Esercitazioni PIC Baseline

9A - Il clock

Lo scopo dell' esercitazione è quello di conoscere le possibilità che offre il microcontroller per generare il clock.

Abbiamo visto finora circuiti che utilizzano il clock interno, derivato da un oscillatore RC di precisione integrato nel chip. Questo sistema è estremamente comodo e pratico in gran parte delle applicazioni, in quanto non occorre un valore di frequenza particolare e neppure una precisione migliore dell' 1% offerto dall' oscillatore integrato.

Dobbiamo osservare ancora che il clock interno, in genere fissato a 4MHz, in vari chip dispone:

- della possibilità di assumere altri valori
- della possibilità di essere riportato all' esterno su un pin apposito

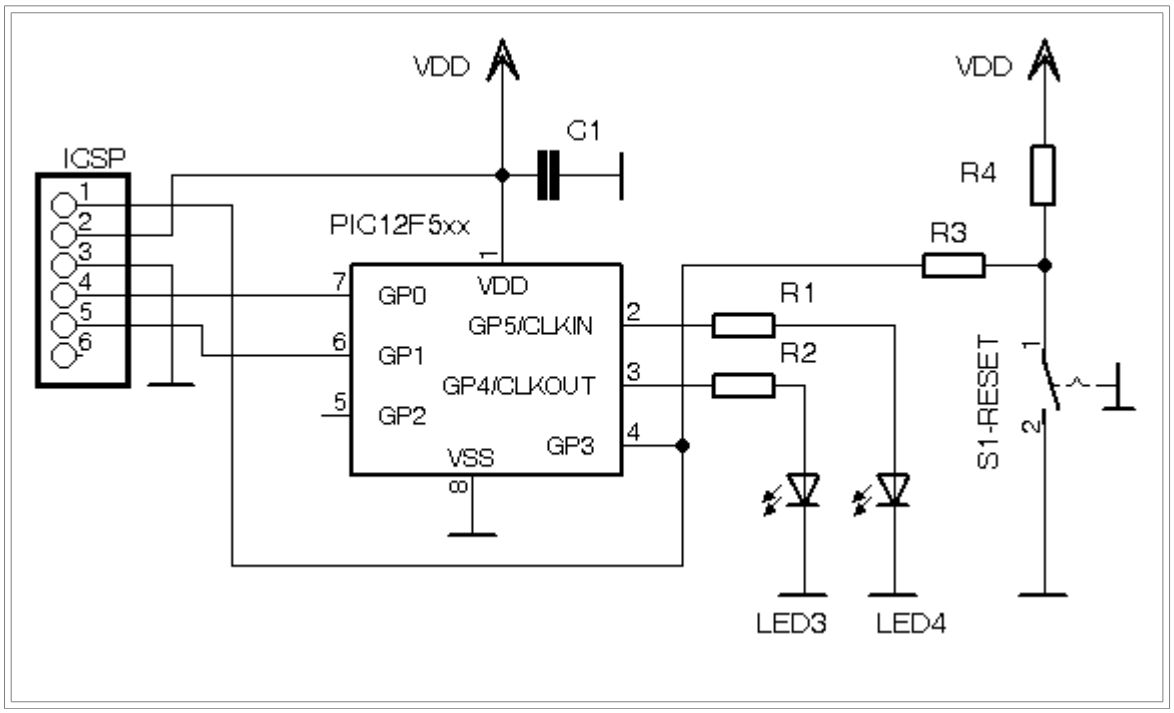
Iniziamo a verificare queste due possibilità.
Innanzitutto va detto che i **PIC Baseline** sono i componenti più "semplici" e non sono dotati di particolari sofisticazioni; solo alcuni chip possiedono le opzioni indicate, mentre nelle famiglie **Midrange** e **PIC18F** sono comuni anche una decina di valori di frequenza alternativi generabili internamente, oltre alla possibilità di oscillatori multipli.
Ora, limitandoci ai Baseline, possiamo osservare che i chip dispongono di clock interno, alcuni con l' opzione di una a doppia velocità e la possibilità di derivare esternamente la frequenza generata, sotto forma di ***Fosc/4***:

PIC	RC interno	Fosc/4
10F200/2/4/6	4MHz	si
10F220/2	4-8MHz	si
12F508/9	4MHz	no
12F510	4-8MHz	no
12F519	4-8MHz	no
16F505	4MHz	si
16F506	4-8MHz	si
16F526	4-8MHz	si

Possiamo, come prima cosa, verificare l' azione del doppio clock sull' esecuzione del programma.

CLOCK a 4 e 8 MHz

riprendiamo un circuito tra quelli già visti, quello dell' esercitazione 2 per il 12F519:



12F508/509 e 16F505 non potranno essere utilizzati, dato che non dispongono del doppio clock interno.

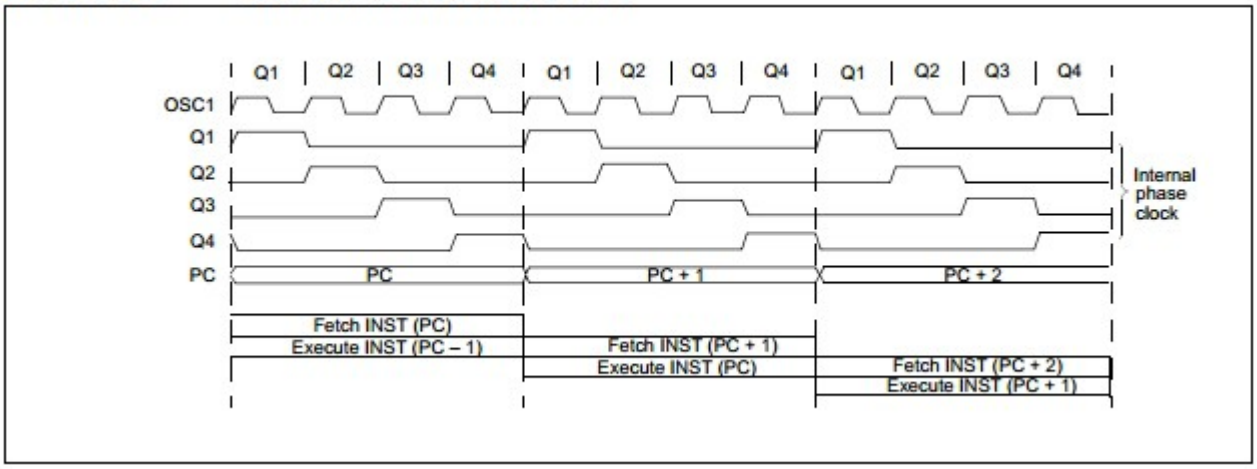
Recuperiamo ora il programma utilizzato per far lampeggiare i due LED dall' esercitazione 2A.

Abbiamo visto che la velocità del lampeggio dipende dalla routine di tempo. E questa, a sua volta, basata su cicli di istruzioni, dipende dal tempo che il processore impiega ad eseguire una istruzione. Esemplifichiamo in una tabella:

Clock	Fosc/4	ciclo istruzione
4MHz	1MHz	1us
8MHz	2MHz	500ns

Questo è dovuto al fatto che l' esecuzione di una istruzione necessità di 4 impulsi del clock

FIGURE 3-3: CLOCK/INSTRUCTION CYCLE



primario, in 4 fasi (Q1, 2 , 3 e 4) che compongono un ciclo-istruzione.

Se il clock raddoppia, passando da 4 a 8MHz, dimezza il tempo di esecuzione, che passa da 1us a 500ns. Questo vuol dire che, se abbiamo realizzato un loop di tempo che a 4MHz impiega 1 secondo per essere eseguito, quando il clock passa a 8MHz il loop impiegherà solo 1/2 secondo.

Come facciamo a modificare il clock?
Se nei PIC18F e negli Enhanced Midrange la commutazione del clock tra i vari valori possibili è gestibile dal programma, nei Baseline questo parametro fa parte delle impostazioni della configurazione iniziale e non può essere modificato durante l' esecuzione del programma.
Dobbiamo scegliere nel config il valore del clock voluto.
Quindi avremo, per 4MHz:

```
;#################################################################
;                        CONFIGURAZIONE
;
; Oscillatore interno 4MHz, no WDT, no CP, pin4=MCLR;
 __config _IntRC_OSC & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_ON
```

e per 8MHz:

```
;#################################################################
;                        CONFIGURAZIONE
;
; Oscillatore interno 8MHz, no WDT, no CP, pin4=MCLR;
  __config _IntRC_OSC & _IOSCFS_8MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_ON
```

Se manteniamo identico il sorgente, selezionando il clock a 8MHz, il lampeggio dei LED raddoppierà in frequenza.

Per ottenere la stessa cadenza, dobbiamo modificare la subroutine di tempo:

```
;#################################################################
;                        CONFIGURAZIONE
;
; Oscillatore interno 8MHz, no WDT, no CP, pin4=MCLR;
  __config _IntRC_OSC & _IOSCFS_8MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_ON
```

Se applichiamo il resto del programma senza modifiche, otteniamo una frequenza di lampeggio dei LED a velocità doppia. Per avere la cadenza di 1Hz, abbiamo due possibilità: la prima è quella di ripetere due volte la chiamata alla subroutine di tempo:

```
mainloop:
; accende LED
  LED_ON

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

; spegne LED
  LED_OFF

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

; loop
  goto    mainloop
```

La seconda possibilità è quella di modificare la subroutine, adeguandola al nuovo clock:

```
;#################################################################
;=================================================================
;=                        SUBROUTINES                            =
;=================================================================
; Delay = 1 secondo @ Clock = 8 MHz
; Fosc/4 = 500ns , 1 secondo = 2000000 cicli
Delay1s_8:            ; 1999996 cicli
  movlw   0x11      ; inizializza 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
; fine conteggio 1999996 cicli - ora somma i 4 mancanti
  retlw    0          ; 4 cicli compresa call
```

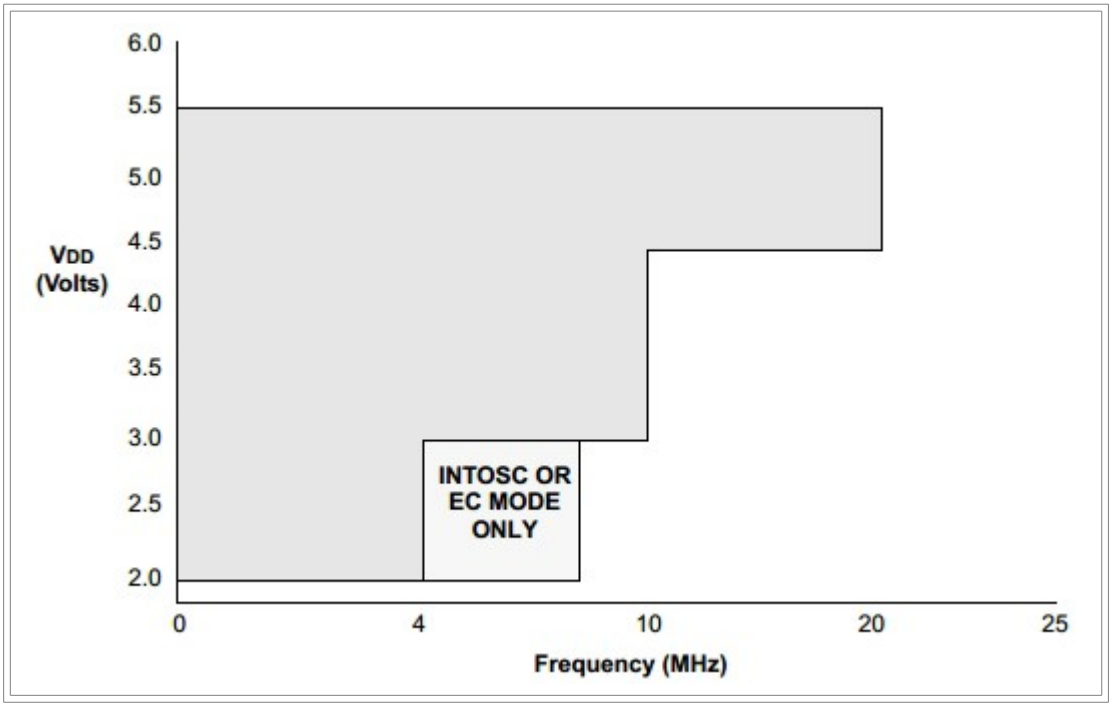
L' algoritmo, del tutto simile al precedente, è derivato dal solito sito ***Delay Code Generator*** ; notiamo che la differenza sta esclusivamente nei valori di pre carica dei contatori per adeguarli al tempo di ciclo dell' istruzione che è passato da 1us a 500ns.

Frequenza e tensione Vdd

E' evidente che la possibilità di avere un chip ad una frequenza fissa è limitante rispetto alla disponibilità di più valori, dato che una maggiore frequenza consente di avere tempi di esecuzione più veloci per potersi adeguare alle necessità del processo controllato (comunicazioni seriali, misure, conversioni AD, ecc.).

Peraltro, va ben compreso che non è un obbligo mandare il chip alla massima velocità possibile, in quanto, per l'esecuzione del programma, può essere sufficiente un clock più basso.
Questo è legato al fatto che il consumo di energia aumenta in modo quadratico con l' aumentare della frequenza; ne deriva che una applicazione a basso consumo dovrà avere un clock minimo e, in ogni caso, abbassando la frequenza dell'oscillatore avremo un risparmio di energia.

Inoltre, alte frequenze di lavoro sono possibili solo in proporzione diretta alla Vdd: i PIC, in generale, possono lavorare tra 2 e 5.5V di Vdd, ma ad essa è legata la massima frequenza possibile, secondo il diagramma seguente:

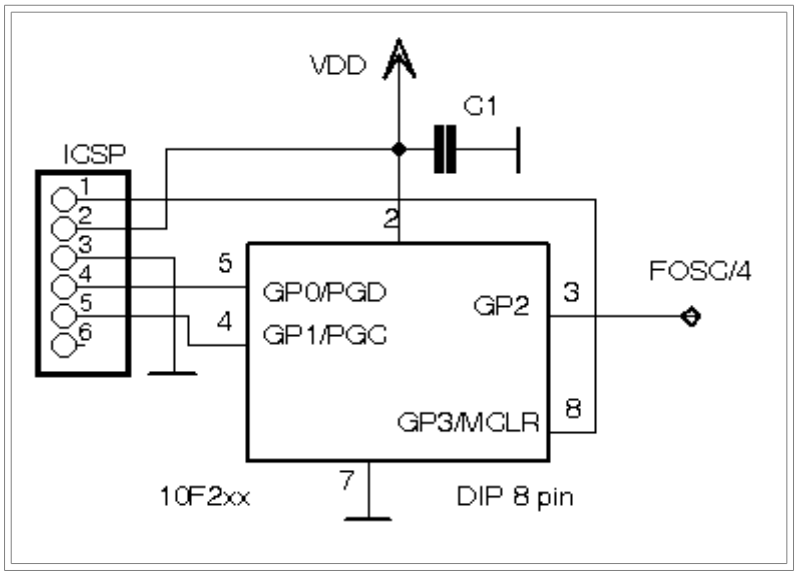


La tabella indica il rapporto tipico tra la Vdd e la frequenza di lavoro: al di sotto dei 4.5V non è possibile operare con sicurezza oltre i 10MHz. Al di sotto dei 3V occorre non superare i 4MHz.
La caratteristica indicata è quella della fig.14-1 del foglio dati del PIC16F526, ma è estendibile a tutti i Baseline, e, con poche variazioni, a tutti i PIC.

