Silvio Navaretti (Fedhman)

PIC12F683, C: INTERVALLOMETRO FOTOGRAFICO

23 September 2016

Questo articolo raccoglie i miei appunti e considerazioni per costruire un intervallometro a 16 tempi con un PIC12F683. All'interno è spiegato come il micro attiva in sequenza dei LED di stato, li fa lampeggiare, riconosce la pressione di un pulsante con anti rimbalzo software e comprende se un pulsante è stato premuto a lungo. Viene mostrato come usare dei flag a singolo bit in un unico registro da 8 bit. Inoltre è trattato il modo più preciso che mi è venuto in mente per misurare 30s usando un timer con prescaler, postscaler e comparatore (con lo zampino di MATLAB). Si accenna alla codifica Charlieplexing per 6 LED pilotati da 3 GPIO.
Magari ci sono modi migliori per fare le stesse cose - "Finché si vive, sempre s'impara". Non mi sono messo a cercare tecniche particolari, il progetto è nato per hobby - e infatti MATLAB c'è cascato dentro.
Mentre scrivevo l'articolo ho fatto diverse modifiche, spero di aver schiacciato tutti i bug :)
Ho dato per scontato (quasi) tutto quello che ho scritto nell'articolo Iniziare coi PIC in C.

DISCLAIMER: non è trattato il modo per pilotare la Canon 40D, NON collegare direttamente il circuito alla macchina. Declino qualunque responsabilità per danni a persone/animali/cose.

Introduzione: il tempo di esposizione

Esposizione: termine che indica la quantità di luce che illumina il sensore fotografico. In parole povere quanto è luminosa la foto scattata.
Quando il fotografo confronta l'esposizione di due foto ragiona su tre parametri: tempo di esposizione, ISO e apertura del diaframma.
Per misurare le variazioni di esposizione **relative** (di una foto rispetto a un'altra) si usa il concetto di stop. Una foto scattata ad "uno stop sopra un'altra" è il doppio più luminosa dell'altra. Invece una foto scattata "uno stop sotto l'altra" è luminosa la metà della precedente.
Il tempo di esposizione è la quantità di tempo in cui l'otturatore rimane aperto davanti al sensore. Approssimativamente è la quantità di tempo in cui il sensore rimane illuminato dalla scena che stiamo fotografando. Allungare o accorciare il tempo di esposizione permette di cambiare l'esposizione (quanto è luminosa la foto).

- Raddoppiare il tempo di esposizione: l'esposizione aumenta di uno stop.
- Dimezzare il tempo di esposizione: l'esposizione cala di uno stop.

Ad esempio, scattando una foto a 1/100s e poi scattandone un'altra a 1/200s il tempo di esposizione è dimezzato: la foto scattata a 1/200s di secondo avrà l'esposizione di uno stop più

bassa rispetto a quella da 1/100s.
Passando da 1/2s a 2s il tempo di esposizione è stato raddoppiato due volte: l'esposizione è aumentata di due stop.

Tutte le reflex e la maggior parte delle compatte avanzate (incluse le bridge) permettono la regolazione manuale del tempo di esposizione. Salti di uno stop tipici di una semiprofessionale/professionale:

```
1/8000s, 1/4000s, 1/3000s,  1/2000s, 1/1000s, 1/500s, 1/250s, 1/125s, 1/60s, 1/30s, 1/1
1/8s, 1/4s, 1/2s, 1s, 2s, 4s, 8s, 15s, 30s
```

Generalmente si possono impostare valori di 1/2 stop o 1/3 stop (ad esempio 1/1250s, 1/800s, 1/640s).
Notare il limite a 30s. Se il fotografo vuole scattare per un tempo superiore deve usare la posa BULB che permette di far scattare la macchina tanto a lungo quanto viene premuto il pulsante di scatto.

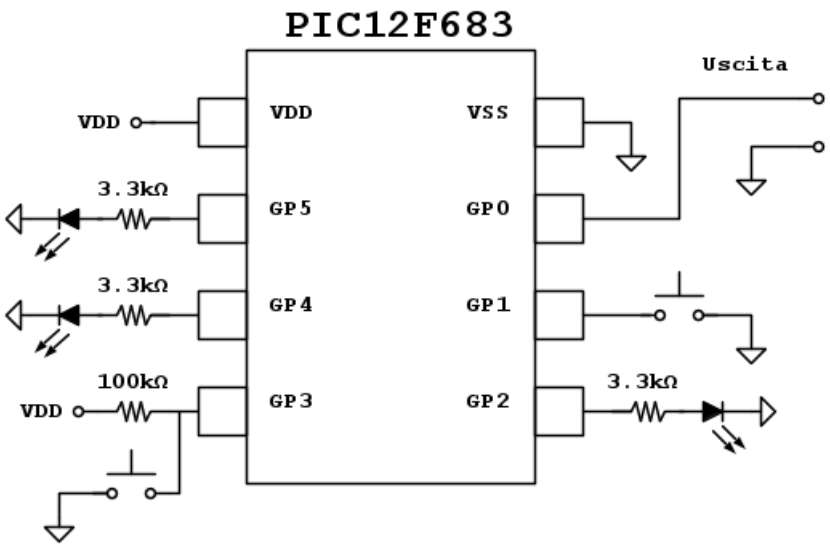
Intervallometro, GPIO disponibili e codifica dei tempi

L'intervallometro è uno strumento che misura intervalli di tempo. Ne costruisco uno con un PIC12F683 per pilotare la posa BULB della mia Canon 40D.

- Serve un pulsante che comandi al PIC di far scattare la macchina fotografica.
- Un altro pulsante selezionerà il tempo desiderato. I tempi andranno a passo di mezzo stop.
- Dei LED accesi o spenti segnaleranno il tempo impostato.
- Un pin di GPIO (General Purpose Input Output) servirà per pilotare l'uscita. Userò GP0.
- Il circuito sarà alimentato da un paio di batterie AAA (tensione di circa 2.5V o meno. Il PIC12F683 è alimentabile da 2V a 5.5V). Cerco di risparmiare quanta più potenza possibile. Spengo i LED quando non servono.

