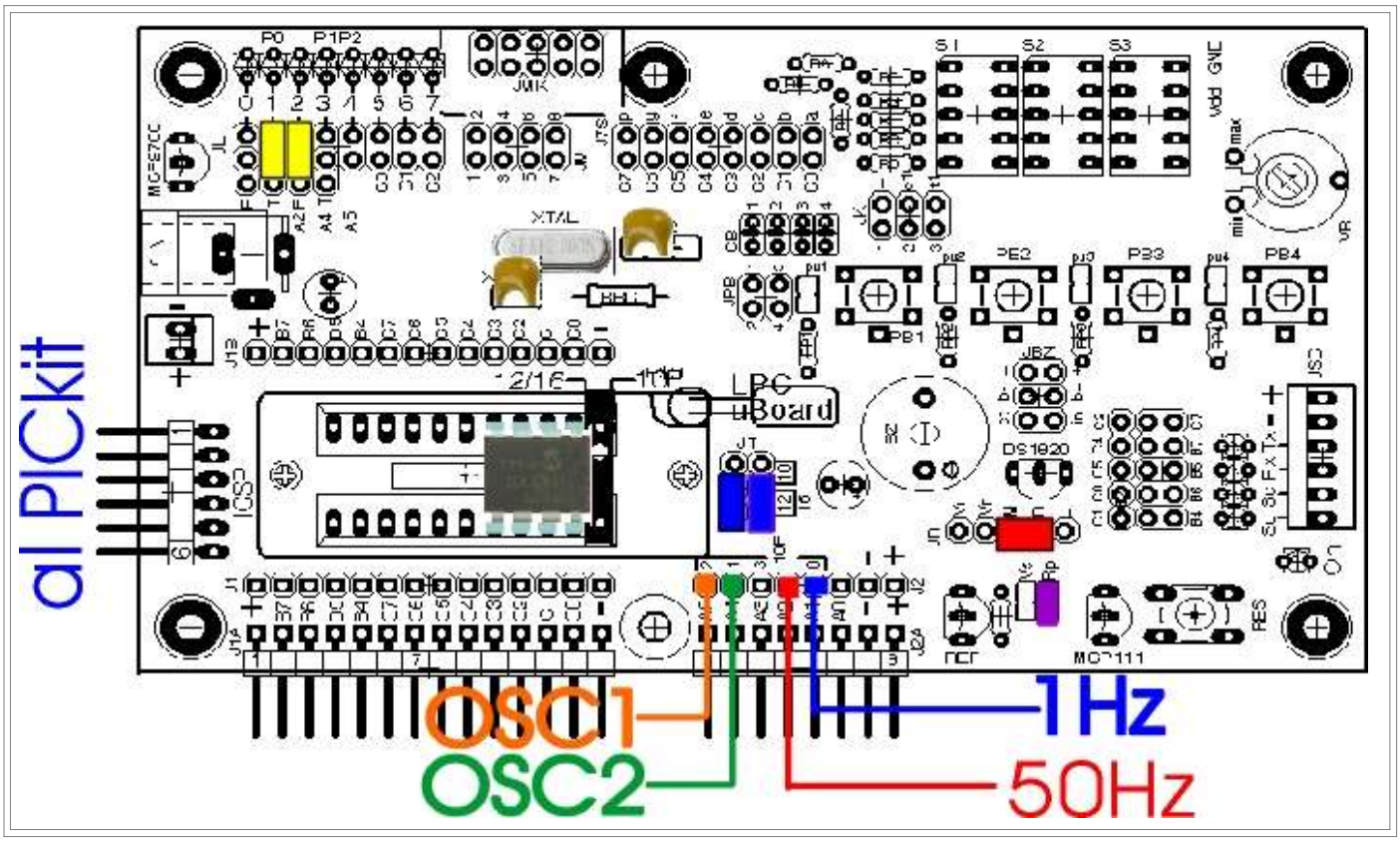
The frequency measured by the oscilloscope is 32767.5Hz, practically the nominal value. The waveform is perfect, indicating that the crystal is working under an ideal load condition; it is obtained with an Abracom quartz and two 10pF capacitors.
In the image on the right you can see the 100ms pulse with a cadence of 1s.

Generate 1Hz and 50Hz starting from 3.2768MHz

Using a 3.2768MHz quartz (multiple of 10 of the previous one) we are still widely enabled to 12F5xx and have a 10 times shorter instruction cycle.
The scheme is similar to the previous one:



to also the arrangement of the components on the **LPCuB**:



The "yellow" jumpers are intended to add LEDs on the outputs for a display of signals even without an oscilloscope.