FOSC/4

La verifica della funzione di uscita **Fosc/4** è riservata a chi dispone di un oscilloscopio o di almeno di un probe per individuare i livelli logici.
Nei Baseline alcuni chip dispongono di questa opzione che consiste nel poter inviare ad un pin il segnale del clock interno diviso 4, ovvero la frequenza del ciclo di istruzione.
Questa possibilità non è di uso comune, ma può servire a componenti esterni, fornendo loro un clock sincronizzato con quello del processore.

Possiamo usare uno schema extra semplificato:



Che il microcontroller non abbia alcunchè collegato, se non ICSP e l' alimentazione dipende dal fatto che quello che ci importa verificare è l' uscita della frequenza di clock. Questa richiede che un pin sia dedicato a questo scopo ed è, quindi, una scelta di base, da effettuare con una azione sul bit 0 del registro **OSSCAL**

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

Portando questo bit a 1, la frequenza generata internamente sarà applicata al pin **GP2**, che, quindi, non potrà essere utilizzato per altri scopi:

```
; calibrazione oscillatore interno e abilitazione FOSC/4
          iorlw 1
```

```
          movwf OSCCAL
```

Al **POR**, come abbiamo visto, il registro **W** contiene il valore di calibrazione dell' oscillatore. L' **OR inclusivo** `iorlw` porta il bit 0 di W a 1 e con questo abilita la funzione.

```
; calibrazione oscillatore interno e abilitazione FOSC/4
          iorlw 1
          movwf OSCCAL
; blocco esecuzione
          goto  $
; fine compilazione
    END
```

Una volta impostato il modo di oscillatore voluto, non occorre altro: applicando la sonda dell' oscilloscopio, oppure il probe logico, o un frequenzimetro al **GP2** potremo rilevare il segnale ***Fosc/4***.
Per inciso, se disponete di un frequenzimetro e volete verificare l' effetto della calibrazione, potete provare escludendola del tutto:

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

e inserendo altri valori diversi da quello di calibrazione, ad esempio quelli estremi della gamma di aggiustamento:

```
; valore minimo di calibrazione
            movlw   b'10000000'
;abilitazione FOSC/4
            iorlw   1
            movwf   OSCCAL
```

e:

```
; valore massimo di calibrazione
            movlw   b'01111110'
;abilitazione FOSC/4
            iorlw   1
            movwf   OSCCAL
```

il che permette di verificare il range di variazione possibile rispetto alla frequenza nominale.

Può essere considerata anche l'eventualità che non si desideri attivare l' uscita Fosc/4, dato che essa sottrae un pin ad altre funzioni. Questa condizione dovrebbe essere già predisposta dal valore di calibrazione dell'oscillatore impostato in fabbrica, che ha sempre bit0 = 0.
Se, però, presumiamo che il valore di calibrazione sia stato cancellato o corrotto, esiste la possibilità che il bit0 sia a 1. Normalmente questo non succede, in quanto i dispositivi di programmazione,

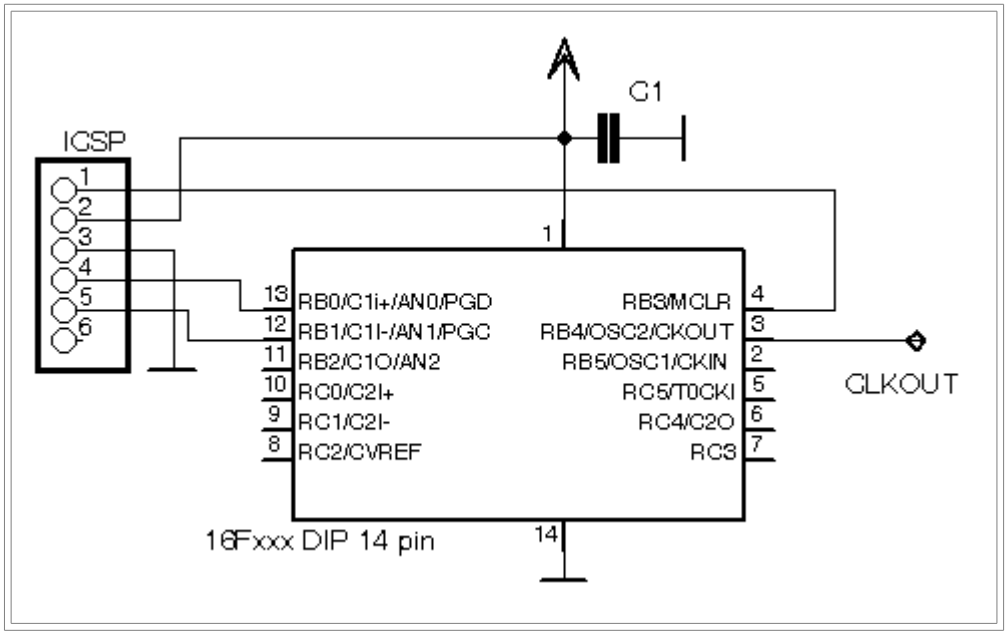
connettendosi al PIC, segnalano se il valore di calibrazione è errato e recuperare quello adeguato è cosa molto semplice, con l'apposita utility dei Pickit Programmer.
Volendo, però, inserire nel programma una ulteriore sicurezza:

```
; calibrazione oscillatore interno con sicurezza per la
; disabilitazione di FOSC/4
          andlw b'11111110'     ; assicura bit0=0
          movwf OSCCAL
```

il che permette di avere con sicurezza, in ogni caso, Fosc/4 disabilitata.

Per quanto riguarda i **12F Baseline**, nessuno di essi prevede l' uscita del clock, mentre è possibile nei 16F Baseline.

L'opzione non fa riferimento a ***FOSC/4***, ma alla modalità RC interno con uscita del clock sul pin **OSC2** (**CLKOUT**), dove, però, non dipende dal valore di calibrazione, ma dal config iniziale.
Anche qui non occorre altro che l' alimentazione e uno strumento per verificare la frequenza in uscita.



Occorre sempre agire in fase di config:

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

Anche qui potremo fare le stesse verifiche viste in precedenza

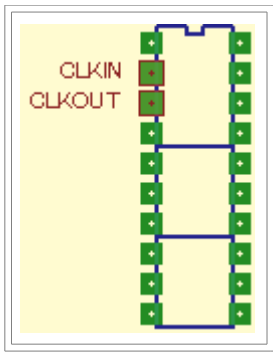
Oscillatori esterni

Oltre alle modalità interne, la maggior parte dei chip offre la possibilità di:

- applicare esternamente la frequenza di clock
- utilizzare componenti esterni per avere un oscillatore a cristallo
- utilizzare componenti esterni per avere un oscillatore RC

<table>
<tr><th rowspan="2">PIC</th><th rowspan="2">RC interno</th><th rowspan="2">Fosc/4</th><th colspan="3">Oscillatore esterno</th></tr>
<tr><th>Quarzi - max.</th><th>RC</th><th>EC</th></tr>
<tr><td>10F200/2/4/6</td><td>4MHz</td><td>si</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>10F220/2</td><td>4-8MHz</td><td>si</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>12F508/9</td><td>4MHz</td><td rowspan="3">no</td><td>4MHz</td><td rowspan="6">si</td><td>-</td></tr>
<tr><td>12F510</td><td>4-8MHz</td><td>8MHz</td><td>-</td></tr>
<tr><td>12F519</td><td>4-8MHz</td><td>8MHz</td><td>-</td></tr>
<tr><td>16F505</td><td>4MHz</td><td rowspan="3">si</td><td rowspan="3">20MHz</td><td rowspan="3">si</td></tr>
<tr><td>16F506</td><td>4-8MHz</td></tr>
<tr><td>16F526</td><td>4-8MHz</td></tr>
</table>

Vediamo innanzitutto che sono esclusi da queste possibilità i PIC10F; la ragione è semplice: per un oscillatore esterno servono un paio di pin da dedicare e in chip con soli 4 I/O disponibili la possibilità non ha senso.
Invece, i chip da 8 pin in su possono sfruttare un oscillatore esterno (sempre a scapito della perdita di un paio di pin come I/O), ma con frequenza massima paria a quella dell' oscillatore interno.
N ei chip a 14 pin, la frequenza dell' oscillatore esterno può arrivare a 20MHz, anche se quella interna è solo 4 o 8.
E' evidente che un oscillatore realizzato con l' aiuto di componenti esterni richiede di avere disponibili pin del chip per collegarli al circuito interno. Nei Baseline questi pin sono identificati come **`OSC1/OSC2`** o **`CLKIN/CLKOUT`**; essendo **sovrapponibili pin to pin**, la disposizione è questa:



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

Le opzioni del clock e il config

Le possibili opzioni di funzionamento del clock sono descritte nei fogli dati dei singoli componenti, ma è possibile averne rapidamente una lista consultando non l' intero data sheet, bensì i soli files ***nomeprocessore.inc*** relativi ai vari chip. Abbiamo visto come questo file sia l' elenco delle label pre definite delle funzioni principali del chip e che questo file sia leggibile con un qualsiasi editor. Nella sezione **_CONFIG** troviamo come prime voci le modalità dell' oscillatore. Ad esempio, per **16F526** abbiamo questa situazione:

```
;----- 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
```

Riassumendo, sono disponibili i seguenti modi:

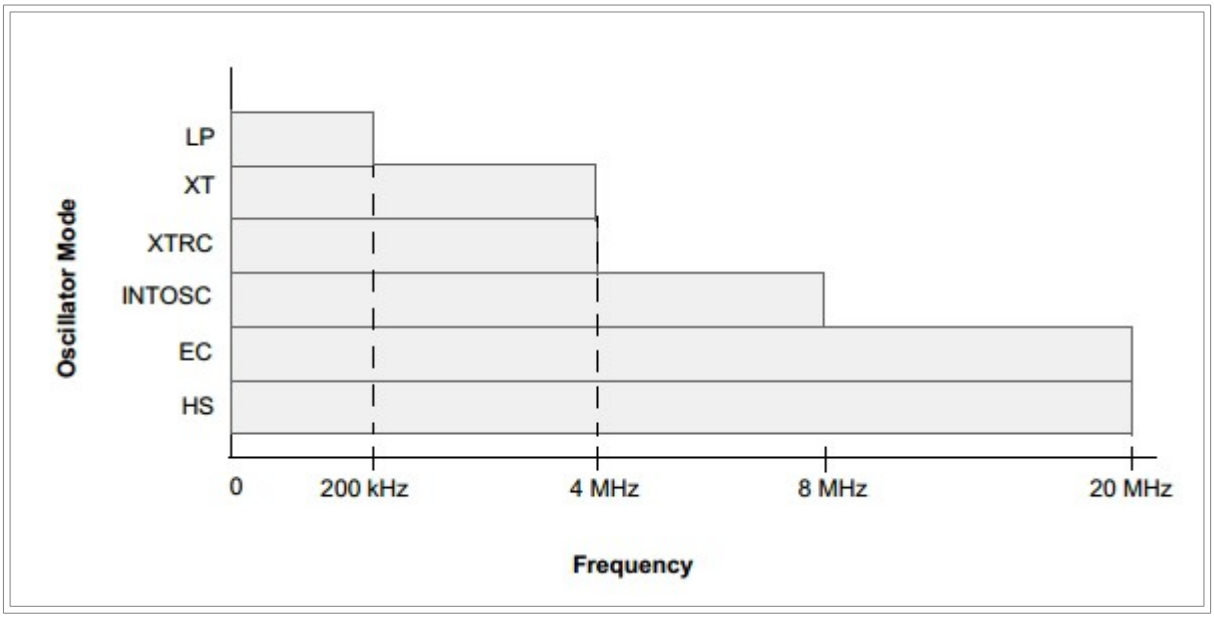
- **LP, XT, HS** - Oscillatore esterno a cristallo
- **EC** - Sorgente di clock esterna
- **ExtRC** - Oscillatore esterno RC
- **IntRC** - Oscillatore interno

Gli ultimi due modi possono avere una uscita Fosc/4.
Possiamo stilare una tabella per i chip che consideriamo in queste esercitazioni:

PIC	Quarzi			Esterni			Interni	
	LP	XT	HS	EC	RC	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

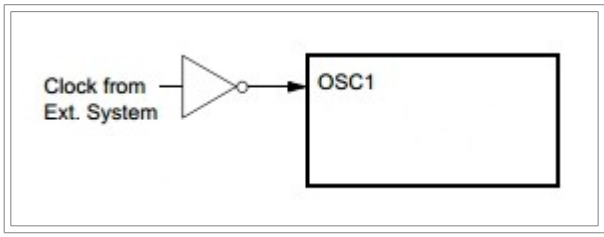
I fogli dati ci offrono un grafico riassuntivo del rapporto tra modalità di funzionamento dell' oscillatore e frequenza ottenibile, che vale la pena di riportare:



Vediamo ora le caratteristiche delle singole opzioni.

Clock esterno – EC

Possiamo applicare una frequenza generata esternamente al pin **`CLKIN/OSC1`**.

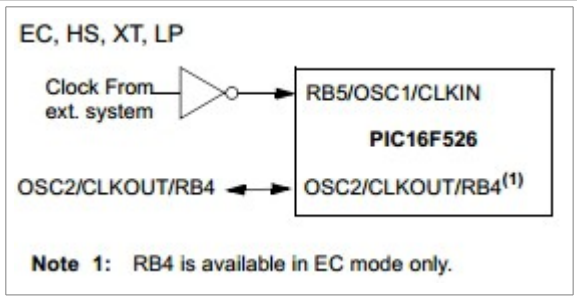


In questo caso, il pin viene predisposto dal config per ricevere il segnale esterno che dovrà avere caratteristiche compatibili con il microcontroller (livelli TTL, tempi di salita e discesa indicati nel foglio dati).

Solamente alcuni chip dispongono di questa opzione, che viene scelta inserendo nel config la voce:

```
 __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
```

Per i Baseline l' opzione **EC** (***External Clock***) corrisponde al fatto che il pin **`OSC1/CLKIN`** diventa l' ingresso per la frequenza esterna e non può assumere altra funzione.

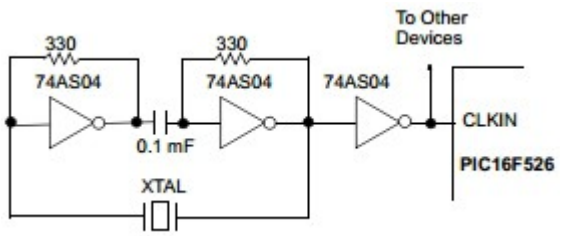


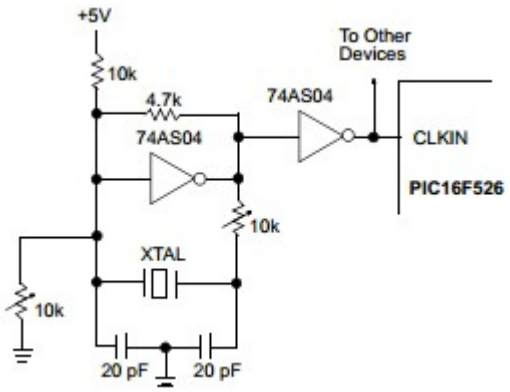
Invece, il pin **`OSC2/CLKOUT`** è assegnabile come I/O. Nell' immagine a lato, riferita al 16F526, ma valida per gli altri 16F Baseline, si tratta di **`RB4`**.