Ricapitolando mi servono due pulsanti (due pin di GPIO) ed un pin di GPIO che userò per pilotare la macchina. Siccome il PIC12F683 ha 6 pin di GPIO ne rimangono tre ancora disponibili: li uso per accendere e spegnere i LED che mostrano il tempo selezionato.

- **Nota1**: Bisogna fare attenzione a GP3: è una porta di I/O che può fungere solo da INPUT. Quindi non si può usarla per pilotare i LED nè per pilotare la reflex. Collegherò a GP3 il pulsante per selezionare il tempo.
- **Nota2**: GP3 non disponde di un resistore di pull up interno. Quindi bisogna metterlo all'esterno. Tutti gli altri pin di GPIO hanno resistori di pull up interni, impostabili individualmente.



Premendo il pulsante di GP3 si modificano i tempi, premendo quello di GP1 si fa scattare la reflex. Un'ulteriore pressione del pulsante di GP1 interromperà subito lo scatto.
Mentre la reflex scatta non occorre che i LED siano accesi. Verranno spenti per risparmiare energia.
La corrente nei LED sarà:

$$I_{LED} \simeq \frac{V_{DD} - V_{LED}}{3.3\ k\Omega}$$

Con 2 batterie AAA cariche e LED rossi, VDD = 2.5V, VLED ~1.8V, ILED ~212uA. In pieno Sole non sono quasi visibili ma userò l'intervallometro di notte...in ogni caso basta schermarli con la mano.

Codifica con 3 LED

Nomi dei LED:

- L5 è il LED connesso a GP5.
- L4 è il LED connesso a GP4.
- L2 è il LED connesso a GP2.

Il tempo di esposizione massimo della 40D è pari a 30s. Parto da 1min e proseguo a incrementi di mezzo stop: 1.5min, 2min, 3min, 4min e via dicendo. Ad ogni pressione del pulsante di GP3 si passerà al tempo successivo. Arrivati all'ultimo tempo potrà premere di nuovo il pulsante di GP3 per tornare al primo.
Userò una codifica binaria. Un LED acceso corrisponde a un 1, un LED spento a uno 0.

L2	L4	L5	Tempo
0	0	0	1 min
0	0	1	1.5 min
0	1	0	2 min
0	1	1	3 min
1	0	0	4 min
1	0	1	6 min
1	1	0	8 min
1	1	1	12 min

Codifica binaria con 3 LED.

Purtroppo i tempi disponibili sono soltanto 8. Per aumentarli a 16 inserisco la possibilità di premere a lungo (1s) il pulsante di GP3 per far lampeggiare i LED accesi. Coi LED che lampeggiano si stanno selezionando i tempi da 9 a 16. Premendo di nuovo per 1s il pulsante di GP3 i LED smetteranno di lampeggiare.

I LED a 1 non lampeggiano

L2	L4	L5	Tempo
0	0	0	1 min
0	0	1	1.5 min
0	1	0	2 min
0	1	1	3 min
1	0	0	4 min
1	0	1	6 min
1	1	0	8 min
1	1	1	12 min

I LED a 1 lampeggiano

L2	L4	L5	Tempo
0	0	0	16 min
0	0	1	24 min
0	1	0	32 min
0	1	1	48 min
1	0	0	64 min
1	0	1	96 min
1	1	0	128 min
1	1	1	192 min

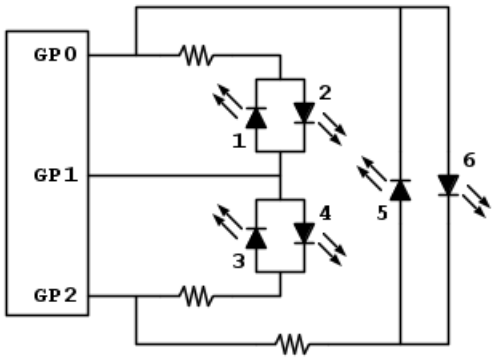
- Siccome si cambia tempo ad ogni pressione del pulsante, anche la lunga pressione di 1s farà passare l'intervallometro al tempo successivo. Non è molto importante perché si ha perennemente un riscontro visivo del tempo disponibile.
- Quando tutti i LED sono spenti non si sa quale sia il tempo: se il primo (1min) o il nono (16min). Pigiando il pulsante il fotografo si accorge subito se i LED lampeggiano o meno.

Sperimentalmente mi sembra che far lampeggiare i LED con periodo 1/3s (1/6s ON, 1/6s OFF) sia abbastanza gradevole alla vista.

Codifica Charlieplexing con 6 LED

C'è un modo, proposto dalla Microchip nel file **8-pin Flash PIC Microcontrollers Tips 'n Tricks** (pagina 7), per pilotare 6 LED con 3 porte di GPIO e 3 resistori. La tecnica è detta Charlieplexing, rimando all'articolo Il Charlieplexing di posta10100.
Z sta per alta impedenza.

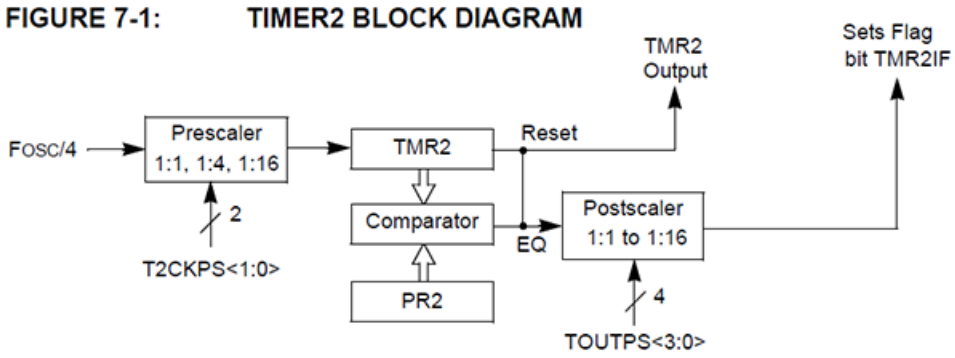
GP			LED					
0	1	2	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
0	1	Z	1	0	0	0	0	0
1	0	Z	0	1	0	0	0	0
Z	0	1	0	0	1	0	0	0
Z	1	0	0	0	0	1	0	0
0	Z	1	0	0	0	0	1	0
1	Z	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0	0
0	1	1	1	0	0	0	1	0
1	0	0	0	1	0	0	0	1
1	0	1	0	1	1	0	0	0
1	1	0	0	0	0	1	0	1
1	1	1	0	0	0	0	0	0