Nota: LED1 **is connected to** the **`GP1 pin`**, which is also one of the signals of the programming ICSP.
On the LPCuB , the LEDs are very low current (max. 2mA) and do not affect the programming, as they are a negligible load. You will notice the LED1 blinking rapidly during this phase, and then acquire its function once the program is started.
If you are using other hardware, make sure that the current in the LED is minimal or unplug it during programming.

The program

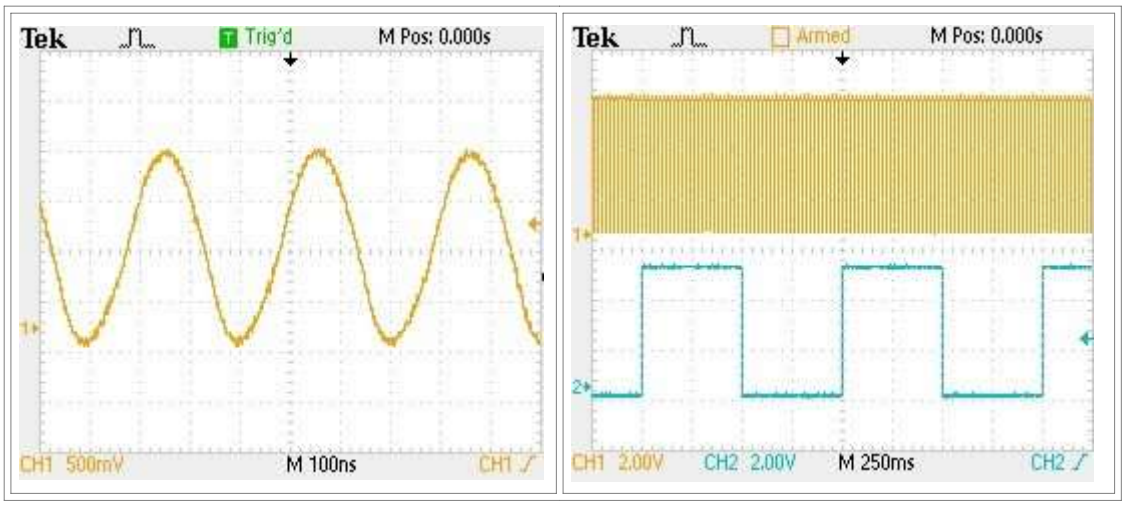
We take advantage of the higher clock frequency (shorter duration of the instruction cycle) with a procedure similar to the previous one to generate a square wave at 50Hz and one at 1Hz: **Timer0** works free running, with a 1:256 pre-divisor.

1. we have ***Fosc = 3276800Hz*** and therefore ***Fosc/4 = 819200Hz***. Every 32 timer counts with a pre divisor at 256 requires 32 x 256 clock cycles, which equals 10.00ms.
2. We then proceed by periodically polling the value of the TMR0 count register and every 10ms (32 counts) we switch the output to 50Hz. On the eighth cycle the counter has reset to zero and starts counting again automatically.
3. Every 50 of these events we switch the 1Hz output to get the corresponding square wave. We can perform the test for the 1Hz output without problems as a count remains in **`TMR0`** for a number of instruction cycles equal to the pre-divisor (here, 256 instructions), allowing the program not to miss the desired event.

Also in this case the program is extraordinarily simple for the possibility of exploiting a clock frequency that does not require operations on counters.

Listing ***9A_50-1Hz.asm*** is for 12F508/509/519.

Here's what the waveform of the OSC2 pin quartz oscillator and the two outputs look like:



The frequency measured by the oscilloscope is 3.2768MHz, the nominal value. The waveform is practically perfect, indicating the oscillator working in an ideal load condition, obtained with a Foxconn quartz and two 22pF capacitors.
You can use quartz with a frequency equal to a power of 2, such as 4.096MHz, 2.048MHz, 1.6384MHz and the like by updating the program cycles.
When using other frequencies (e.g. the 4MHz of the internal oscillator) more complicated firmware may be required to obtain the desired values, usually with less accuracy.

Conclusions

It should be clear how, by using an appropriate value of the oscillator, we can greatly simplify the software to generate precise time values, useful for internal routines or to produce output frequencies, values often not obtainable with the desired precision using inadequate crystals or the internal oscillator.
The use of the timer allows you to perform other operations during the count, taking advantage of the effect of the pre-divider.