Questo modo non è di uso comune, ma è applicabile dove esiste già nel sistema un generatore di clock, ad esempio quando il PIC fa parte di una scheda complessa assieme ad altri processori.

Oppure quando è necessaria una specifica frequenza che viene prodotta da un oscillatore apposito, di cui ne esistono moltissimi modelli con caratteristiche di precisione e stabilità nel tempo e con la temperatura e dimensioni diverse.
L'uso di questi componenti è solitamente dettato dalla necessità di elevate precisioni e stabilità nel tempo e con la temperatura.

Nel caso in cui non si impieghi un oscillatore integrato, ma si realizzi con gate, occorre utilizzare TTL (LS, AS, ecc) o CMOS HCT per avere il giusto andamento del segnale.



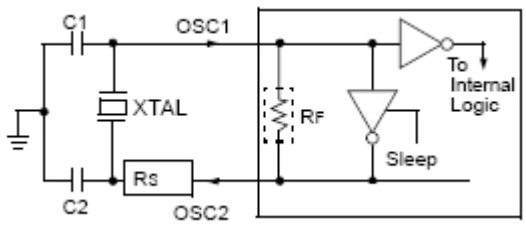


Rispetto al clock integrato, una fonte esterna di clock può servire più processori o altri circuiti.
Rispetto ai 4/8MHz, il clock esterno, per i PIC16F5xx, può salire a 16-20MHz, con il corrispondente aumento di prestazioni: se a 4MHz una istruzione è eseguita in 1 us, a 20MHz richiede solo 200ns.

Il foglio dati del componente specifica la frequenza massima applicabile in funzione della tensione di alimentazione, che, come abbiamo visto, è proporzionale.

Clock esterno - LP, XT, HS

Possiamo realizzare un oscillatore controllato a cristallo (quarzo o oscillatore ceramico), da pochi kHz a un paio di decine di MHz, aggiungendo alcuni componenti esterni:



La struttura interna dei gate corrispondenti ai pin **`OSC1/OSC2`** viene modificata al config in modo da realizzare un oscillatore che usa come elemento di stabilità e frequenza un cristallo di quarzo oppure un elemento ceramico. Occorre aggiungere la coppia di condensatori ***C1/C2***: solitamente si tratta di ceramici NP0 tipicamente da 18-22pF.
La resistenza ***Rs*** va aggiunta utilizzando cristalli low power per limitare la corrente.

La resistenza ***Rf*** il cui scopo è portare il gate in zona di funzionamento lineare, è integrata ed ha un valore di circa 10Mohm. Solitamente non ne occorre una esterna .
Da notare che questa opzione impegna entrambi i pin **`OSC1/OSC2`** che, pertanto, non potranno avere altra destinazione. Esistono tre modi con si configura il gate dell' oscillatore:

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

L' opzione **LP** (***L**ow **P**ower*) viene scelta inserendo nel config la voce:

```
 __config   __FOSC_LP    ; LP oscillator
oppure
 __config   _LP_OSC      ; LP oscillator
```

per cristalli fino a 200kHz. Tipicamente si tratta di quarzi a **32768Hz** usati per generare tempi precisi, orologi, data logger e simili. Questo genere di cristalli funzionano con una potenza di 1-2 microwatt, mentre la maggior parte dei quarzi HF possono dissipare da 100 a 500 microwatt. Per questo motivo, quando è necessario limitare la dissipazione di potenza: un eccesso di corrente può non impedire l' oscillazione, ma questa avviene su armoniche elevate ed è instabile e casuale, oltre

al fatto che una driving con potenza eccessiva accorcia la vita del cristallo. Da qui potrebbe rendersi necessaria la ***Rs***, principalmente per elementi ultra low power del genere usato in orologeria.
Nella pratica, la scelta dei componenti per l' oscillatore LP può essere laboriosa e può non essere immediato trovare un cristallo adeguato al funzionamento su OSC1/OSC2 in LP, sopratutto tra i recuperi da smontaggio di schede vari; inoltre, la frequenza generata è legata al valore dei condensatori in modo molto più deciso che per i modi HS e XT, che non sono troppo critici.
Per LP si va da 6-12pF a un centinaio e può essere necessario avere valori diversi per C1 e C2 per ottenere la giusta frequenza di oscillazione.
Anche se dai condensatori dipende molto la frequenza generata, usando valori inadeguati, se il quarzo è adatto, il funzionamento difficilmente può mancare; con valori dell' ordine di 33-100pF, l' oscillazione può essere anzi più stabile, ma il tempo di start-up (avvio) dell' oscillatore può diventare sensibilmente elevato (anche secondi !).

L'opzione **XT** (***XT****al*) richiede:

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

e va usata con cristalli o risonatori ceramici da 200kHz a 4MHz. Rispetto al modo precedente, cambia l'energia disponibile per l' oscillatore, che viene aumentata. I condensatori, tra 18 e 47pF non sono particolarmente critici, ma occorrono elementi di buona qualità.

L' opzione **HS** (***H****igh* ***S****peed*) richiede:

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

E' il modo da adottare per cristalli da 4MHz in su, dove l' energia richiesta dall' oscillatore è massima (la già citata necessità di maggiore energia con l' aumentare della frequenza).
Da osservare che i **12F5xx** non supportano questo modo: la loro frequenza massima operativa, in ogni caso, è indicata a 4MHz. Diversamente, i PIC **16F5xx** dispongono del modo **HS**, dato che possono arrivare a 16-20MHz.

Comunemente, **XT** e **HS** sono quelli più impiegati: la scelta di uno o dell' altro dipende dal tipo di cristallo usato e va definita, assieme ai valori dei condensatori, per ottenere una oscillazione stabile in tutta la gamma di tensioni di alimentazione e temperature a cui il microcontroller sarà sottoposto.
E' possibile che uno stesso cristallo possa funzionare in entrambi i modi, ma probabilmente uno dei due darà una forma d'onda ed una stabilità migliore.

Alcune note:

- una volta impostato un modo dell' oscillatore con componenti esterni, questi dovranno essere cablati come previsto, pena il mancato funzionamento del microcontroller.

- scegliendo nel config un modo inadeguato ai componenti usati è probabile che l'oscillatore non si avvii o che funzioni in modo casuale e su frequenze diverse da quella nominale

- elevata stabilità si ottiene solo utilizzando componenti di qualità. I condensatori dovranno essere ceramici a coefficiente di temperatura neutro (NP0), le resistenze a strato metallico e il cristallo del tipo adatto per questo uso, evitando elementi di recupero malamente dissaldati (in quanto un eccesso di temperatura sui pin fa decadere le prestazioni del cristallo). In ogni caso, la frequenza generata non sarà più precisa o stabile di quanto lo siano i componenti utilizzati.

- le voci del **config** relative ai modi dell' oscillatore sfortunatamente non sono univoche per tutti i PIC, tanto che Microchip ha aggiunto nel file ***.inc*** vari alias. Questo non evita che essi possano essere insufficienti, rendendo necessario leggere il file per reperire le giuste label.

Come vediamo dalle esercitazioni, la gran parte delle applicazioni semplici e anche meno semplici lavora correttamente con l' oscillatore interno. I fogli dati riportano una precisione dell' ordine dell'1%, più che adeguata anche per applicazioni abbastanza critiche, come una trasmissione seriale asincrona, mentre la stabilità con la temperatura è adeguata nell' ambito delle variazioni comuni. Inoltre, la calibrazione dell' oscillatore interno permette di variare entro un certo ambito la frequenza generata e consente sia di ottenere valori leggermente diversi dalla fondamentale, sia di aggiustarne il valore in conseguenza di forti variazioni della temperatura.

Che molti esempi sul web riportino il cristallo e i condensatori come elemento tipico di un circuito con PIC dipende essenzialmente dal fatto che **il chip non dispone di clock interno**, **come i PIC obsoleti** (es. 1**6F84, 16F876/877**). Però, a volte, dipende solamente dalla mancata conoscenza delle possibili configurazioni alternative dell' oscillatore e, soprattutto, del fatto che i PIC meno obsoleti hanno tutti l'opzione di uno più frequenze disponibili attraverso l'oscillatore interno.
Anche a riguardo della precisione, va detto che un cristallo serve solamente nelle applicazioni time-sensitive, ovvero che richiedono una elevata precisione nelle temporizzazioni. Dal punto di vista più generale, a meno che sia necessaria una frequenza estremamente precisa/stabile o il chip non abbia oscillatore interno, usare un cristallo esterno a 4MHz o, peggio, un RC esterno quando esiste l'oscillatore interno è del tutto inutile e sacrifica pin altrimenti utilizzabili per l' I/O.

Gli oscillatori esterni vanno usati quando:

- il chip non dispone di oscillatore interno (chip obsoleti)
- è necessaria l' elevata stabilità del quarzo, che è maggiore di quella dell' oscillatore interno, soprattutto al variare della temperatura.
- sono necessarie frequenze precise di valori diversi da quelli ottenibili dall' oscillatore interno, ad esempio 32KHz per RTC o 2.476MHz per le trasmissioni seriali, ecc.

Una nota ulteriore per chi volesse fare misure con l' oscilloscopio sull' oscillatore a cristallo: il pin da testare è **`OSC2/CLKOUT`**, che è quello a impedenza minore e dove la sonda dell' oscilloscopio o frequenzimetro crea il minore disturbo. Applicando la sonda su **`OSC1`** si caricherà l' oscillatore con l' impedenza e la capacità della stessa.

Microchip ha emesso alcuni documenti sugli oscillatori:

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

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

e qui **trovate altre pagine sull' argomento**.

In ogni caso, utilizzando componenti adeguati ed un cablaggio decente, non abbiamo mai verificato problemi nell'implementazione di oscillatori a cristallo.
Se avete problemi in questo ambito, prima di accusare il chip o altro:

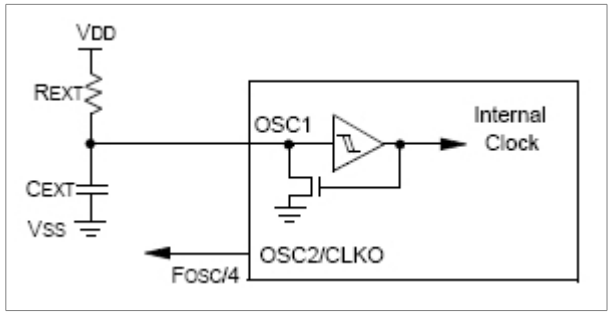
- **verificate i componenti e la correttezza dei collegamenti, sopratutto se state usando breadbord che possono avere contatti di inserzione danneggiati**
- **verificate la corretta impostazione dell' oscillatore scelto nel config**.

Se qui è tutto a posto, la causa è certo da cercarsi nel cristallo che è di tipo non adeguato (ad esempio, elementi che sono previsti per lavorare overtone, come i quarzi per trasmissione, o cristalli per low power inadeguati all' uso con microcontroller) oppure è danneggiato, come capita con elementi di recupero.
Raramente i condensatori pongono problemi; usando componenti nuovi, ceramici o ceramici mutistrato, NP0 o a bassa deriva, non ci sono mai problemi nell' avvio dell' oscillazione, anche se i valori non perfettamente centrati posso portare ad un qualche piccolo spostamento nella frequenza.

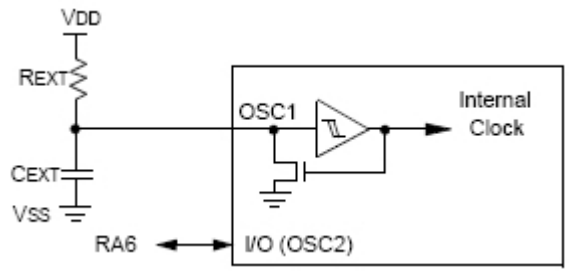
In relazione alla possibilità di aggiustamento della frequenza, si può considerare la possibilità di usare un trimmer capacitivo assieme a ***C1*** per aggiustarne la capacità in funzione delle caratteristiche del quarzo.

Clock esterno - RC

Un oscillatore esterno può essere realizzato anche usando una **rete RC**. Abbiamo due possibilità:



La prima consente di avere la rete RC esterna, collegata su **`OSC1`**, e su **`OSC2`** ottenere in uscita la **Fosc/4**.
Solamente i Baseline 16F5xx dispongono di questa opzione.



La seconda consente di avere la rete RC esterna, collegata su **OSC1**, e su **OSC2** avere disponibile la funzione di I/O digitale del pin.
Questa modalità è disponibile sia sui Baseline 12F che 16F.

Questo modo va utilizzato in un unico caso:

- quando **occorre una frequenza non disponibile internamente**,
- ma **di cui non importa nè precisione, nè stabilità** (*timing insensitive application* - applicazioni non sensibili alle temporizzazioni)
- e la richiesta progettuale è il minimo costo

Infatti, una certa precisione e stabilità sono ottenibili solamente utilizzando per R e C componenti di elevata qualità: se resistenze di precisione e stabili col variare della temperatura sono facilmente reperibili, lo stesso non si può dire dei condensatori. E se la ricerca nel progetto è relativa al basso costo, questo non si ottiene certo usando resistenze a strato metallico allo 0.1% e condensatori in mica.

Invece, anche se l' RC esterno è teoricamente impiegabile fino alla frequenza massima di lavoro del chip, questo modo è consigliabile e necessario quando si vogliano **clock con frequenze basse**, non possibili con i cristalli, ad esempio 1 o 2kHz.
In questo senso va fatta una considerazione fondamentale:

in effetti, i PIC sono dispositivi "**statici**", nel senso che **non hanno la necessità di un clock minimo per poter funzionare**; il clock parte da 0, ovvero, può essere interrotto, lasciando in sospeso l' esecuzione del programma. Questa è la chiave del funzionamento del modo **sleep**, dove lo stop al processore avviene proprio arrestando il clock.

E' ovvio che un basso clock provoca un tempo di ciclo istruzione proporzionalmente lungo: ad esempio, 4MHz di clock danno un tempo di esecuzione di 1us, ma 4kHz portano il tempo di esecuzione a 1ms per istruzione.
Per contro, **il consumo energetico si riduce drasticamente**: per PIC16F506, ad esempio, la corrente di alimentazione è valutata 1.4mA @ 5V con un clock di 20MHz e passa a soli 11uA @ 32KHz, con la possibilità di abbassare la tensione Vdd a 2V (foglio dati, param. D010).

Dal punto di vista realizzativo, le specifiche indicano che la ***Rext*** deve essere compresa tra 3 e 100Kohm, mentre Cext può essere anche omesso utilizzando le capacità parassite del circuiti, ma questa scelta è del tutto sconsigliata per evidenti ragioni. Quindi ***Cext*** potrà partire da 20pF in su.
La sezione "***Electrical Characteristic***" del foglio dati indica le specifiche dell' applicazione.