Non ho verificato lo schema, lo lascio come appunto. Le combinazioni non permettono di mostrare tutti i numeri da 0 a 63, però si potrebbe passare rapidamente da una combinazione all'altra, sfruttando la lentezza del cervello umano. Ad esempio per mostrare 110000 bisognerebbe alternare velocemente (più di 1/50s a dir tanto) le combinazioni 0 1 Z e 1 0 Z. Non sarebbe un problema per il microcontrollore.
Non vale la pena complicare il progetto per disporre di 64 tempi, sono troppi per il mio intervallometro.

Scegliere la frequenza di clock e contare 30s col timer2

Il timer2 del PIC12F683 ha 8 bit e dispone di prescaler, postscaler e comparatore interno. Ho misurato il tempo di esposizione tramite gli interrupt del timer2. Vi sarà un interrupt ogniqualvolta il timer2 andrà in overflow, scandendo quindi il tempo di overflow in overflow - ma bisogna considerare la presenza del postscaler e del comparatore interno.



Schema del timer2. Pagina 51 del datasheet.

Il timer2 è a 8 bit: conta 2^8 = 256 volte e poi va in overflow. Tuttavia il valore del timer2 viene continuamente comparato col numero salvato in PR2: appena il timer2 incrementa raggiungendo PR2 il timer2 viene resettato, come se andasse in overflow. Il periodo di overflow è pertanto

$$T_{overflow} = \frac{Valore\ in\ PR2 \cdot prescaler}{\frac{1}{4} \cdot f_{oscillatore}} = \frac{Valore\ in\ PR2 \cdot 4 \cdot prescaler}{f_{oscillatore}}$$

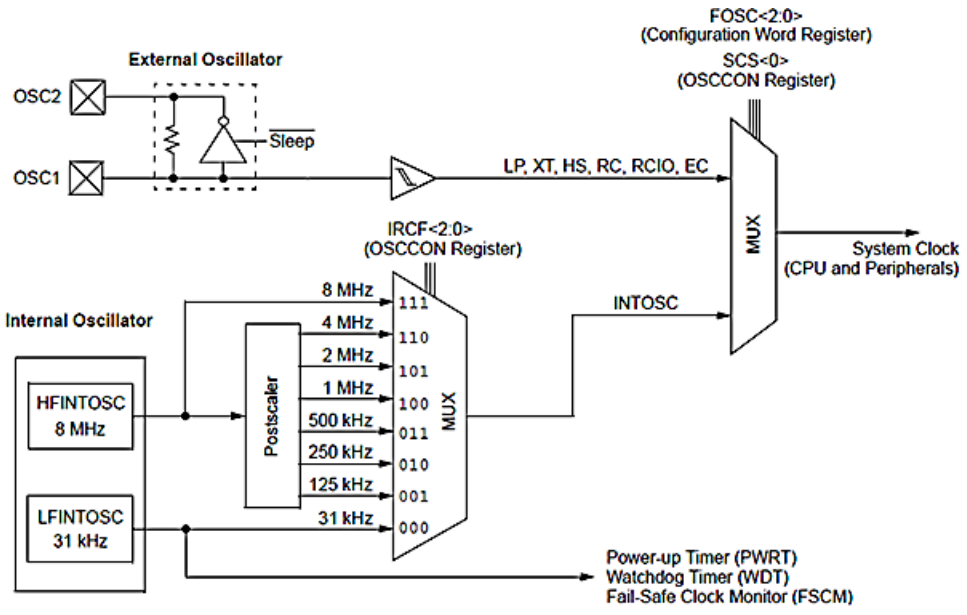
PR2 è un registro a 8 bit, possiamo regolare il tempo di overflow scrivendoci dentro un numero da 0 a 255. Più sarà alto il numero più sarà lungo il periodo di overflow del timer2.
Il postscaler offre la possibilità di ritardare gli interrupt del timer2. Normalmente passato un periodo di tempo pari a Toverflow scatterebbe un interrupt: ma il postscaler conta gli overflow, attivando gli interrupt soltanto dopo un certo numero di overflow. Il periodo di interrupt sarà

$$T_{interrupt} = T_{overflow} \cdot postscaler$$

Con postscaler che può valere da 1 a 16, con passo 1.
Nota: c'è la possiblità di scrivere liberamente nel registro TMR2, cioè il valore "a cui è arrivato il timer2". Scrivendo un valore nel timer si resetta automaticamente il contatore del postscaler. Questo è importante perché permette di "far ripartire" il tempo di interrupt.

Contare 30 secondi

Passato un periodo di tempo pari a Tinterrupt si incrementa una variabile. In tal modo si tiene traccia del tempo totale che è passato. Insomma si scandisce il tempo di Tinterrupt in Tinterrupt. Per non usare altri componenti esterni scelgo l'oscillatore interno e per risparmiare energia opto per una bassa frequenza di clock. Si potrebbe scegliere l'oscillatore interno a basso consumo LFINTOSC a 31kHz, **ma non è calibrato di fabbrica**. Ripiego su HFINTOSC settando IRCF2 = 0, IRCF1 = 0 e IRCF0 = 1 per ottenere 125kHz dal suo postscaler.



Schema dell'origine del clock, pagina 21 del datasheet.

Il prescaler del timer2 vedrà un clock a 125kHz/4 = 31.25kHz (periodo 32us).

$$T_{interrupt} = 32\mu s \cdot PR2 \cdot prescaler \cdot postscaler$$

Si possono scegliere liberamente tre parametri per regolare Tinterrupt:

- PR2 può valere da 1 a 256 (scrivendo poi da 0 a 255 nel registro, si inizia a contare da zero)
- prescaler può valere 1, 4 o 16.
- postscaler può valere da 1 a 16.

