TIMER PROGRAMMABILE

1 secondo – 60 ore

Autore: Enzo Carollo

Data: 16-2-2013

Introduzione.

Salve a tutti, ho scritto questo codice con l'ambiente di Sviluppo MPLAB IDE e la versione demo del compilatore C HI-TECH per PIC10/12/16.

Una volta compilato, usa circa il 93 % di memoria programma e l' 84 % della RAM, ma con le ottimizzazioni questi valori possono scendere parecchio.

Descrizione Firmware.

Dalle tempistiche in gioco, ho scelto l'utilizzo del Timer1 interno a 16 bit, prescalato di un fattore di divisione pari a 8, con il quale riesco ad ottenere tempi relativamente lunghi.

Avendo a disposizione un oscillatore interno a 4MHz, e gestendo in modo opportuno il timer sopra citato, riesco a scandire mezzo secondo per ogni interruzione, più che sufficiente ad avere una temporizzazione base adeguata, cuore della tempistica del progetto.

Da questa ogni secondo aggiorno i contatori interni e lo stato del Relè, mentre per il resto del tempo il microcontrollore continua a testare i Jumpers relativi alle ore, minuti e secondi sia per la pausa che per l'impulso. Appena il micro rileva un jumper chiuso utilizza i dati provenienti dal convertitore analogico/digitale per salvare lo stato dei Trimmers.

Ho fatto in modo che il firmware riconosca un solo jumper chiuso alla volta, sia per la pausa che per l'impulso, così da evitare ambiguità nella programmazione dei tempi. Quelli relativi invece al modo Continuo(loop) e Impulso/Pausa sono letti solo all'accensione, come descritto nel progetto originale. Sempre all'accensione, viene caricato anche il contenuto della eeprom, la quale ci fornisce l'ultimo settaggio effettuato.

Nella figura in basso è possibile osservare lo schema a blocchi principale del programma:

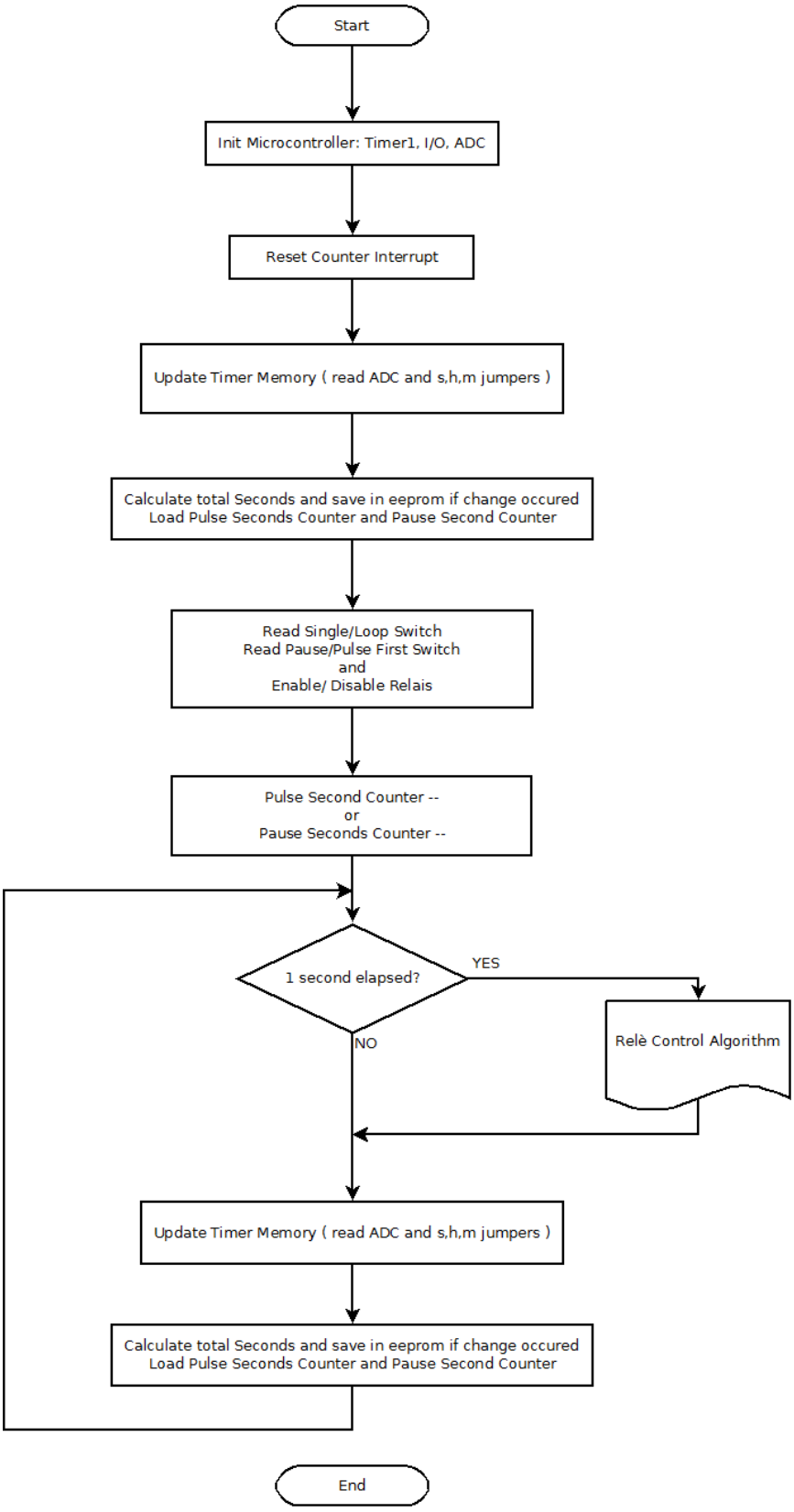

Figura 1 - Flow chart principale

Come si può vedere, l'aggiornamento dello stato del relè avviene ogni secondo, tutto questo dentro a "Relè Control Algorithm", i cui dettagli sono visibili nello schema successivo.

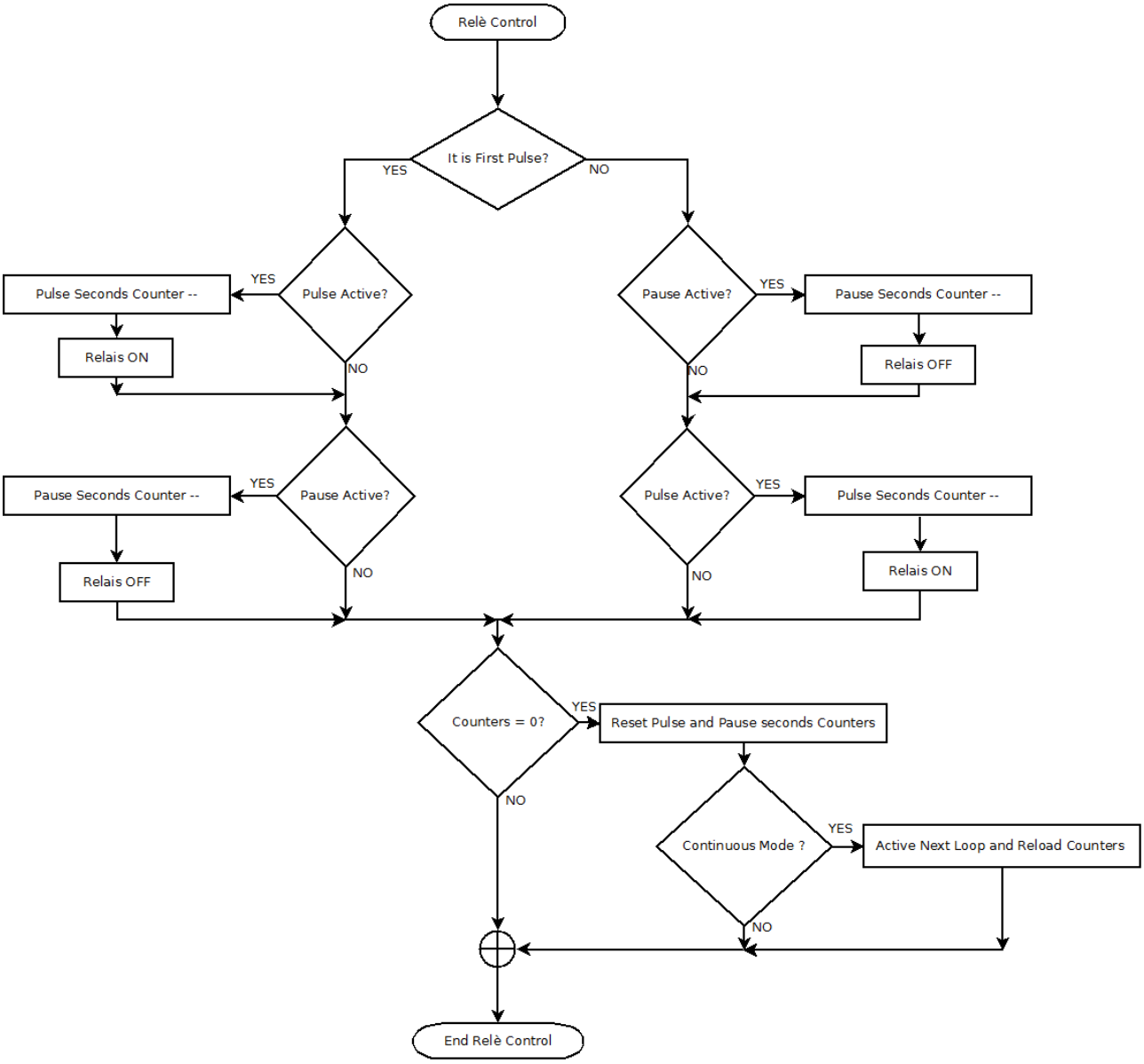

Figura 2 - Flow Chart " Relè Control Algorithm"

Durante questo controllo, il PIC calcola anche il tempo rimanente da scandire, sia per la pausa che per l'impulso.