L' oscillatore RC è basato sulla carica del condensatore Cext attraverso la resistenza Rext. Raggiunta circa la tensione 0.75xVdd, un transistor interno scarica rapidamente il condensatore e la tensione cade a circa 0.25Vdd, ottenendo la tipica onda triangolare, che sarà poi squadrata dai circuiti successivi (vedi schemi della tabella precedente). E' possibile calcolare il periodo dell' oscillazione ottenibile da una combinazione RC con la formula:

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

dove è la tensione di commutazione dello Schmitt trigger posto all' ingresso del pin OSC1 e del valore di circa 0.9Vdd (param. D043).
Ad esempio, per ***Vdd=5V, R=10k e C=22pF*** si ha:

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

Per cui la frequenza sarà:

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

e: $$Fosc/4 = 494kHz$$

Se consideriamo la sola alimentazione a 5V, possiamo usare una formulazione abbreviata:

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

Ovviamente il valore reale ottenuto dipende dalla tolleranza dei componenti R e C usati e dalle caratteristiche e tolleranze costruttive del chip. In più, va considerata la loro variazione con la temperatura.

L' opzione **RC esterno** richiede:

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

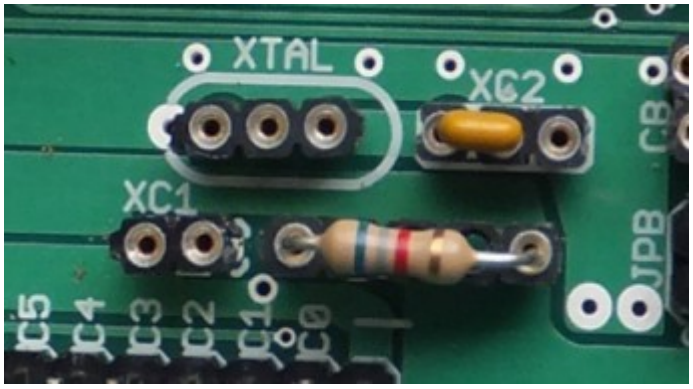
se si vuole l'I/O digitale disponibile. Per i chip in cui è possibile l' uscita ***Fosc/4***:

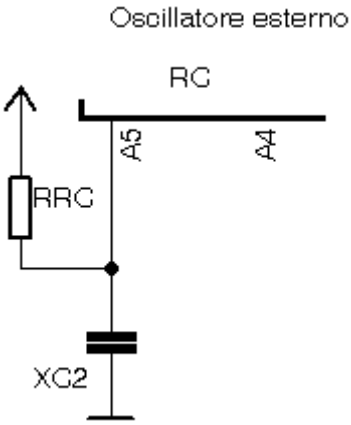
```
 __config  _FOSC_ExtRC_CLKOUT  ; EXTRC with CLKOUT on OSC2
oppure
 __config  _ExtRC_OSC_CLKOUT   ; EXTRC with CLKOUT on OSC2
```

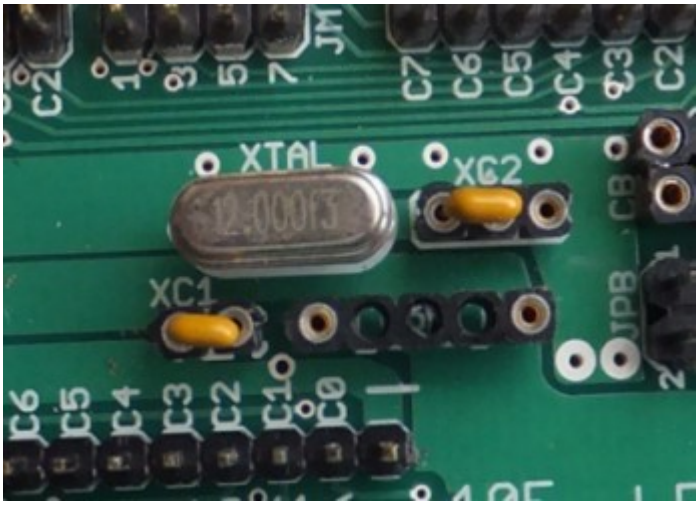
Anche qui, le voci del **config** relative a questi modi dell' oscillatore sfortunatamente non sono univoche per tutti i PIC, tanto che Microchip ha aggiunto nel file ***.inc*** vari alias. Questo non evita che essi possano essere insufficienti, rendendo necessario leggere il file per reperire le giuste label.

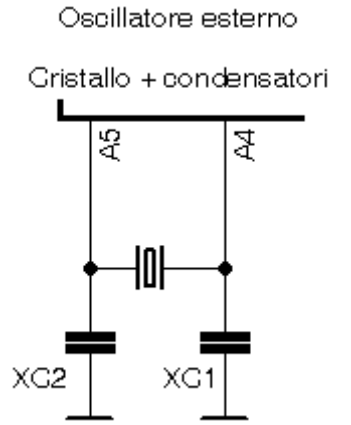
In pratica

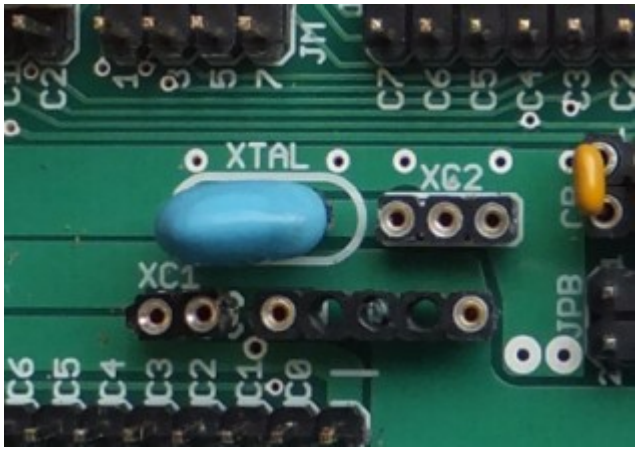
La scheda di sviluppo dispone della possibilità di sperimentare tutte le possibili variazioni dell' oscillatore esterno in quanto esiste un' area dedicata ai componenti:

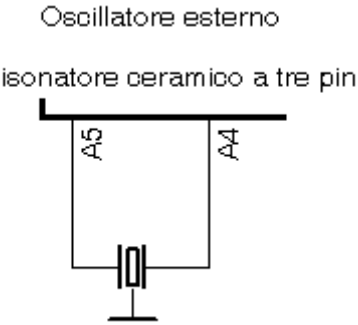












Osserviamo che il risonatore ceramico a tre pin integra i condensatori, per cui non occorre aggiungerli.
la scelta dell' oscillatore ceramico al posto del quarzo è dettata dal minore costo di questo componente e dalla sua funzionalità, seppure a fronte di minore precisione/stabilità.

Volendo sperimentare questi oscillatori, sarà necessario:

- inserire i componenti richiesti da ogni schema
- considerare che uno o due pin non possono essere utilizzati per altri scopi
- e che nel sorgente va modificato il config in funzione del modo scelto

Se operiamo su supporti diversi dalla **LPCuB** basterà realizzare i circuiti indicati, avendo cura di verificare che le connessioni siano sicure, sopratutto su breadboard.

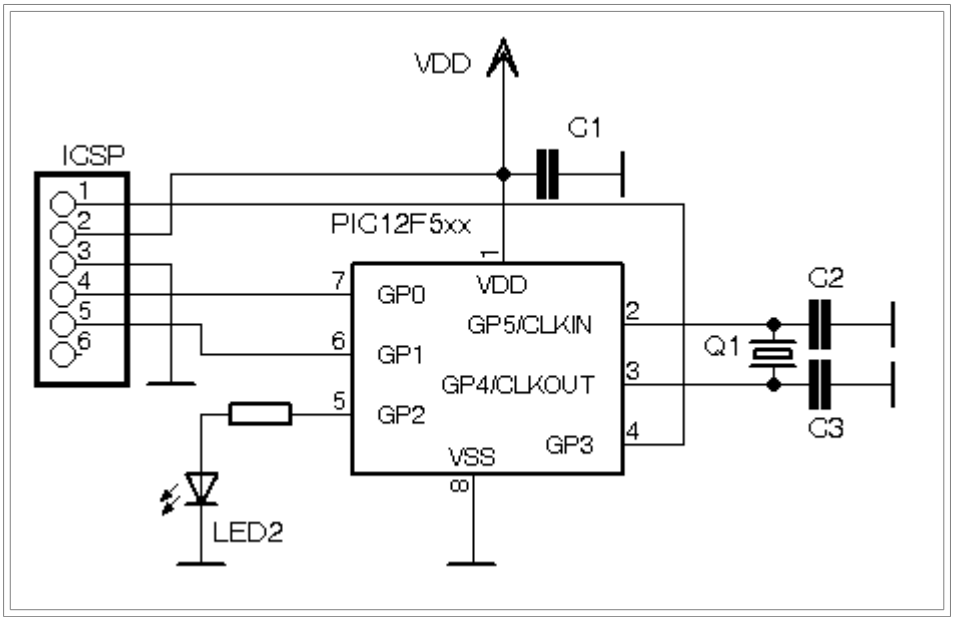
Un esempio applicativo - modalità LP/XT/HS

Possiamo vedere in pratica l' effetto del clock sull' esecuzione del programma facendo lampeggiare un LED. Per la modalità con oscillatore esterno, occorre collegare i componenti come indicato sopra, occupando i pin **OSC1/OSC2**. Questo implica che i pin non potranno essere usati come I/O. Questo vuol dire che per i chip a 8 pin (12F5xx) si può arrivare alla perdita di due I/O digitali su 6, mentre per i PIC a più pin la perdita di 2 I/O è meno sensibile. In particolare:

PIC	CLKIN	CLKOUT	Pin non utilizzabili come 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

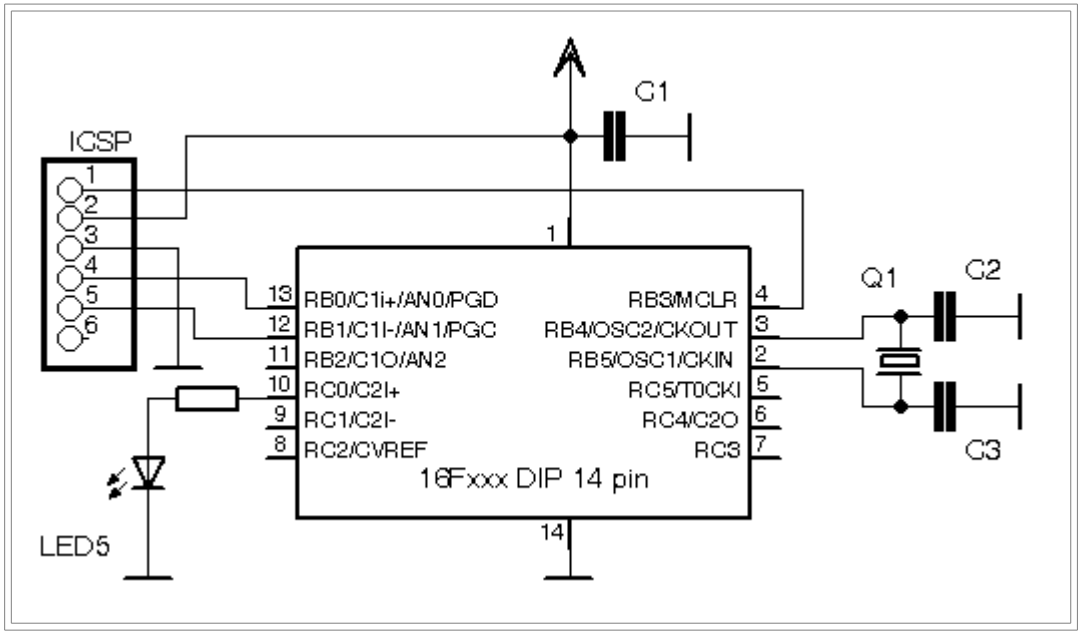
- indica un modo non disponibile per quel chip. Nei chip 10F2xx è possibile solo il modo RC interno, con l'eventuale uscita ***Fosc/4*** vista sopra.

Si possono sperimentare oscillatori esterni a cristallo con i seguenti schemi:



Si potranno utilizzare quarzi fino a 4MHz circa (o 8MHz, a seconda del modello), anche se il chip funziona senza problemi con frequenze maggiori, ma si tratterebbe di "overclock", condizione non

certificate dal costruttore.
Per i 16F5xx:

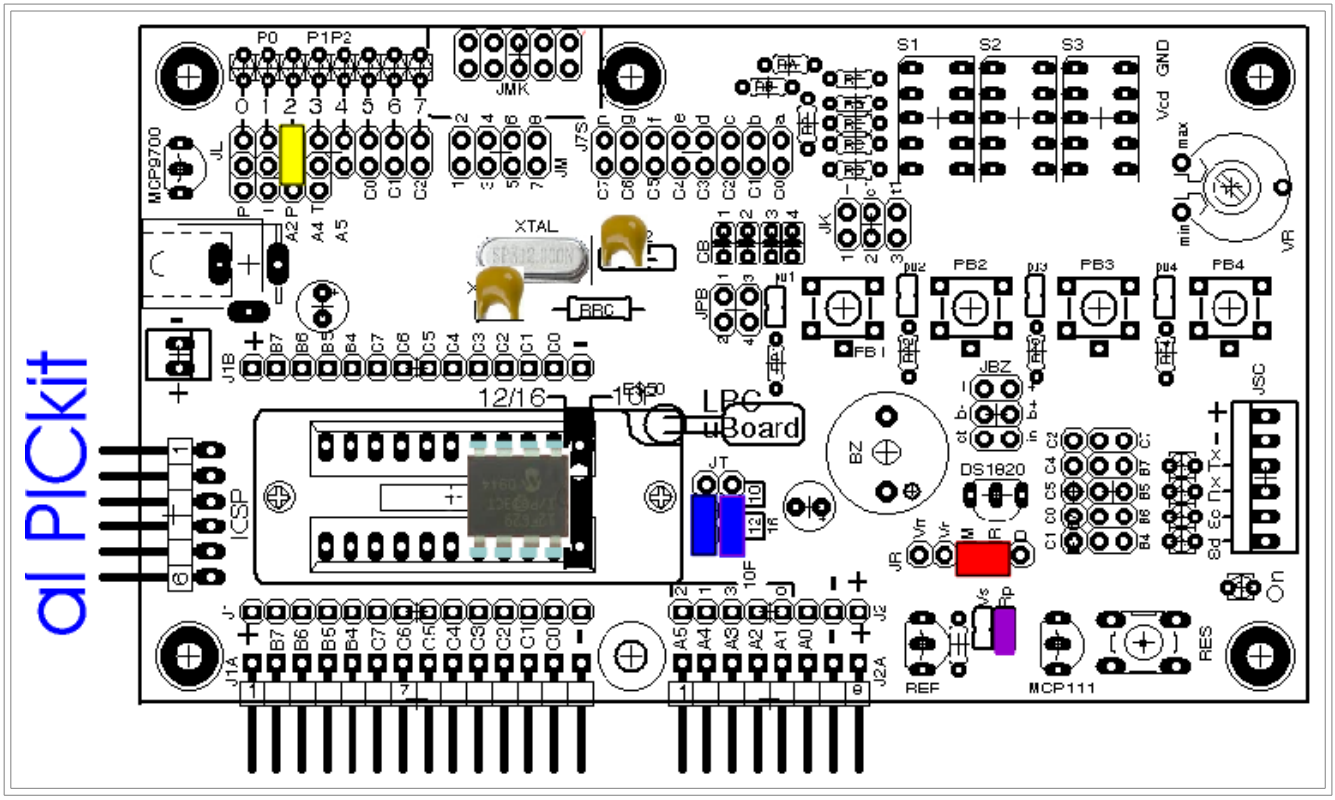


Si potranno utilizzare quarzi fino a 20MHz. Vanno bene cristalli in contenitore HC49 o comunque adatti per l' impiego con oscillatori del genere. I condensatori potranno essere ceramici di qualità da 15-27pF. Potremo usare anche risonatori ceramici a due pin o quelli a tre pin; in quest' ultimo caso non occorrono i condensatori esterni.