Uso una variabile per contare il tempo, **scattiInterrupt**. Appena scatta l'interrupt del timer2 la incremento di uno. Facendola partire da 0, passati 30s il suo valore sarà

$$scattiInterrupt_{dopo\ 30s} = \frac{30s}{T_{interrupt}}$$

Arrotondato per eccesso ad intero (scelgo di accorgermi del passaggio di 30s in ritardo). Questo numero è importante perché sarà il valore massimo salvato nella variabile del programma e tale valore non può superare una certa cifra.
Sto usando un microcontrollore a 8 bit. I suoi registri di memoria dati sono a 8 bit (o almeno così sembra: pagine 9-10 del datasheet). Userò variabili **unsigned char** per non dovermi preoccupare di come (e se...) il compilatore gestisce variabili int e long.
Le variabili unsigned char avranno quindi valore massimo pari a 255. Questo è un problema perché Tinterrupt va scelto in modo che il valore massimo di scattiInterrupt sia inferiore a 255.

Un esempio che non funziona: scegliamo PR2 = 13, prescaler = 1 e postscaler = 1. Otteniamo Tinterrupt = 416us, quindi ogni 416us scatterà un interrupt del timer2 e scattiInterrupt verrà incrementata. Passati 30s scattiInterrupt varrà 30s/416us = 72115.38462 = 72116. È maggiore di 255, quella combinazione di PR2, prescaler e postscaler non va bene.
Un esempio che funziona: scegliamo PR2 = 256, prescaler = 4 e postscaler = 8. Tinterrupt = 262.144ms e il valore massimo di scattiInterrupt sarà 114.440918.
Però ci si accorgerà del passaggio di un minuto quando scattiInterrupt varrà 115, all'interrupt successivo. Quindi un po' in ritardo rispetto al valore ideale 114.440918.

Conclusione:
Vi sono tre variabili che posso impostare, PR2, prescaler e postscaler. Variandole cambio il periodo di interrupt. Inizializzo una variabile a 0 e ogni qualvolta viene servito l'interrupt la incremento. Il suo valore massimo sarà 30s/(periodo interrupt); ad ogni interrupt confronto tale valore con quello attualmente salvato nella variabile. Appena sono uguali capisco che sono passati 30s e resetto a 0 la variabile.
Il sistema sarebbe più preciso se il valore massimo della variabile fosse intero. Al contempo però dev'essere minore di 255 per poterlo salvare in un registro da 8 bit.

La miglior combinazione di PR2, prescaler e postscaler per contare 30 secondi

Lascio fare a MATLAB:

```
format long;
fOscillatore = 125e3;
periodoOverflow = 1/(fOscillatore/4);
tempo = 30; %impostati 30s
PR2 = 1:1:256; %il registro va invece da 0 a 255
scartoMinAssoluto = 256; %valore iniziale
for prescaler = [1 4 16]
    for postscaler = 1 : 1 : 16
        Toverflow = periodoOverflow.*PR2.*prescaler.*postscaler;
        scarto = tempo./Toverflow - floor(tempo./Toverflow);
        [scarto_min, valorePR2] = min(scarto);
        numInterruptMax = tempo/(prescaler*postscaler*periodoOverflow*valorePR2);
        if scartoMinAssoluto > scarto_min && numInterruptMax < 256 && numInterruptMax >
            scartoMinAssoluto = scarto_min;
            risultati = table(scartoMinAssoluto, numInterruptMax,...
                valorePR2, prescaler, postscaler)
        end
    end
end
```

Risulta che con PR2 = 125, prescaler = 4 e postscaler = 15 il numero di interrupt per contare 30s è esattamente 125.
Il periodo di interrupt sarà 32us*125*4*15 = 240ms, il valore massimo della variabile di conta degli interrupt 30s/240ms = 125.
L'analisi dei grafici dello scarto è abbastanza interessante - ma non è così rilevante e l'articolo è lungo.

Interrupt del timer0 (la suocera): verifica dello stato dei pulsanti e aggiornamento dei LED

Userò l'interrupt di overflow del timer0 per effettuare periodicamente i seguenti controlli/azioni:

- Verifica dell'avvenuta pressione del pulsante di GP3, d'ora in poi chiamato pulsante3. La pressione del pulsante3 sarà valida soltanto se in precedenza il pulsante3 non era premuto.
- Se la pressione del pulsante3 è valida, cambia lo stato dei LED, passando al tempo successivo.
- Se la pressione del pulsante3 sta perdurando per diversi periodi di interrupt del timer0 (cioè se il fotografo sta premendo il pulsante a lungo), attiva o disattiva il lampeggio dei LED.

- Verifica dell'avvenuta pressione del pulsante di GP1, d'ora in poi chiamato pulsante1. La pressione del pulsante1 sarà valida soltanto se in precedenza il pulsante1 non era premuto.
- Se è la prima volta che pulsante1 viene premuto, fai scattare la macchina, spegni i LED e disattiva le azioni eseguite premendo pulsante3. Altrimenti interrompi lo scatto, riaccendi i LED come prima e riattiva le azioni di pulsante3.

Tutto questo verrà eseguito ad ogni overflow del timer0. Il timer0 ha 8 bit dispone di un prescaler con valori

```
1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:128, 1:256
```

la frequenza dell'oscillatore è 125kHz, al prescaler del timer0 arrivano 125kHz/4 = 31.25kHz. Il periodo che arriva al prescaler è 1/31.25kHz = 32us, quindi il periodo di overflow del timer0 sarà 32us*256*prescaler.
Scelgo di impostare il prescaler a 1:4, ottenendo Toverflow = 32us*256*4 = 32.768ms.
In tal modo il timer0 farà scattare un interrupt 1s/32.768ms = ~30.52 volte al secondo, più che sufficienti per monitorare costantemente le azioni di chi premerà i pulsanti.