9A_10-1Hz.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
; Title         : Assembly & C Course - Tutorial 8A
;                 10Hz and 1Hz generator
;                 External Oscillator 14.318MHz
; PIC           : 16F505/506/526
; Support       : MPASM
; Version       : V.16F5xx-1.0
; Date          : 01-05-2013
; Hardware ref. :
; Author        :Afg
;
;-----------------------------------------------------------------
; ################################################################
; Processor Selection

 #ifdef 16F505
    LIST p=16F505 ; Processor Definition
    #include <p16F505.inc>
 #endif
 #ifdef 16F506
    LIST p=16F506 ; Processor Definition
 #include <p16F506.inc>
 #define procanalog
 #endif
 #ifdef 16F526
    LIST p=16F526 ; Processor Definition
 #include <p16F526.inc>
 #define procanalog
 #endif

   radix dec

;*****************************************************************
;                 DEFINITION OF PORT USE

millis equ 0x01 ; RC0 output 10Hz
according to 0x02 ; RC1 output
1Hz
;
; ################################################################
;                     CONFIGURATION
;
; External Oscillator HS, No WDT, No CP, No MCLR

 #ifdef 16F526
 __config _HS_OSC & _WDTE_OFF & _CP_OFF & _CPDF_OFF & _MCLRE_OFF
 #endif
 #ifdef 16F505
 __config _HS_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif
 #ifdef 16F506
 __config _HS_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif
```

```
; ######################################################################
;                        RAM
;
; general purpose RAM
      UDATA
d1 res 1 ; Counters for delay D2
res 1


; ######################################################################
;                        RESET ENTRY
;
; Reset Vector RESVEC
          TAILS  0x00

; Internal Oscillator Calibration
; movwf OSCCAL

; ######################################################################
;                       MAIN PROGRAM
;
Start
 #ifdef procanalog
; disable ADCON0 CLRF analog
    inputs

; Disable comparators to release the digital function bcf
    CM1CON0, C1ON
    bcf CM2CON0, C2ON
 #endif

; disable T0CKI from PORTB5
         ; b'11011111'
           ; 1------- GPWU disabled
           ; -1------ GPPU disabled
           ; --0----- internal clock
           ; ---1 ---- Falling
           ; ----1--- prescaler at WDT
           ; -----111 1:256
    movlw b'11011111'
    OPTION

; I/O initializations at the reset movlw
    0x03   ; RC1:0 to 1 movf preset
           PORTC

; Assign RC1:0 to movlw b'11111100' digital
    output function
    Tris   PORTC
    Goto   Ml

; ######################################################################
;                       TIME SUBROUTINES
;
; clock 14.318MHz tcyc 0.0002793686s
; Delay 0.05 seconds = 178975 cycles
```

```
; Delay 0.0499988825255 seconds = 178971 cycles
Delay_ms               ; 178971 cycles
    movlw 0xD0         ; +178963 Movwf
    D1 Cycles
    movlw 0x8C
    movwf d2
Delay_ms_0
    decfsz d1, f
     goto $+2
    decfsz d2, f goto
     Delay_ms_0
    goto $+1           ; +4
    goto $+1
    retlw 0            ; +4 cycles (including call)

; Main Loop
ML:
; 1
    Call  Delay_ms     ; 178971 cycles
                       ; +2 loop cycles
    movlw Millis       ; +1
    XORWF PORTC,f      ; +1 = 178975
;
2
    Call  Delay_ms     ; 178971 cycles
    Goto  $+1          ; +2
    movlw Millis       ; +1
    XORWF PORTC,f      ; +1 = 178975
;
3
    Call  Delay_ms     ; 178971 cycles
    Goto  $+1          ; +2
    movlw Millis       ; +1
    XORWF PORTC,f      ; +1 = 178975
;
4
    Call  Delay_ms     ; 178971 cycles
    Goto  $+1          ; +2
    movlw Millis       ; +1
    XORWF PORTC,f      ; +1 = 178975
;
5
    Call  Delay_ms     ; 178971 cycles
    Goto  $+1          ; +2
    movlw Millis       ; +1
    XORWF PORTC,f      ; +1 = 178975
;
6
    Call  Delay_ms     ; 178971 cycles
    Goto  $+1          ; +2
    movlw Millis       ; +1
    XORWF PORTC,f      ; +1 = 178975
;
7
    Call  Delay_ms     ; 178971 cycles
    Goto  $+1          ; +2
    movlw Millis       ; +1
    XORWF PORTC,f      ; +1 = 178975
;
```

```
8
    Call  Delay_ms      ; 178971 cycles
    Goto  $+1           ; +2
    movlw Millis        ; +1
    XORWF PORTC,f       ; +1 = 178975
```