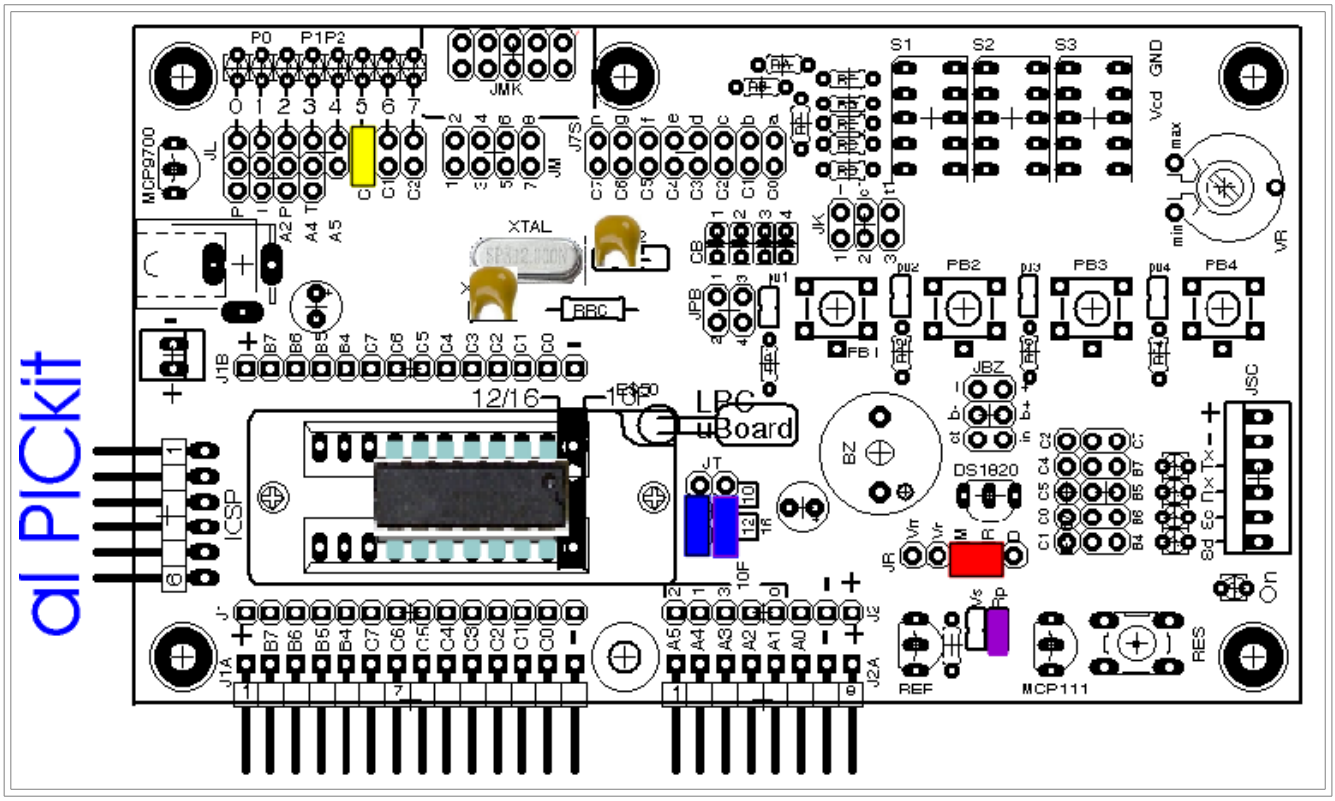
Cristallo: 1-4(20)MHz modo **XT** o **HS**	Cristallo: 32-200kHz modo **LP**	Risonatore ceramico 2 pin modo **XT** o **HS**	Risonatore ceramico 3 pin modo **XT** o **HS**

Se usate componenti di recupero, assicuratevi che si tratti di oggetti adeguati all' uso. In particolare, non sono adatti cristalli per oscillatori serie, cristalli per oscillatori in armonica, filtri ceramici di MF e simili.
La scelta del **modo LP** è obbligatoria per i **quarzi low power da 32kHz** usati in orologeria; per modelli di dimensioni molto miniaturizzate può essere necessario aggiungere la ***Rs*** come da schema precedente. Per altri quarzi, solitamente per frequenze al di sotto dei 4MHz, va bene il modo **XT**, ma se il cristallo è "duro" va impostato **HS**, anche per frequenze minori, sopratutto con oscillatori ceramici. **HS** va scelto per cristalli al di sopra dei 4MHz.

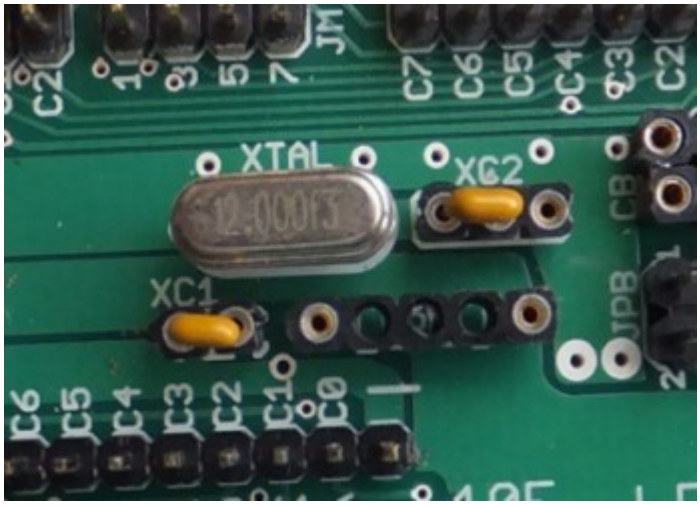
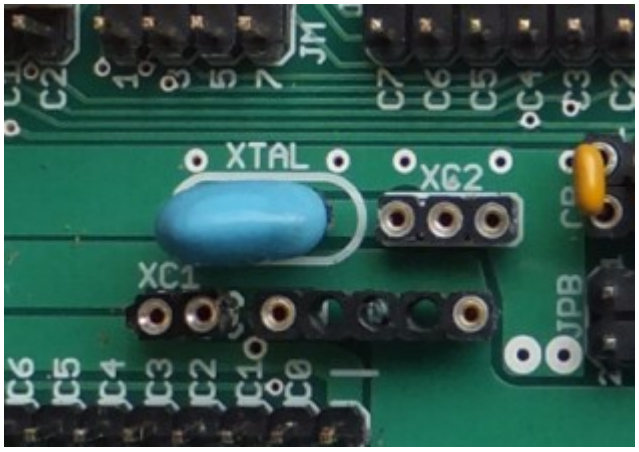
Per quanto riguarda la disposizione sulla board di sviluppo **LPCuB**, per i DIP-8 pin:



e per i DIP-14 pin:



Nel reale, un ingrandimento dell' area dei componenti dell' oscillatore:

	I componenti dell' oscillatore sono inseriti sugli appositi zoccoli con contatti a tulipano. Si deve aver cura, se si utilizzano componenti di recupero, di assicurarsi che i terminali siano perfettamente puliti da residui di saldatura, onde non danneggiare i contatti degli zoccoli. Lo zoccolo di ***XC2*** consente di installare condensatori passo 2.54" o 5.08". Osservate nella foto il posizionamento del condensatore col passo più stretto.
	Nel caso in cui si usi un oscillatore ceramico **a tre pin**, questo trova posto sullo zoccolo ***XTAL***, mentre i condensatori non sono necessari.

Il programma

La differenziazione tra le modalità di oscillatore riguarda solamente il config. Ad esempio, per la modalità **XT**:

```
; PER 12F519
; Oscillatore interno, no WDT, no CP, GP3 = MCLR
;
 __config _XT_OSC & _WDTE_OFF & _CP_OFF & _CPDF_OFF & _MCLR_ON

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

; PER 16F505/526
```

```
 #ifdef __16F526
; Oscillatore interno, 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
```

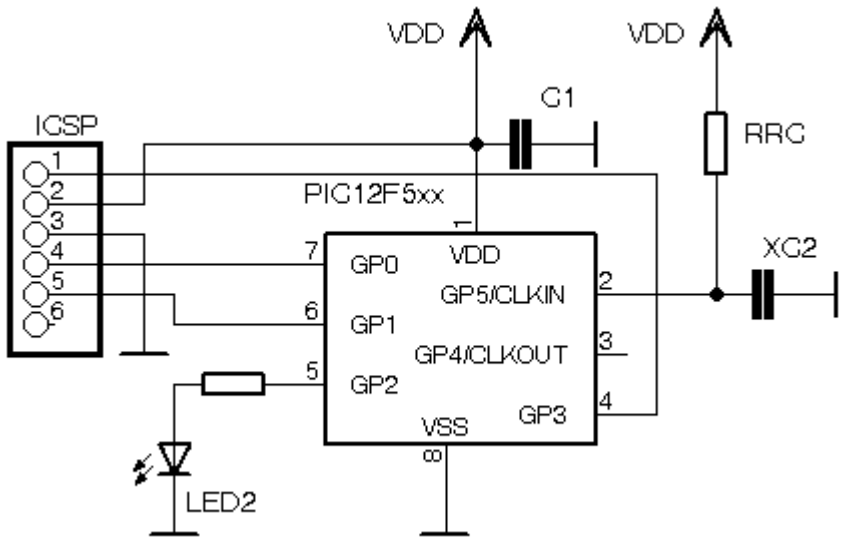
Per la modalità **LP** si utilizzerà la label **_LP_OSC**.
Per la modalità **HS** (possibile solo sui 16F5xx) si userà **_HS_OSC**.

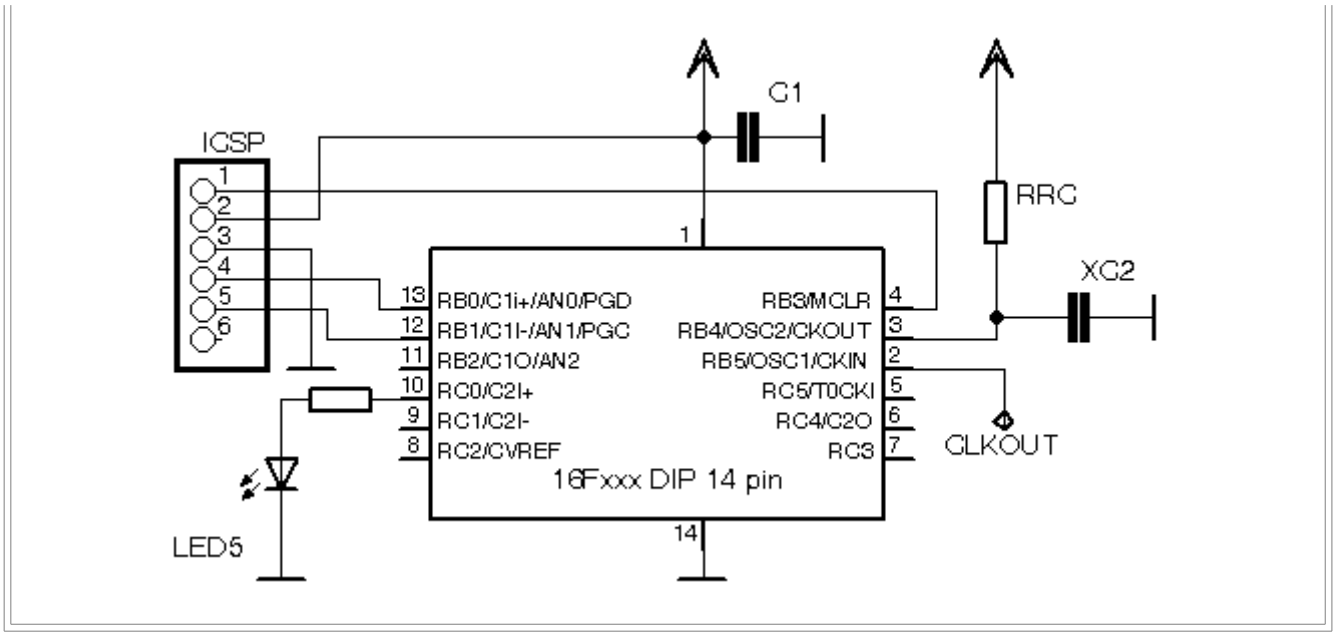
Non sarà possibile utilizzare i pin **OSC1/OSC2**.

Inoltre, occorrerà adeguare eventuali routine di tempo alla nuova frequenza del clock.

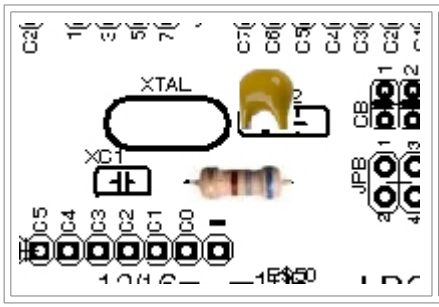
Modalità ExtRC

Possiamo sperimentare anche questo oscillatore.





Nel caso dei PIC a 14pin possiamo abilitare la modalità ExtRC-Clkout e verificare sul pin **`RB5/OSC2`** l' uscita della frequenza generata/4.



La disposizione dei componenti è identica alla precedente, con la differenza che dovranno essere inseriti la ***Rext*** (**RRC**) e il ***Cext*** (**XC2**)

Lo zoccolo di ***XC2*** consente di installare condensatori passo 2.54" o 5.08". Osservate nella disegno il posizionamento del condensatore col passo più stretto.

Usando una resistenza da 33k, possibilmente di buona qualità (strato metallico, 1 o 2%) ed un condensatore da 33pF (multistrato ceramico o film plastico) dovremmo ottenere una frequenza di oscillazione di:

$$T = 2.3 \times 33 \times 33 = 2504 \text{ ns} \rightarrow 399.5\text{Hz}$$

che corrisponde ad un ciclo di istruzione di poco inferiore a 10ms.

Ovviamente potete utilizzare qualsiasi altra combinazione, adeguando il software alla scelta fatta.

Il programma

Anche qui le variazioni consistono esclusivamente nella configurazione in cui va posto il modo voluto per l' oscillatore, usando la label **`_ExtRC_OSC`** per i 12F5xx e i 16F5xx.

Inoltre, per i soli 16F5xx possiamo usare anche **`_ExtRC_OSC_RB4EN`** ottenendo l' uscita della ***Fosc/4*** sul pin **`OSC2`**.

Ovviamente è necessario adeguare le routines di tempo.