Ricordare lo stato dei pulsanti, antirimbalzo software

Verifico ~30 volte al secondo se i pulsanti è stato premuti. Ci sono due problemi da considerare:

1. Se il fotografo è lento potrebbe premerli per più di un periodo di interrupt del timer0. Ad esempio se preme pulsante3 per 70ms il micro lo vede premuto per due interrupt del timer0. Quindi esegue due volte le azioni dovute...non va bene.
2. Ogni volta che qualcuno preme un pulsante il suo contatto meccanico rimbalza su e giù più volte. Si adottano tecniche di antirimbalzo (debouncing) per non campionare i rimbalzi. Non ho misurato la frequenza media dei rimbalzi ma sperimentalmente mi sono accorto che è più rapida di quella degli interrupt del timer0; i rimbalzi si esauriscono tra un interrupt e l'altro. Li stiamo ignorando via software.

Per rimediare al punto 1 servono due variabili che si ricordino lo stato precedente del pulsante. Con poca fantasia le ho chiamate GP3_precedente e GP1_precedente. L'idea è la stessa per entrambi i pulsanti (esempio con pulsante3):
NOTA: le porte di I/O hanno resistori di pullup a VDD (1). Quando il fotografo preme un pulsante mette a GND (0) quella porta.

- Il micro si avvia, GP3_precedente = 1, nessuno stava premendo il pulsante.
- Il fotografo d'ora in poi sta premendo pulsante3.
- Dopo pochissimo tempo (millisecondi) scatta un interrupt del timer0.
- La ISR dell'interrupt trova GP3 == 0 perché il fotografo sta premendo il pulsante. Inoltre legge GP3_precedente == 1. Quindi esegue le azioni dovute alla pressione del pulsante3 e imposta GP3_precedente = 0.

- Scatta un altro interrupt del timer0. Questa volta la ISR trova GP3 == 0 (il fotografo sta ancora premendo il pulsante) ma al contempo si accorge che GP3_precedente == 0, quindi capisce che all'interrupt precedente il pulsante era già premuto. La ISR non fa alcuna azione che riguarda la pressione di pulsante3 (ma magari fa altro che **non** riguarda pulsante3).
- Il fotografo smette di premere il pulsante3.
- Scatta un altro interrupt del timer0. Ora la ISR trova GP3 == 1, pertanto imposta GP3_precedente = 1 e non fa alcuna azione che riguarda il pulsante3. Ora il sistema è pronto ad acquisire la prossima pressione del pulsante.

Si sfrutta l'idea che gli interrupt avvengano ~30 volte al secondo, molto più rapidamente di come agisce il fotografo.

Verificare se GP3 è stato premuto a lungo

Quanto a lungo? Scelgo 32.768*30 = 983.04ms. Quindi dovrò contare 30 interrupt del timer0, servirà una variabile che funga da contatore. Ho usato una unsigned char (8bit), l'ho chiamata nel codice **contatoreAlternaLampeggia** (i nomi lunghi e chiari aiutano la mia pessima memoria). Bisogna contare solamente se il pulsante è premuto ed era premuto anche all'interrupt precedente (GP3 == 0, GP3_precedente == 0). Una volta arrivato a contatoreAlternaLampeggia == 30 inverto lo stato di un bit di flag chiamato **LampeggiaLED**.

Gestire 7 bit di flag in un singolo registro da 8 bit

Flag: una variabile da 1 bit che mi serve per segnalare qualcosa. Può assumere solo due valori, 0 oppure 1.
Il PIC12F683 ha 128B di dati, cioè 128 registri da 8 bit.
Potrei tranquillamente usare un registro da 8 bit per ogni flag. Sprecherei 7 degli 8 bit di ogni registro ma la quantità di memoria disponibile è più che sufficiente.
Questo weekend sono in vacanza, mi prendo un po' di tempo in più e faccio le cose per bene: metto tutti e 7 i flag che mi servono in un unico registro da 8 bit, di fatto risparmiando 6 registri.

```
#define qL2ON FLAGS & 1     //Negli if i define di query (che iniziano con q,
#define onL2ON FLAGS |= 1   //come qL2ON) verranno controllati con == 0 oppure
#define offL2ON FLAGS &= ~1 //senza scrivere == 1, siccome contengono 0 oppure
#define qL4ON FLAGS & 2     //delle potenze di 2.
#define onL4ON FLAGS |= 2
#define offL4ON FLAGS &= ~2             //REGISTRO FLAGS
#define qL5ON FLAGS & 4                 //bit   funzione
#define onL5ON FLAGS |= 4               //0     L2ON
#define offL5ON FLAGS &= ~4             //1     L4ON
#define qGP1_precedente FLAGS & 8       //2     L5ON
#define onGP1_precedente FLAGS |= 8     //3     GP1_precedente
```

```
#define offGP1_precedente FLAGS &= ~8   //4     GP3_precedente
#define qGP3_precedente FLAGS & 16      //5     lampeggiaLED
#define onGP3_precedente FLAGS |= 16    //6     scattoAttivo
#define offGP3_precedente FLAGS &= ~16  //7     nessuna funzione
#define qLampeggiaLED FLAGS & 32
#define onLampeggiaLED FLAGS |= 32
#define offLampeggiaLED FLAGS &= ~32
#define qScattoAttivato FLAGS & 64
#define onScattoAttivato FLAGS |= 64
#define offScattoAttivato FLAGS &= ~64

unsigned char FLAGS = 0b00011111;
```

Il registro da 8 bit si chiama FLAGS. Ognuno dei suoi bit ha un significato diverso. Ho scritto dei define che corrispondono alle seguenti operazioni (esempio sul flag GP1_precedente):

Operazione di query (il bit è ad 1?)

qGP1_precedente sta per queryGP1_precedente, ovvero chiedi se quel bit è 1. Il flag GP1_precedente è il bit 3 del registro, quindi con l'operazione FLAGS AND (00001000 = 2^3 = 8) otteniamo due possibili valori: se GP1_precedente è a zero la AND ci dà il risultato 00000000 = 0, altrimenti otteniamo 00001000 = 8. In C si scrive

```
FLAGS & 8;
```

NOTA: il terzo bit di FLAGS non viene cambiato. Rimane del valore che aveva prima, non stiamo assegnando un valore a quel bit.