```
; 9
    Call Delay_ms        ; 178971 goto
    cycles $+1           ; +2
    movlw millis         ; +1
    xorwf PORTC,f        ; +1 = 178975
;
10  Call Delay_ms        ; 178971 goto
    cycles $+1           ; +2
    movlw millis|second  ; +1
    xorwf PORTC,f        ; +1 = 178975

    Goto ML              ; Loop Closure - 2 Cycles

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

9A_1Hz.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
; Title         : Assembly & C Course - Tutorial 8A
;                 Generates a 100ms pulse every second on GP2.
;                 Quartz clock 3.2768MHz
; PIC           : 12F508/09
; Support       : MPASM
; Version       : V.508-1.0
; Date          : 01-05-2013
; Hardware ref. :
; Author : afg
;-----------------------------------------------------------------
; Pin use :
; ----------------
; 12F508/09/19 @ 8 pin
;
;              |‾‾\/‾‾|
;   Vdd      -|1    8|- Vss
;   GP5      -|2    7|- GP0
;   GP4      -|3    6|- GP1
;   GP3/MCLR -|4    5|- GP2
;              |______|
;
; Vdd            1: ++
; GP5/OSC1/CLKIN 2: OSC1
; GP4/OSC2       3: OSC2
; GP3/! MCLR/VPP 4:
; GP2/T0CKI      5: Pulse Out
; GP1/ICSPCLK    6:
; GP0/ICSPDAT    7:
; Vss            8: --
```

```
;
;*****************************************************************
;             DEFINITION OF PORT USE
;
; GPIO map
;| 5    | 4 | 3 | 2 | 1 | 0 |
;|------|---|---|---|---|---|
;| LED |    | in |   |   |   |
;
;#define GPIO,GP0 ;
;#define GPIO,GP1 ;
;#define GPIO,GP5 ;
;#define GPIO,GP3 ; Input only
;#define GPIO,P4 ;
#define pulse GPIO,GP2 ; Pulse Output
;
; ###############################################################
;               Choice of #ifdef
 12F509 processor
   LIST p=12F509 ; Processor Definition
 #include <p12F509.inc>
 #endif
 #ifdef 12F508
   LIST p=12F508 ; Processor definition
 #include <p12F508.inc>
 #endif
   radix dec

; ###############################################################
;                 CONFIGURATION
;
; LP External Oscillator, No WDT, No CP, No MCLR
 __config _LP_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF

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

; Internal Oscillator Calibration
;   movwf OSCCAL
; ###############################################################
;               MAIN PROGRAM
;
MAIN:
; Reset Initializations
    clrf GPIO           ; GPIO preset latch to 0

; TRISGPIO   --111011 GP2 out
     movlw b'11111011'
     trip-of-a-kind GPIO    ; To the Management Register

; set Timer0 for prescaler 1:32
; disable T0CKI to have GP2 as digital I/O movlw
      b'11010100'
      OPTION
```

```
; wait overflow
oflww movf TMR0,w
      skpnz
       Goto oflww
; Timer Overflow - Send BSF Pulse
            Pulse     ; pulse = 1
Cy1   movlw .25       ; wait for 25 count = 0.1s
      subwf TMR0,w    ; Ok?
      skpc
       goto Cy1
      Bcf   Pulse     ; pulse = 0