Conclusioni

Riassumiamo alcune definizioni:

- **Fosc** : ***Frequency of the oscillator*** - la frequenza dell' oscillatore, di qualunque tipo sia
- **Fosc/4** : la frequenza interna relativa ad un ciclo-istruzione, pari a 1/4 della frequenza dell' oscillatore
- **tcyc** : il tempo di esecuzione (ciclo) di una istruzione

Come possiamo facilmente capire, i diversi tipi di clock non influenzano le istruzioni se non per il diverso tempo di esecuzione. Ricordiamo che questo tempo è dato da:

$$t = 1/Fosc/4$$

Per avere una idea della variazione:

Fosc [MHz]	Fosc/4 [MHz]	tcyc [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

- Una variazione nella modalità del clock comporta l' obbligo (oltre alla necessità eventuale di aggiungere componenti esterni) della corretta configurazione iniziale. Ricordiamo che la configurazione non fa parte del programma vero e proprio, ma è indispensabile nel sorgente per fissare le caratteristiche dell' oscillatore.
- La variazione della frequenza di clock interessa solamente le parti del programma che agiscono attraverso un calcolo del tempo; aumentando la ***Fosc***, diminuisce il tempo di esecuzione e quindi occorre adeguare le routines di tempo o le azioni che utilizzano i cicli istruzione come elementi di tempo.

- l' aumento della frequenza dell' oscillatore, riducendo il tempo di esecuzione delle istruzione, consente maggiori performances del processore.

- il consumo di energia del microcontroller è proporzionale alla frequenza. Applicazioni a basso consumo dovranno utilizzare basse frequenze di lavoro.

- la frequenza di lavoro del microcontroller è proporzionale alla tensione di alimentazione: al diminuire della tensione si riduce anche la possibilità di operare con frequenze elevate.

Per quanto riguarda i vari modi di oscillatore, dove possibili, la scelta va fatta secondo queste considerazioni:

- **oscillatore interno** : praticamente adatto a tutte le applicazioni dove il tempo di esecuzione ammette una tolleranza attorno all'1%. Presenta il doppio vantaggio di non necessitare di componenti esterni e di lasciare due pin liberi come I/O.

- **oscillatore esterno a cristallo** : necessario solamente per i processori che non dispongono di oscillatore interno, in quelle applicazioni dove tempo di esecuzione richiede maggiore precisione e stabilità nel tempo/temperatura oppure è necessaria una frequenza non disponibile attraverso l' oscillatore interno. Occorrono componenti esterni e si occupano i due pin dell' oscillatore.

- **oscillatore esterno** : da utilizzare se nel circuito è già presente una sorgente di clock o se necessita una stabilità e precisione fornibile da oscillatori integrati (TCXO e simili) oppure è richiesta una frequenza non disponibile attraverso l' oscillatore interno. Occupa un pin per l' ingresso del clock esterno.

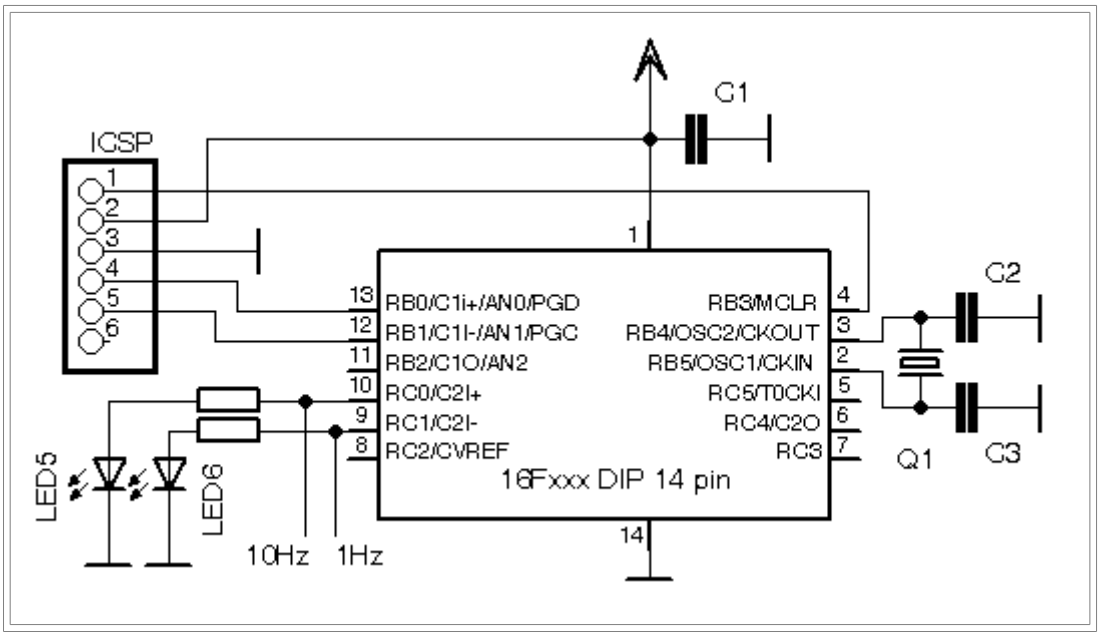
- **oscillatore RC** : può essere utilizzato quando precisione e stabilità del clock non sono elementi determinanti, ma serve una frequenza che non è disponibile attraverso l' oscillatore interno. Occupa un pin per l' ingresso del clock esterno.

In ogni caso, la precisione delle temporizzazioni nel programma dipende dalla precisione della Fosc.

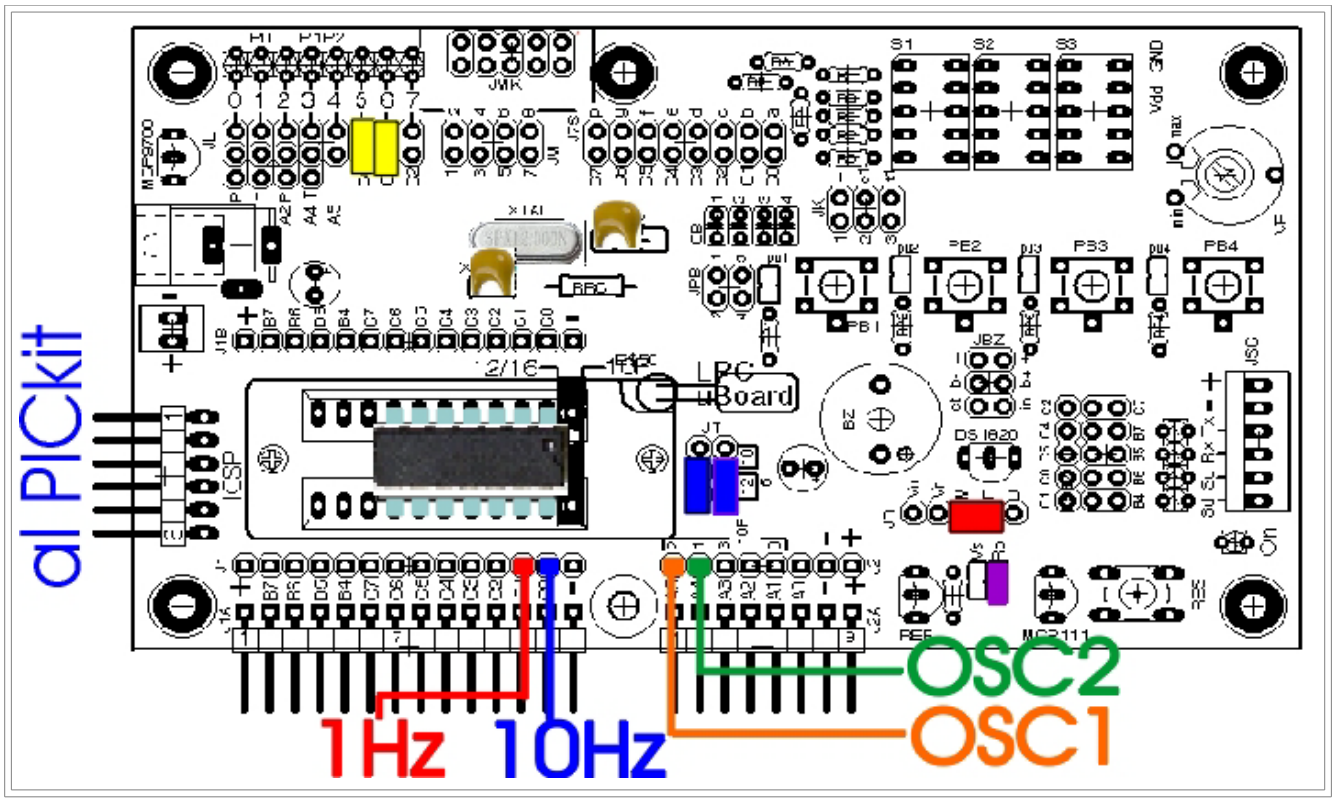
Una applicazione: generare 1Hz partendo da 14.318MHz

La frequenza di 1Hz è utile nelle applicazioni RTC. Solitamente si ricorre ad un cristallo nell' oscillatore per avere sia una elevata precisione che una buona stabilità nel tempo e con le variazioni di temperatura.
Usiamo, quindi, un oscillatore esterno a quarzo.
Ci sono varie possibilità di scelta per il valore della frequenza dell' oscillatore: una prima che utilizza i quarzi per orologeria a 32768Hz e un' altra con valori di frequenza maggiori. Vediamo dapprima la seconda possibilità, forse meno comune della prima, ma che utilizza comunque un quarzo di facilissima reperibilità e basso costo.
Tra i tanti valori disponibili, possiamo usare la frequenza di **14.318000MHz**. Perché questo valore apparentemente strano? Semplicemente perché si tratta di un cristallo di uso quanto mai comune, facilmente reperibile ed economico, essendo presente nelle base tempi delle mainboard PC. Si tratta normalmente di elementi in contenitore HC-49 con terminali per montaggio a foro o SMD. A noi servirà la prima opzione per una installazione diretta sugli zoccoli della **LPCuB**. Oltre al cristallo, servono due condensatori ceramici da 15-30pF; questi non sono critici, ma occorre siano di buona qualità.

L'uscita della frequenza generata di 1Hz è inviata ad un pin del **PORTC**. Dato che il software lo permette facilmente si è ottenuta una uscita anche a 10Hz. Due LED collegati alle uscite indicano con il loro lampeggio il funzionamento del circuito. Lo schema elettrico è analogo a quello già visto:



e sulla **LPCuB**:



I jumper "gialli" hanno lo scopo di aggiungere LED sulle uscite per una visualizzazione dei segnali anche senza oscilloscopio.

Il programma

Il sorgente è ***9A_10-1Hz.asm***.
La frequenza di 14.318 MHz origina un tempo di ciclo per istruzione di 0.28us circa Con un loop di 178975 cicli si ottiene un ritardo di 50ms. Ad ogni temporizzazione si inverte lo stato del pin di uscita dei 10Hz, mentre ogni 10 si inverte lo stato del pin dell' uscita 1Hz, ottenendo così onde quadre con la frequenza richiesta.

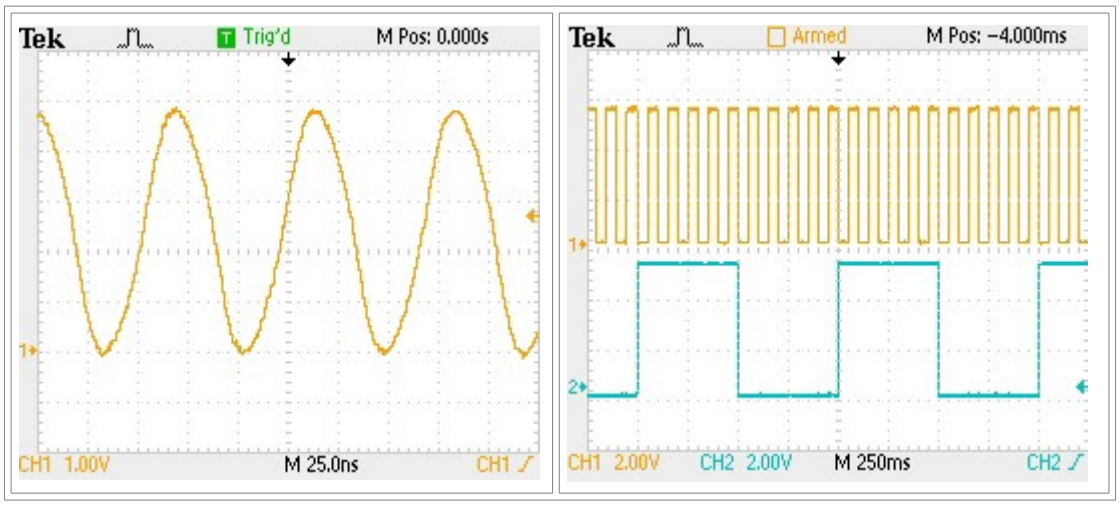
La solita fase iniziale di adeguamento al processore scelto, occorre nel config selezionare la modalità **HS** (High Speed) per il corretto funzionamento dell' oscillatore. In particolare, viene definito un parametro **`procanalog`** che permette poi la selezione delle giuste azioni per escludere le periferiche analogiche in quei chip che ne sono forniti.

L' assegnazione della memoria RAM è effettuata con la direttiva **`UDATA`**.

Segue semplicemente una catena di 10 iterazioni del ritardo voluto, che è sotto forma di una subroutine. Al termine di ogni iterazione viene effettuata l' inversione del valore del pin con un OR esclusivo, in modo da ottenere in uscita un' onda quadra simmetrica, con semi periodo 50ms. Alla decima iterazione viene commutata anche l' uscita a 1Hz.
Da notare la prima temporizzazione che è 2us più breve delle altre per compensare il **`goto`** che chiude il loop.

Ecco come si presentano la forma d'onda dell' oscillatore quarzato al pin OSC2 e le frequenze in

uscita di 1 e 10Hz:

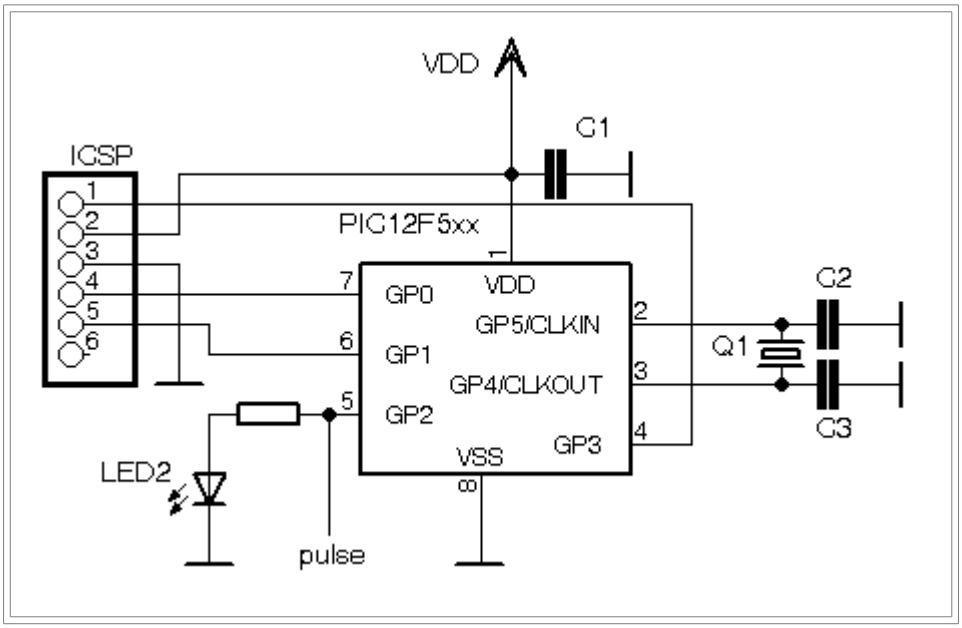


Non è necessaria la calibrazione del' oscillatore interno, che non è usato (ma che viene comunque espressa nel sorgente come commento per ricordarne la necessità in altre occasioni). Altrettanto per la disabilitazione di T0CKI da RB5, che è definito nel config come pin dell' oscillatore esterno.