Entrare in un if nel caso che il bit sia ad 1

Attenzione, se GP1_precedente è ad 1 dalla AND bit a bit esce un 8, non un 1! È importante per gli if, non possiamo scrivere

```
if(FLAGS & 8 == 1){
}
```

perché risulterà sempre falso! Da quell'operazione si può ottenere soltanto 0 oppure 8.
Ho così due opzioni equivalenti per entrare in un if con GP1_precedente a 1, scrivere

```
if(FLAGS & 8 != 0){
}
```

oppure

```
if(FLAGS & 8){
}
```

Entrare in un if nel caso il bit sia a 0

È sufficiente scrivere

```
if(FLAGS & 8 == 0){
}
```

Semplifichiamoci la vita: usiamo il define

Ho scritto i #define perché mi scordo continuamente il significato dei bit del registro FLAGS. Per entrare nell'if col bit a 1 basta scrivere

```
if(qGP1_precedente){
}
```

Invece per entrare col bit a 0:

```
if(qGP1_precedente == 0){
}
```

Mettere 1 in un bit del registro FLAGS

Prendiamo come esempio il bit 5, LampeggiaLED. Per alzare a 1 il quinto bit e lasciare gli altri immutati bisogna fare la OR bit a bit tra flags e 00010000 = 2^5 = 32, aggiornando il valore del bit 5 di FLAGS. In C si scrive

```
FLAGS = FLAGS |= 32;
```

oppure più rapidamente

```
FLAGS |= 32;
```

Scrivendo il define

```
#define onLampeggiaLED FLAGS |= 32
```

Basterà scrivere nel programma

```
onLampeggiaLED;
```

per tirare su il bit 5 del registro FLAGS, ovvero asserire il bit che userò per far lampeggiare i LED.

Mettere 0 in un bit del registro FLAGS

Prendendo ad esempio il bit 6, scattoAttivato:
Per metterci dentro 0 bisogna fare la AND bit a bit tra FLAGS e il numero 11011111 = NOT 2^6 = NOT 64. In C si scrive

```
FLAGS = FLAGS & ~64
```

Oppure più rapidamente

```
FLAGS &= ~64
```

Nuovamente conviene usare il define

```
#define offScattoAttivato FLAGS &= ~64
```

E nel programma scriveremo

```
offScattoAttivato;
```

Il programma

Tutto il programma è mostrato in sequenza, riga per riga, anche se i blocchi sono separati dai miei commenti e dai titoli dei paragrafi.
Il nucleo del programma è la ISR. Nel main non accade nulla fuorché l'inizializzazione delle variabili.
Ad ogni overflow del timer2 si incrementa la variabile **ms240**, chiamata così perché passano 240ms tra un overflow del timer2 e l'altro. La variabile **s30** sta invece per **trenta secondi** e verrà incrementata ogni 30s.

Prima del main

Impostazioni ottenute col tool di configurazione dei bit (MPLABX, Run/Set Configuration Bits)

```
#pragma config FOSC = INTOSCIO /*Oscillator Selection bits (INTOSCIO oscillator:
I/O function on RA4/OSC2/CLKOUT pin, I/O function on RA5/OSC1/CLKIN)*/
#pragma config WDTE = OFF //Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF //Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = OFF //MCLR pin function is digital input
#pragma config CP = OFF //Code Protection bit
#pragma config CPD = OFF //Data Code Protection bit
#pragma config BOREN = OFF //Brown Out Detect (BOR disabled)
#pragma config IESO = OFF //Internal External Switchover bit
#pragma config FCMEN = OFF //Fail-Safe Clock Monitor Enabled bit
```

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

Define del registro FLAGS:

```
#define qL2ON (FLAGS & 1)   //Negli if i define di query (che iniziano con q,
#define onL2ON FLAGS |= 1   //come qL2ON) verranno controllati o con == 0 oppure
#define offL2ON FLAGS &= ~1 //senza scrivere == 1, siccome contengono 0 oppure
#define qL4ON (FLAGS & 2)   //delle potenze di 2.
#define onL4ON FLAGS |= 2
#define offL4ON FLAGS &= ~2             //REGISTRO FLAGS
#define qL5ON (FLAGS & 4)               //bit   funzione
#define onL5ON FLAGS |= 4               //0     L2ON
#define offL5ON FLAGS &= ~4             //1     L4ON
#define qGP1_precedente (FLAGS & 8)     //2     L5ON
#define onGP1_precedente FLAGS |= 8     //3     GP1_precedente
#define offGP1_precedente FLAGS &= ~8   //4     GP3_precedente
#define qGP3_precedente (FLAGS & 16)    //5     lampeggiaLED
#define onGP3_precedente FLAGS |= 16    //6     scattoAttivo
#define offGP3_precedente FLAGS &= ~16  //7     nessuna funzione
#define qLampeggiaLED (FLAGS & 32)
#define onLampeggiaLED FLAGS |= 32
#define offLampeggiaLED FLAGS &= ~32
#define qScattoAttivato (FLAGS & 64)
#define onScattoAttivato FLAGS |= 64
#define offScattoAttivato FLAGS &= ~64
```

Altri define comodi:

```
#define VALORE_PR2 124
#define offUscita GP0 = 0
#define onUscita GP0 = 1
```

Variabili e prototipo della ISR:

```
unsigned char FLAGS = 0b00011111;
unsigned char valoreLED = 0;
unsigned char contatoreAlternaLampeggia = 0;
unsigned char contatoreAlternaStatoLED = 0;
unsigned char ms240 = 0, s30 = 0, min = 0;

void interrupt ISR(void);
```