      Goto oflww
;*****************************************************************
;                        THE END
    END
```

9A_50-1Hz.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
; Title         : Assembly & C Course - Tutorial 8A
;                 It generates 50Hz and 1Hz.
;                 Quartz clock 3.2768MHz
; PIC           : 12F508/09/19
; Support       : MPASM
; Version       : V.509-1.0
; Date          : 01-05-2013
; Hardware ref. :
; Author        :Afg
;
;
;-----------------------------------------------------------------
;
; Pin use :
; ---------------
; 12F508/9 @ 8 pin
;
```

```
;           |‾‾\/‾‾|
; Vdd      -|1    8|- Vss
; GP5      -|2    7|- GP0
; GP4      -|3    6|- GP1
; GP3/MCLR -|4    5|- GP2
;           |______|
;
; Vdd 1: ++
; GP5/OSC1/CLKIN 2: OSC1
; GP4/OSC2       3: OSC2
; GP3/! MCLR/VPP 4:
; GP2/T0CKI      5: Out 50Hz
; GP1/ICSPCLK    6: Out 1Hz
; GP0/ICSPDAT    7:
; Vss            8: --
;
;*****************************************************************
; DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;| OSC2| OSC1| in |50Hz | 1Hz | |
;
;#define GPIO,GP0 ;
;#define GPIO,GP5 ; OSC2
;#define GPIO,GP4 ; OSC1
;#define GPIO,GP3 ; Input only
#define hz50pin GPIO,GP2 ; output 50Hz
#define hz1pin GPIO,GP1 ; 1Hz output

hz50 EQU 4
hz1 EQU 2
;
; ###################################################################
; Choice of #ifdef
 12F509 processor
   LIST p=12F509 ; Processor Definition
   #include <p12F509.inc>
 #endif
 #ifdef 12F508
   LIST p=12F508 ; Processor definition
   #include <p12F508.inc>
 #endif
 #ifdef 12F519
   LIST p=12F519 ; Processor definition
   #include <p12F519.inc>
 #endif

   radix dec

; ###################################################################
;                         CONFIGURATION
;
; External Oscillator, No WDT, No CP, No MCLR
 #ifdef _12F519
 __config _XT_OSC & _WDTE_OFF & _CP_OFF & _CPDF_OFF & _MCLRE_OFF
```

```
 #else
 __config _XT_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif

; ########################################################################
; VARIABLES

hzcycle EQU D'50' ; number of cycles per 1Hz


; ########################################################################
; RAM
;
; general purpose RAM
        UDATA     ; start area RAM
hzcntr res 1      ; counter for 1Hz

; ########################################################################
; RESET ENTRY
;
; Reset Vector
RESVEC CODE 0x00

     Bsf   Hz50pin     ; set 50Hz high output
     Bsf   Hz1Pin      ; set 1Hz high output

     movlw B'00111000' ; GP2:0 out
     TRIS GPIO

; set Timer0 for internal clock, 1:256
;           b'11010111'
;             1------- GPWU disabled
;             -1------ GPPU disabled
;             --0----- internal clock
;             ---1 ---- Falling
;             ----0--- prescaler to Timer0
;             -----111 1:256
     movlw b'11010111'
     OPTION

     movlw  hzcycle    ; Pre Initialize Counter    1Hz
     movwf  hzcntr
     CLRF   TMR0       ; clear TMR0

; Fosc = 3.2768MHz
; Timer 1:256
; Cycle 1 count 32 10ms
; cycle 2       64 20ms
; cycle 3       96 30ms
; cycle 4      128 40ms
; cycle 5      160 50ms
; cycle 6      192 60ms
; cycle 7      224 70ms
; cycle 8      256 80ms

Cy1  movlw  D'32'      ; wait for TMR0=32 (10 msec)
     subwf  TMR0,W
```

```
      skpc              ; = ?
       Goto  Cy1        ; Minor - Waiting
      movlw  hz50       ; equals - inverts output 50Hz
      xorwf  GPIO,f
      call   OneHz      ; Verification for 1Hz output

Cy2   movlw  D'64'      ; wait for TMR0=64 (20 msec)
      subwf  TMR0,W
      skpc
       goto  Cy2
      movlw  hz50
      xorwf  GPIO,f
      call   OneHz

Cy3   movlw D'96'       ; wait for TMR0=96 (30 msec)
      subwf  TMR0,W
      skpc
       goto  Cy3
      movlw  hz50
      xorwf  GPIO,f
      call   OneHz

Cy4   movlw  D'128'     ; wait for TMR0=128 (40 msec)
      subwf  TMR0,W
      skpc
       goto  Cy4
      movlw  hz50
      xorwf  GPIO,f
      call   OneHz

Cy5   movlw  D'160'     ; wait for TMR0=60 (50 msec)
      subwf  TMR0,W
      skpc
       goto  Cy5
      movlw  hz50
      xorwf  GPIO,f
      call   OneHz

Cy6   movlw  D'192'     ; wait for TMR0=192 (60 msec)
      subwf  TMR0,W
      skpc
       goto  Cy6
      movlw  hz50
      xorwf  GPIO,f
      call   OneHz

Cy7   movlw  D'224'     ; wait for TMR0=224 (70 msec)
      subwf  TMR0,W
      skpc
       Goto  Cy7
      Movlw  hz50
      XORWF  GPIO,f
      Call   OneHz

Cy8   movf   TMR0,W     ; wait for TMR0=0 (256) (80 msec)
      skpz
       Goto Cy8
```

```
      movlw hz50
      xorwf GPIO,f
      call OneHz

      goto Cy1          ; Loop

;
; SUBROUTINE
;
; OneHz - toggle output 1Hz every 50 cycles of 10ms

OneHz
       decfsz hzcntr,f ; counter=counter-1 = 0?
        retlw 0          ; No - Return
       movlw hz1         ; si - toggle 1Hz out
       xorwf GPIO,f
       movlw hzcycle     ; Presets for Next Count Movwf
       HZCNTR
       retlw 0

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