Generare 1Hz partendo da 32768Hz

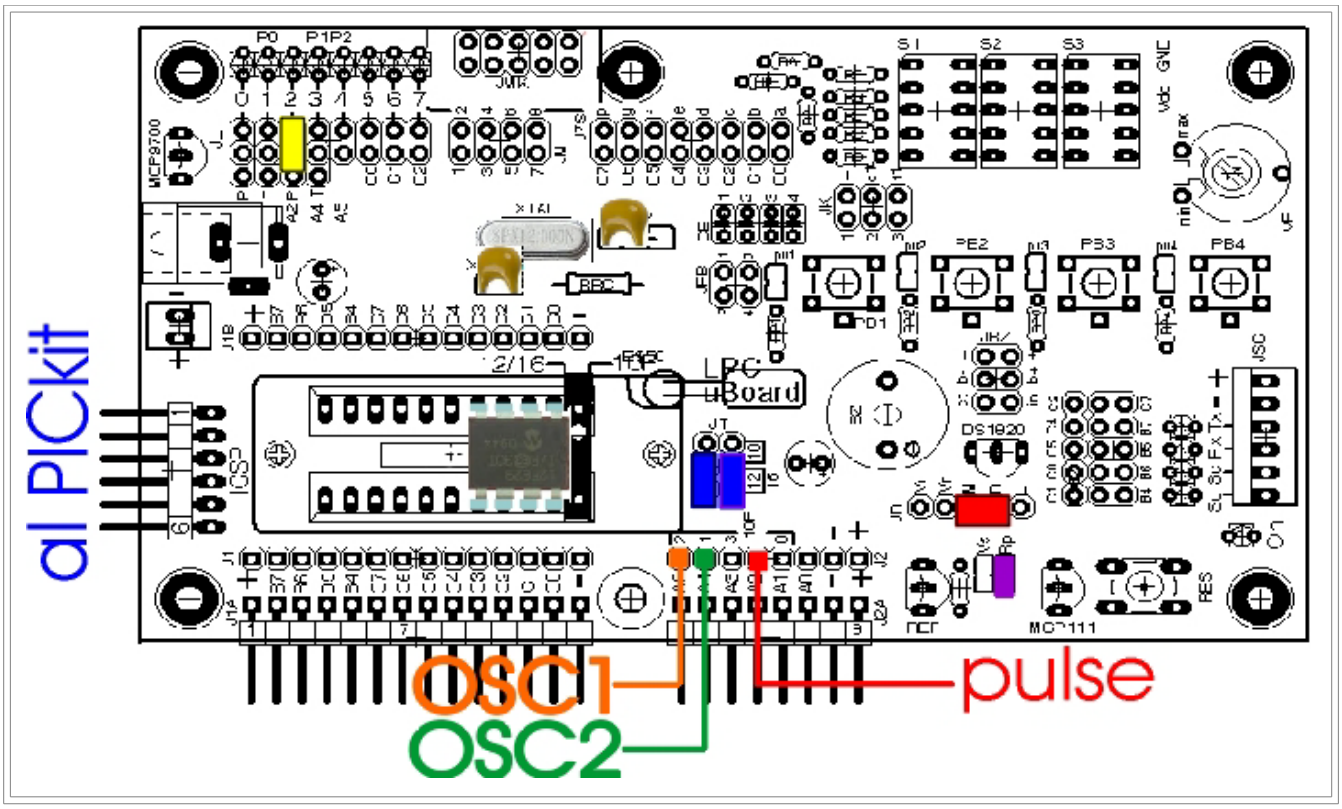
I Baseline a 8 pin dichiarano una frequenza massima di 4(8)MHz, per cui il circuito precedente non è applicabile. Occorre un cristallo di frequenza minore. Lo stesso programma sarebbe adeguabile per un cristallo da 3.5795MHz, che è 1/4 di 14.318MHz; però questo valore non è così comune, mentre lo è il 32768Hz. Questi quarzi sono reperibili dal recupero di vecchie schede madri (dove svolgono proprio funzione di supporto dell' RTC) oppure da uno dei tanti orologi elettrici Made in China., anche se un componente nuovo, di qualità e con un foglio dati dove reperire il valore nominale dei condensatori, non costituisce certo una spesa elevata.

Lo schema applicativo è analogo al precedente.



Facciamo uscire il segnale sul GP2, dove un LED permette di visualizzarlo.

Per l' hardware sulla **LPCuB**:



Il jumper “giallo” collega il LED 2 all’uscita GP2 e lampeggerà alla frequenza prodotta.
Le forme d’onda dell’ oscillatore si possono prelevare dai pin indicati, usando sonde x10.
Quarzo e condensatori occupano i rispettivi zoccoli.

Il programma

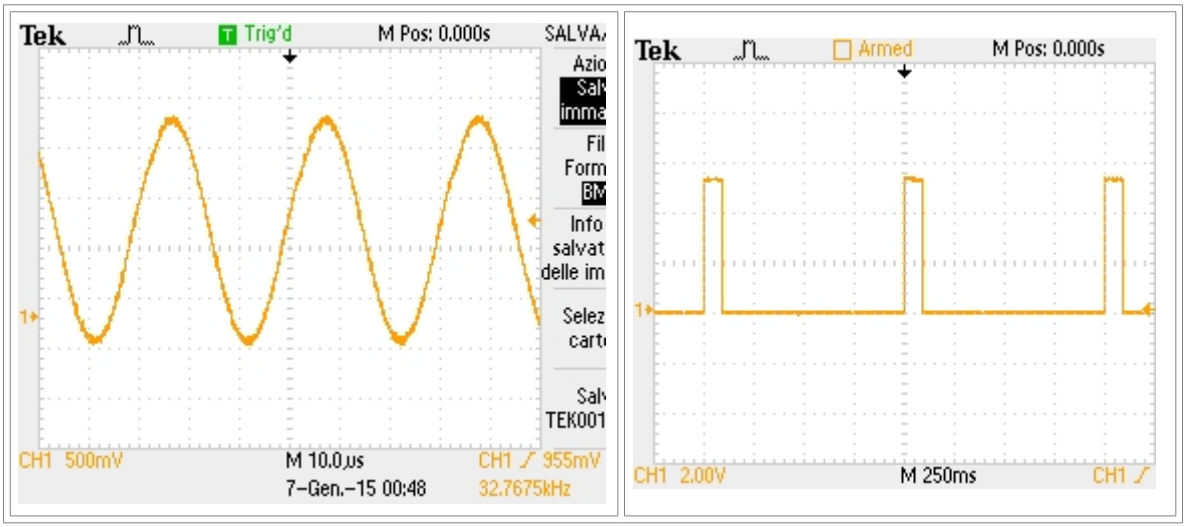
Il sorgente è ***9A_1Hz.asm***.

Occorre adeguare il config iniziale: va impostata la modalità di oscillatore **LP** (*Low Power*) perchè si tratta di un quarzo a bassa potenza; un errore nell' impostazione di una potenza maggiore può non impedire l' oscillazione, ma questa avviene in modo disordinato e su una frequenza molto superiore a quella nominale, rendendo il funzionamento errato e casuale. I condensatori potranno essere, sempre ceramici, da 7 a 18pF a seconda del cristallo usato. Inserendo condensatori di capacità maggiore, ad esempio 47-100pF l' oscillazione è possibile e, in generale, più stabile, ma il tempo di avvio può essere considerevole (anche un paio di secondi!).

Il programma si basa sull' uso del **Timer0** in free-running: il timer non viene caricato e conta liberamente in continuazione tra 0 e 255.
Il valore della frequenza di oscillazione è la chiave della semplicità del programma: 32768 è un multiplo di 2 (2^15); la logica del processore è binaria e l' impiego di numeri potenze di 2 semplifica grandemente le operazioni.
Con ***Fosc = 32768Hz*** si ha ***Fosc/4 = 8192Hz***. Questo vuol dire che si eseguono 8192 cicli istruzione per il tempo di 1 secondo.
Con un prescaler di 32, il ciclo di overflow del conteggio del **Timer0** sarà in 1 secondo esatto (256 x 32 = 8192). Al termine di ogni overflow, verificato in polling, viene inviato un impulso alto da 100ms sul pin in uscita. Nel frattempo il timer continua a contare e il programma ricicla in attesa di un nuovo overflow.

Il polling è possibile in quanto un valore nel registro di conteggio **TMR0** permane per un numero di cicli istruzione pari al pre divisore, dando modo al programma di non perdere l' evento voluto.
Il tempo dell' impulso è valutato verificando quando **TMR0** , nel suo conteggio continuo, raggiunge il valore di 25, il che corrisponde a 97.6525ms. Un LED collegato sull' uscita lampeggia brevemente una volta al secondo, seguendo l'andamento dell'impulso.
Da notare che non è richiesta alcuna locazione di RAM.

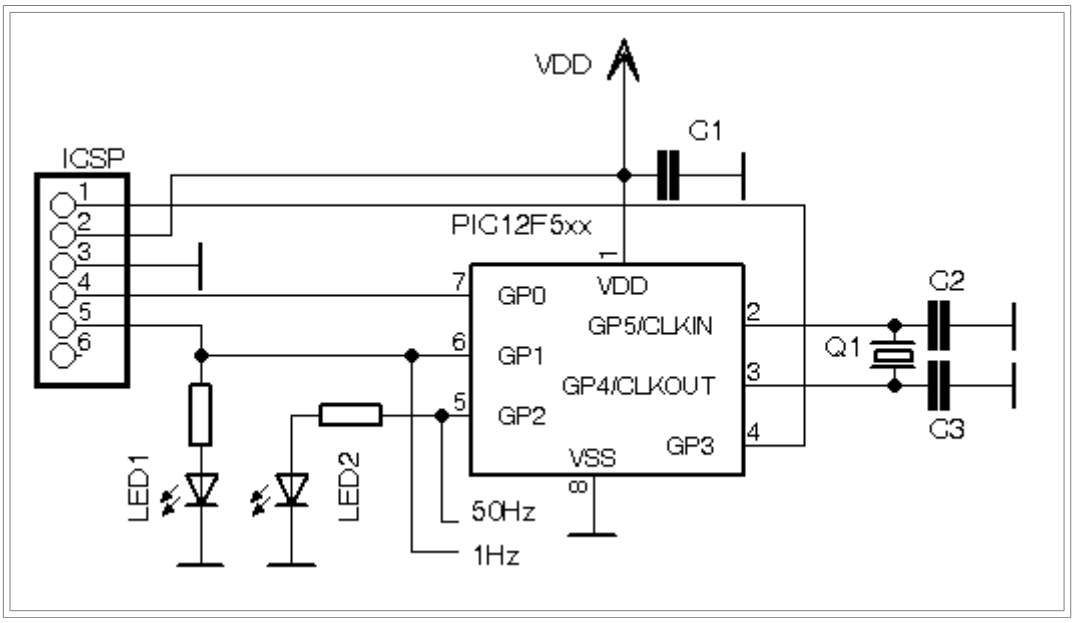
Ecco come si presentano la forma d'onda dell' oscillatore quarzato al pin OSC2 e l' uscita:



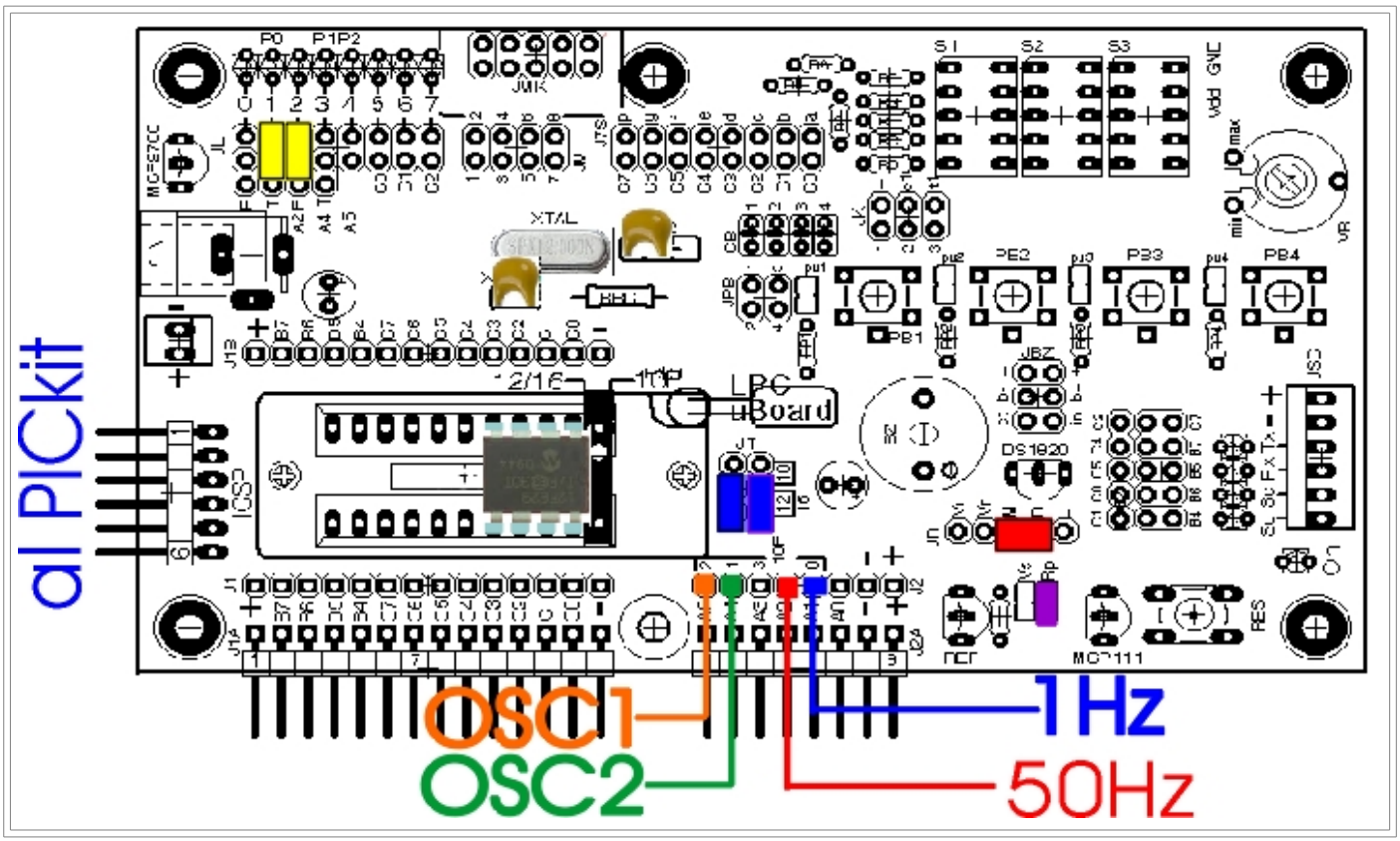
La frequenza misurata dall'oscilloscopio è 32767.5Hz, praticamente il valore nominale. La forma d'onda è perfetta, indice che il cristallo lavora in un condizione di carico ideale; è ottenuta con un quarzo Abracom e due condensatori da 10pF.
Nell'immagine a destra si vede l' impulso da 100ms con cadenza 1s.