Il main

Prima vengono impostati tutti i registri per configurare il PIC12F683, poi vengono inizializzate le variabili. Infine c'è ciclo while(1){}, vuoto.

```
void main(int argc, char** argv) {
    //IMPOSTAZIONI DI RESET, Ultra Low-Power Wake-Up Enable bit disabilitato,
    PCON = PCON & 0b000000011; //BOR disabilitato.
    //IMPOSTAZIONI DEL CLOCK
    OSTS = 0; //A 0 il dispositivo funziona con l'oscillatore interno.
    SCS = 1; //A 1 l'oscillatore è quello interno.
    IRCF0 = 1; IRCF1 = 0; IRCF2 = 0; //MUX oscillatore interno a 125 kHz
    //IMPOSTAZIONI DEGLI INTERRUPT GENERICHE
    GIE = 1; //A 1 abilita tutti gli interrupt
    PEIE = 1; //A 1 abilita gli interrupt delle periferiche
    INTE = 0; //GP2/INT External Interrupt Enable bit
    EEIE = 0; ADIE = 0; CCP1IE = 0; CMIE = 0; OSFIE = 0;
    //TIMER0
    T0CS = 0; //Sorgente interna di clock
    PSA = 0; //Assegna il prescaler al timer0
    PS0 = 1; PS1 = 0; PS2 = 0; //Prescaler del timer0 a 1:4
    T0IE = 1; //Abilita gli interrupt del timer0;
    //TIMER2
    TMR2ON = 0; //Il timer2 parte SPENTO.
    TMR2 = 0; //Metti 0 fin dall'inizio nel contatore del timer2.
    T2CKPS1 = 0; T2CKPS0 = 1; //Prescaler a 1:4
    TOUTPS3 = 1; TOUTPS2 = 1; TOUTPS1 = 1; TOUTPS0 = 0; //Postscaler a 1:15
    PR2 = VALORE_PR2; //Così conta 125 volte, iniziando da 0.
    TMR2IE = 1; //Abilita gli interrupt del timer2
    //TIMER1, COMPARATORE, CONVERTITORE A/D
    TMR1ON = 0; TMR1IE = 0; //Disabilita il timer1 ed i suoi interrupt
    CM0 = 1; CM1 = 1; CM2 = 1; //Tutti i pin del comparatore come I/O
    ADON = 0; //Disabilita il convertitore A/D

    //GPIO
    OPTION_REG = OPTION_REG & 0b01111111; /*Mette 0 in GPPU, resistori di
        pullup settabili coi bit WPU*/
    IOC5 = 0; IOC4 = 0; IOC3 = 0; IOC2 = 0; IOC1 = 0; IOC0 = 0; //Interrupt disabilitat
    ANS3 = 0; ANS2 = 0; ANS1 = 0; //Porte come I/O digitali.
    //Con TRISIO a 1 porte come input.
    TRISIO5 = 0; TRISIO4 = 0; TRISIO2 = 0; TRISIO1 = 1; TRISIO0 = 0;
     //Resistori di Pull up abilitati con 1.
    WPU5 = 0; WPU4 = 0; WPU2 = 0; WPU1 = 1; WPU0 = 0;
```

```
    offUscita;
    contatoreAlternaLampeggia = 0;
    contatoreAlternaStatoLED = 0;
    ms240 = 0, s30 = 0, min = 0;
    onGP1_precedente; onGP3_precedente;
    offScattoAttivato;
    offLampeggiaLED;
    //All'accensione tutti i LED saranno accesi, con valoreLED = 7 alla prima
    //pressione di GP3 verranno spenti tutti, impostando l'opzione 0.
    onL5ON; onL4ON; onL2ON;
    valoreLED = 7;

    while(1){}
}
```

La ISR

Parte iniziale e commenti:

```
void interrupt ISR(void){
    //Frequenza timer = Foscillatore/4 = 125kHz/4 = 31.25kHz
    //Periodo timer = 1/31.25kHz = 32 us
    //Timer0, prescaler 1:4, overflow ogni 256*4*32us = 32.768ms
    //Timer2, con PR2 (comparatore) a 124, prescaler 1:4, postscaler 1:15,
    //Timer2 overflow ogni = 32us*125*4*15 = 240ms
    //Per contare 30s occorrono 30s/240ms = 125 interrupt
```

L'if dell'interrupt di overflow del timer2

```
    if(TMR2IF == 1){ //OVERFLOW DEL TIMER2
        ms240 = ms240 +1; //Sono passati 240ms
        if(ms240 == 125){
            ms240 = 0;
            s30 = s30 +1; //Sono passati 30s
        }
        if(s30 == 2){ //E' passato 1 minuto
            s30 = 0;
            min = min+1;
            if(qLampeggiaLED == 0){ //CASO dei led che NON lampeggiano
                switch(valoreLED){
                    case 0: if(min == 0){offScattoAttivato; offUscita;} break;
                    case 1: break; //minuto e mezzo trattato dopo
```

```
                    case 2: if(min == 2){offScattoAttivato; offUscita;} break;
                    case 3: if(min == 3){offScattoAttivato; offUscita;} break;
                    case 4: if(min == 4){offScattoAttivato; offUscita;} break;
                    case 5: if(min == 6){offScattoAttivato; offUscita;} break;
                    case 6: if(min == 8){offScattoAttivato; offUscita;} break;
                    case 7: if(min == 12){offScattoAttivato; offUscita;} break;
                    default: offScattoAttivato; offUscita; valoreLED = 0; break;
                }
            }
            else{ //CASO dei led che lampeggiano
                switch(valoreLED){
                    case 0: if(min == 16){offScattoAttivato; offUscita;} break;
                    case 1: if(min == 24){offScattoAttivato; offUscita;} break;
                    case 2: if(min == 32){offScattoAttivato; offUscita;} break;
                    case 3: if(min == 48){offScattoAttivato; offUscita;} break;
                    case 4: if(min == 64){offScattoAttivato; offUscita;} break;
                    case 5: if(min == 96){offScattoAttivato; offUscita;} break;
                    case 6: if(min == 128){offScattoAttivato; offUscita;} break;
                    case 7: if(min == 192){offScattoAttivato; offUscita;} break;
                    default: offScattoAttivato; offUscita; valoreLED = 0; break;
                }
            }
        }
        //CASO DEL MINUTO E MEZZO, opzione 1 quando i led non lampeggiano
        //Con s30 = 1 sono passati 30 secondi, più 1 min fa 1.5 min
        if(valoreLED == 1 && qLampeggiaLED == 0 && s30 == 1 && min == 1){
            offUscita;
            offScattoAttivato;
        }
        TMR2IF = 0;  //RIPRISTINO INTERRUPT TIMER2
    }
```