Generare 1Hz e 50Hz partendo da 3.2768MHz

Usando un quarzo da 3.2768MHz (multiplo di 10 del precedente) siamo ancora ampiamente in quanto permesso ai 12F5xx ed abbiamo un ciclo istruzione 10 volte più breve.
Lo schema è analogo al precedente:



a pure la disposizione dei componenti sulla **LPCuB**:



I jumper "gialli" hanno lo scopo di aggiungere i LED sulle uscite per una visualizzazione dei segnali anche senza oscilloscopio.

Nota: il **LED1** si trova collegato al pin **GP1**, che è anche uno dei segnali della programmazione ICSP.
Sulla LPCuB i LED sono a bassissima corrente (max. 2mA) e non influiscono sulla programmazione, dato che costituiscono un carico trascurabile. Si noterà il LED1 lampeggiare rapidamente durante questa la fase, per acquistare poi la sua funzione una volta avviato il programma.
Se usate altro hardware, assicuratevi che la corrente nel LED sia minima oppure scollegatelo durante la programmazione.

Il programma

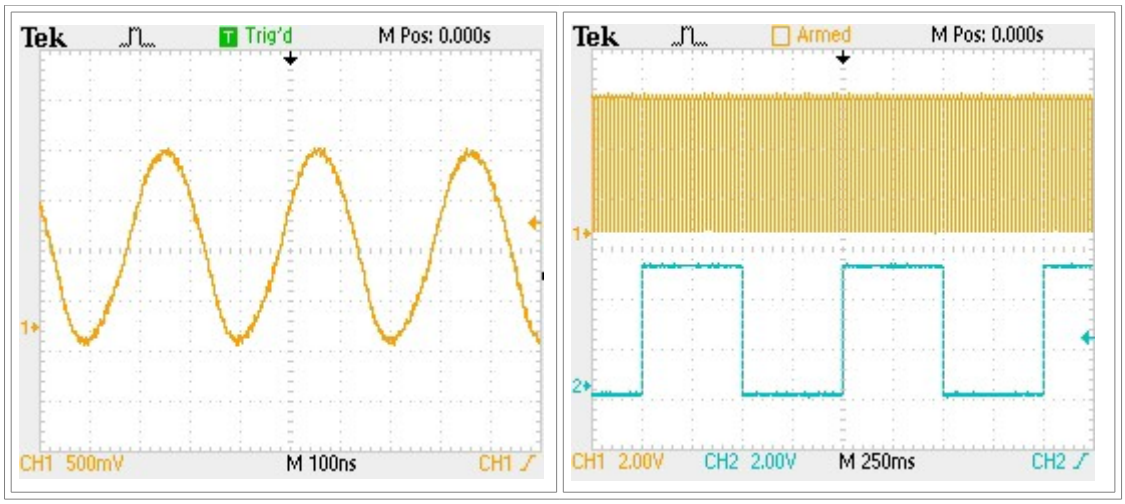
Sfruttiamo la maggiore frequenza del clock (minore durata del ciclo di istruzione) con una procedura simile alla precedente per generare un'onda quadra a 50Hz e una a 1Hz: il **Timer0** lavora free running, con un pre divisore 1:256.

1. abbiamo ***Fosc = 3276800Hz*** e quindi ***Fosc/4 = 819200Hz***. Ogni 32 conteggi del timer con un pre divisore a 256 richiedono 32 x 256 cicli di clock, il che equivale a 10.00ms.
2. Procediamo, allora, verificando periodicamente in polling il valore del registro di conteggio **TMR0** e ogni 10ms (32 conteggi) commutiamo l' uscita a 50Hz. All' ottavo ciclo il contatore si è azzerato e riprende da capo il conteggio automaticamente.
3. Ogni 50 di questi eventi commutiamo l' uscita 1Hz per ottenere l' onda quadra corrispondente. Possiamo effettuare il test per l' uscita 1Hz senza problemi in quanto un conteggio permane in **TMR0** per un numero di cicli istruzione pari al pre divisore (qui, ben 256 istruzioni), dando modo al programma di non perdere l' evento voluto.

Anche in questo caso il programma è straordinariamente semplice per la possibilità di sfruttare una frequenza di clock tale da non richiedere operazioni su contatori.

Il listato ***9A_50-1Hz.asm*** è previsto per 12F508/509/519.

Ecco come si presentano la forma d'onda dell' oscillatore quarzato al pin OSC2 e le due uscite:



La frequenza misurata dall'oscilloscopio è 3.2768MHz, il valore nominale. La forma d'onda è praticamente perfetta, indice dell' oscillatore che lavora in un condizione di carico ideale, ottenuta con un quarzo Foxconn e due condensatori da 22pF.
Si posso usare quarzi con frequenza pari a una potenza di 2, come 4.096MHz, 2.048MHz, 1.6384MHz e simili aggiornando i cicli del programma.
Usando altre frequenze (ad esempio i 4MHz dell' oscillatore interno) può diventare necessario un firmware più complicato per ottenere i valori voluti, solitamente con minore precisione.

Conclusioni

Dovrebbe essere chiaro come, utilizzando un valore opportuno dell' oscillatore, possiamo semplificare grandemente il software per generare valori precisi di tempo, utili per le routines interne o per produrre frequenze in uscita, valori spesso non ottenibili con la voluta precisione usando cristalli non adeguati o l' oscillatore interno.
L'uso del timer consente di effettuare altre operazioni durante il conteggio, sfruttando l'effetto del pre divisore.

9A_10-1Hz.asm

```
;*******************************************************************
;-------------------------------------------------------------------
;
; Titolo        : Corso Assembly & C - Esercitazione 8A
;                 Generatore 10Hz e 1Hz
;                 Oscillatore esterno 14.318MHz
; PIC           : 16F505/506/526
; Supporto      : MPASM
; Versione      : V.16F5xx-1.0
; Data          : 01-05-2013
; Ref. hardware :
; Autore        : afg
;
;-------------------------------------------------------------------
;###################################################################
; Selezione del processore

 #ifdef __16F505
    LIST p=16F505 ; Definizione del processore
    #include <p16F505.inc>
 #endif
 #ifdef __16F506
    LIST p=16F506 ; Definizione del processore
 #include <p16F506.inc>
 #define procanalog
 #endif
 #ifdef __16F526
    LIST p=16F526 ; Definizione del processore
 #include <p16F526.inc>
 #define procanalog
 #endif

   radix dec

;*******************************************************************
;                 DEFINIZIONE DI IMPIEGO DEI PORT

millis equ 0x01 ; RC0 uscita 10Hz
second equ 0x02 ; RC1 uscita 1Hz
;
;###################################################################
;                     CONFIGURAZIONE
;
; Oscillatore esterno 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
```

```
;#################################################################
;                        MEMORIA RAM
;
; general purpose RAM
      UDATA
d1 res 1 ; contatori per ritardo
d2 res 1


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

; calibrazione oscillatore interno
; movwf OSCCAL

;#################################################################
;                       MAIN PROGRAM
;
Start
 #ifdef procanalog
; disabilita gi ingressi analogici
    clrf ADCON0

; disabilita comparatori per liberare la funzione digitale
    bcf  CM1CON0, C1ON
    bcf  CM2CON0, C2ON
 #endif

; disabilita T0CKI da PORTB5
         ; b'11011111'
           ; 1------- GPWU disabilitato
           ; -1------ GPPU disabilitato
           ; --0----- clock interno
           ; ---1---- falling
           ; ----1--- prescaler al WDT
           ; -----111 1:256
    movlw  b'11011111'
    OPTION

; inizializzazioni dell' I/O al reset
    movlw  0x03       ; preset RC1:0 a 1
    movf   PORTC

; Assegna a RC1:0 la funzione di uscita digitale
    movlw  b'11111100'
    tris   PORTC
    goto   ml

;#################################################################
;                       SUBROUTINE DI TEMPO
;
; clock 14.318MHz tcyc 0.0002793686s
; ritardo 0,05 secondi = 178975 cicli
```

```
; ritardo 0.0499988825255 secondi = 178971 cicli
Delay_ms                ; 178971 cicli
    movlw  0xD0         ; +178963 cicli
    movwf  d1
    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 cicli (inclusa call)

; main loop
ml:
;1
    call  Delay_ms      ; 178971 cicli
                        ; +2 cicli del loop
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
;2
    call  Delay_ms      ; 178971 cicli
    goto  $+1           ; +2
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
;3
    call  Delay_ms      ; 178971 cicli
    goto  $+1           ; +2
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
;4
    call  Delay_ms      ; 178971 cicli
    goto  $+1           ; +2
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
;5
    call  Delay_ms      ; 178971 cicli
    goto  $+1           ; +2
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
;6
    call  Delay_ms      ; 178971 cicli
    goto  $+1           ; +2
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
;7
    call  Delay_ms      ; 178971 cicli
    goto  $+1           ; +2
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
;8
    call  Delay_ms      ; 178971 cicli
    goto  $+1           ; +2
    movlw millis        ; +1
    xorwf PORTC,f       ; +1 = 178975
```

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

    goto  ml            ; chiusura loop - 2 cicli

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

9A_1Hz.asm

```
;*********************************************************************
;---------------------------------------------------------------------
;
; Titolo        : Corso Assembly & C - Esercitazione 8A
;                 Genera un impulso da 100ms ogni secondo su GP2.
;                 Clock a quarzo 3.2768MHz
; PIC           : 12F508/09
; Supporto      : MPASM
; Versione      : V.508-1.0
; Data          : 01-05-2013
; Ref. hardware :
; Autore : afg
;---------------------------------------------------------------------
; Impiego pin :
; ----------------
; 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: Out impulso
; GP1/ICSPCLK    6:
; GP0/ICSPDAT    7:
; Vss            8: --
```

```
;
;*****************************************************************
;             DEFINIZIONE DI IMPIEGO DEI PORT
;
;GPIO map
;| 5   | 4 |  3 | 2 | 1 | 0 |
;|-----|---|----|---|---|---|
;| LED |   | in |   |   |   |
;
;#define GPIO,GP0 ;
;#define GPIO,GP1 ;
;#define GPIO,GP5 ;
;#define GPIO,GP3 ; solo input
;#define GPIO,P4  ;
#define  pulse GPIO,GP2 ; uscita impulso
;
;#################################################################
;               scelta del processore
 #ifdef __12F509
   LIST p=12F509 ; Definizione del processore
 #include <p12F509.inc>
 #endif
 #ifdef __12F508
   LIST p=12F508 ; Definizione del processore
 #include <p12F508.inc>
 #endif
   radix dec

;#################################################################
;                 CONFIGURAZIONE
;
; Oscillatore esterno LP, no WDT, no CP, no MCLR
 __config _LP_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF

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

; calibrazione oscillatore interno
;   movwf OSCCAL
;#################################################################
;               MAIN PROGRAM
;
MAIN:
; inizializzazioni al reset
    clrf  GPIO          ; preset GPIO latch a 0

; TRISGPIO   --111011  GP2 out
     movlw b'11111011'
     tris  GPIO         ; al registro direzione

; imposta Timer0 per prescaler 1:32
; disabilita T0CKI per avere GP2 come I/O digitale
      movlw  b'11010100'
      OPTION
```

```
; attesa overflow
oflww movf  TMR0,w
      skpnz
       goto oflww
; overflow del timer - invia impulso
      bsf   pulse     ; pulse = 1
Cy1   movlw .25       ; attesa per 25 count = 0.1s
      subwf TMR0,w    ; ok?
      skpc
       goto Cy1
      bcf   pulse     ; pulse = 0

      goto  oflww
;*******************************************************************
;                        THE END
    END
```

9A_50-1Hz.asm

```
;*******************************************************************
;-------------------------------------------------------------------
;
; Titolo        : Corso Assembly & C - Esercitazione 8A
;                 Genera 50Hz e 1Hz.
;                 Clock a quarzo 3.2768MHz
; PIC           : 12F508/09/19
; Supporto      : MPASM
; Versione      : V.509-1.0
; Data          : 01-05-2013
; Ref. hardware :
; Autore        : afg
;
; ******************************************************************
;-------------------------------------------------------------------
;
; Impiego pin :
; ----------------
; 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: --
;
;*******************************************************************
; DEFINIZIONE DI IMPIEGO DEI PORT
;
;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 ; solo input
#define hz50pin GPIO,GP2 ; uscita 50Hz
#define hz1pin  GPIO,GP1 ; uscita 1Hz

hz50 EQU 4
hz1 EQU 2
;
;#####################################################################
; scelta del processore
 #ifdef __12F509
   LIST p=12F509 ; Definizione del processore
   #include <p12F509.inc>
 #endif
 #ifdef __12F508
   LIST p=12F508 ; Definizione del processore
   #include <p12F508.inc>
 #endif
 #ifdef __12F519
   LIST p=12F519 ; Definizione del processore
   #include <p12F519.inc>
 #endif

   radix dec

;#####################################################################
;                          CONFIGURAZIONE
;
; Oscillatore esterno, 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

;#####################################################################
; VARIABILI

hzcycle EQU D'50' ; numero di cicli per 1Hz


;#####################################################################
; MEMORIA RAM
;
; general purpose RAM
        UDATA     ; inizio area RAM
hzcntr res 1      ; contatore per 1Hz

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

     bsf   hz50pin     ; set 50Hz output high
     bsf   hz1pin      ; set 1Hz output high

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

; imposta Timer0 per clock interno, 1:256
;           b'11010111'
;             1------- GPWU disabilitato
;             -1------ GPPU disabilitato
;             --0----- clock interno
;             ---1---- falling
;             ----0--- prescaler al Timer0
;             -----111 1:256
     movlw  b'11010111'
     OPTION

     movlw  hzcycle    ; pre inizializza contatore 1Hz
     movwf  hzcntr
     clrf   TMR0       ; clear TMR0

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

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

```
     skpc                ; = ?
      goto  Cy1          ; minore - attesa
     movlw  hz50         ; uguale - inverte uscita 50Hz
     xorwf  GPIO,f
     call   OneHz        ; verifica per uscita 1Hz

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

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

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

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

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

Cy7  movlw  D'224'       ; attesa per TMR0=224 (70 msec)
     subwf  TMR0,W
     skpc
      goto  Cy7
     movlw  hz50
     xorwf  GPIO,f
     call   OneHz

Cy8  movf  TMR0,W       ; attesa per TMR0=0 (256) (80 msec)
     skpz
      goto Cy8
```

```
     movlw hz50
     xorwf GPIO,f
     call  OneHz

     goto  Cy1        ; loop

; ****************************************************************
; SUBROUTINE
;
; OneHz - toggle uscita 1Hz ogni 50 cilci da 10ms

OneHz
      decfsz hzcntr,f  ; counter=counter-1 = 0?
       retlw 0         ; no - ritorno
      movlw  hz1       ; si - toggle 1Hz out
      xorwf  GPIO,f
      movlw  hzcycle   ; preset per prossimo conteggio
      movwf  hzcntr
      retlw  0

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