L'if dell'interrupt di overflow del timero

```
    if(T0IF == 1){ //OVERFLOW DEL TIMER0
        //Se GP3 è premuto e PRIMA era sempre premuto, e non scatta, e non e'
        //già cambiato il lampeggio
        if(GP3 == 0 && qGP3_precedente == 0 && qScattoAttivato == 0 && contatoreAlterna
            contatoreAlternaLampeggia = contatoreAlternaLampeggia +1;
            if(contatoreAlternaLampeggia == 30){ //Passati 32.768ms*30 = 983ms
                if(qLampeggiaLED){offLampeggiaLED;}
                else{onLampeggiaLED;}
            }
```

```
}
//Se GP3 è premuto e PRIMA non lo era, e non scatta. Per sicurezza
//azzera contatoreAlternaLampeggia
if(GP3 == 0 && qGP3_precedente && qScattoAttivato == 0){
    offGP3_precedente;
    contatoreAlternaLampeggia = 0;
    valoreLED = valoreLED +1;
    if(valoreLED == 8){
        valoreLED = 0;
    }
    switch(valoreLED){
        case 0: GP5 = 0; GP4 = 0; GP2 = 0;
                offL5ON; offL4ON; offL2ON; break;
        case 1: GP5 = 1; GP4 = 0; GP2 = 0;
                onL5ON; offL4ON; offL2ON; break;
        case 2: GP5 = 0; GP4 = 1; GP2 = 0;
                offL5ON; onL4ON; offL2ON; break;
        case 3: GP5 = 1; GP4 = 1; GP2 = 0;
                onL5ON; onL4ON; offL2ON; break;
        case 4: GP5 = 0; GP4 = 0; GP2 = 1;
                offL5ON; offL4ON; onL2ON; break;
        case 5: GP5 = 1; GP4 = 0; GP2 = 1;
                onL5ON; offL4ON; onL2ON; break;
        case 6: GP5 = 0; GP4 = 1; GP2 = 1;
                offL5ON; onL4ON; onL2ON; break;
        case 7: GP5 = 1; GP4 = 1; GP2 = 1;
                onL5ON; onL4ON; onL2ON; break;
        default: GP5 = 0; GP4 = 0; GP2 = 0;
                offL5ON; offL4ON; offL2ON;
                valoreLED = 0; break; //RIPRISTINO PER SICUREZZA
    }
}
//Se GP3 è stato rilasciato e non scatta, riarmalo e azzera contatoreAlternaLam
if(GP3 == 1 && qGP3_precedente == 0 && qScattoAttivato == 0){
    contatoreAlternaLampeggia = 0;
    onGP3_precedente;
}
//AZZERAMENTO TIMER2
if(qScattoAttivato == 0){
    TMR2ON = 0; //Timer spento
    TMR2 = 0; //Metti 0 nel registro del timer2, resettando a 0 pure
    //i CONTATORI del postscaler e del prescaler.
    offUscita;
```

```
            if(qLampeggiaLED){
                //Eventualmente lampeggia i LED se attivi, ogni 5*32.768ms = 163.84ms
                contatoreAlternaStatoLED = contatoreAlternaStatoLED +1;
                if(contatoreAlternaStatoLED == 5){
                    contatoreAlternaStatoLED = 0;
                    if(qL5ON){GP5 = GP5^1;}
                    if(qL4ON){GP4 = GP4^1;}
                    if(qL2ON){GP2 = GP2^1;}
                }
            }
            else{ //Se non c'è da farli lampeggiare accendili normalmente
                if(qL5ON){GP5 = 1;}else{GP5 = 0;}
                if(qL4ON){GP4 = 1;}else{GP4 = 0;}
                if(qL2ON){GP2 = 1;}else{GP2 = 0;}
            }
        }
        //Se GP1 è premuto e PRIMA non lo era
        if(GP1 == 0 && qGP1_precedente){
            offGP1_precedente;
            if(qScattoAttivato == 0){
                onScattoAttivato;
                ms240 = 0; s30 = 0; min = 0; //Riazzera i contatori.
                PR2 = VALORE_PR2; //Così conta 125 volte, iniziando da 0.
                TMR2 = 0; //Mette 0 nel registro di conta del timer2
                TMR2ON = 1; //Attiva il timer2
                onUscita;
                GP5 = 0; GP4 = 0; GP2 = 0;//Spegni i LED momentaneamente
            }
            else{
                offUscita;
                offScattoAttivato;
                if(qL5ON){GP5 = 1;}else{GP5 = 0;}
                if(qL4ON){GP4 = 1;}else{GP4 = 0;}
                if(qL2ON){GP2 = 1;}else{GP2 = 0;}
            }
        }
        //Se GP1 è stato rilasciato riarmalo
        if(GP1 == 1 && qGP1_precedente == 0){
            onGP1_precedente;
        }
        T0IF = 0; //RIPRISTINO INTERRUPT TIMER0
    }
}//CHIUDE LA FUNZIONE ISR
```

Memoria occupata e qualche idea sulla potenza assorbita dal circuito

Col compilatore XC8 in versione gratuita: occupato il 15% della memoria dati (128B) e il 25% della memoria per i programmi (2kB).
Alimentando il circuito col PICkit3 a 3V la corrente assorbita è di circa 1.05mA con tutti e 3 i LED accesi e 0.13mA con tutti i LED spenti. Non ho effettuato misure (non dichiaro neanche l'incertezza) ma l'idea è quella, siamo intorno a 3mW con 3 LED accesi.

Bibliografia

- Datasheet del PC12F683.
- Il Charlieplexing.
- 8-pin Flash PIC Microcontrollers Tips 'n Tricks.