Come già accennato, all'accensione vengono caricati i valori contenuti in eeprom, i jumpers Loop/Single, Pulse/ Pause e lo stato del rele. Successivamente il Timer rimane nel suo ciclo infinito nel quale continua ad aggiornare il suo stato, e qualora necessario attivare e disattivare l'uscita.

All'accensione del dispositivo avvengono le inizializzazioni base delle periferiche, I/O, ecc.. e successivamente la lettura della eeprom interna e il caricamento dei contatori principali del Timer.

Il modo continuo e il tipo di inizio vengono caricati solo una volta, come nel progetto originale, e questo avviene con le seguenti righe di codice:

```
...
   if (FIRST_PULSE_PIN == SW_CLOSED)
     {
      Timer_Status.Mode = FIRST_PULSE;
      RELE_ON;
      Pulse_Time_Elapsed--;
     }
   else
     {
      Timer_Status.Mode = FIRST_PAUSE;
      RELE_OFF;
      Pause_Time_Elapsed--;
     }

   if (CONTINUOUS_PIN == SW_CLOSED)
      Timer_Status.Continuos_Flag = TRUE;
   else
      Timer_Status.Continuos_Flag = FALSE;
...
```

Jumpers letti al power-up

Organizzazione Del codice.

Per il funzionamento da Timer, questo Firmware utilizza una struttura principale, composta da:

```
      TIMER_TIME Pulse;
      TIMER_TIME Pause;
      unsigned char Continuos_Flag :1;    // loop mode
      unsigned char Mode :1;              // first pulse
      unsigned char Value_Updated :1;     // for reload counters
      unsigned char Reset_Flags :1;       // for first timer update
```

Struttura Principale

all'interno di questa separo l'impulso dalla pausa con la seguente struttura:

```
typedef struct
{
   unsigned char Seconds :6;            // 0-60
   unsigned char Seconds_Flag :1;
   unsigned char Minutes :6;
   unsigned char Minutes_Flag :1;
   unsigned char Hours :6;
   unsigned char Hours_Flag :1;
} TIMER_TIME;
```

Struttura Temporale

La temporizzazione base è data dall'interruzione dovuta al Timer 1 che cade ogni mezzo secondo, il quale incrementa semplicemente un contatore. Questo verrà poi resettato una volta aggiornato il relè.

```
void interrupt
isr(void)
{
   if (PIR1bits.TMR1IF == 1)   // timer1 overflow interrupt
   {
      TMR1L = 0xdb;
      TMR1H = 0x0b;              // Reload prescaler with 62500
      Global_Counter++;
      PIR1bits.TMR1IF = 0;     // clear the interrupt flag, enable next loop
   }
}
```

Dettagli dell'interruzione

Il calcolo del tempo trascorso viene effettuato conteggiando i secondi totali dati dalle ore minuti ed ovviamente i secondi sia per l'impulso che per la pausa, grazie alla funzione "Calc_Seconds()".

I due contatori generali vengono decrementati ad ogni secondo, attivando o disattivando il relè nel momento programmato, questi occupano 3 bytes ciascuno e sono:

```
unsigned short long   Pulse_Time_Elapsed;
unsigned short long   Pause_Time_Elapsed;
```

Contatori dei secondi relativi all'impulso e alla pausa.

Salvataggio nella EEPROM Interna.

Riguardo al salvataggio in eeprom, dato che ha un numero di scritture "limitate", eseguo l'operazione solo la prima volta in cui uno dei jumpers h,m,s risulta chiuso, sia per l'impulso che per la pausa. Se uno dei due non viene toccato, di default rimane 1 secondo.

```
...
 if(EE_Update == TRUE)   // Save only if change occured
   {
     eeprom_write((START_EEPROM_ADDR+i),Timer_Status.Timer_Byte[i] );   // Set Update flags
     __delay_us(1);                                                     // Delay after byte saving
   }
...
```

Dettagli del Salvataggio in EEprom

Tale flag denominato "*EE_Update*" è attivato solo quando c è una differenza fra i livelli logici degli input precedenti con quelli attuali.

Collaudo.

La prova su banco, l'ho effettuata utilizzando un oscilloscopio, con il quale ho potuto anche verificare i tempi degli impulsi e delle pause. Scrivendo questo codice ho fatto in modo che alla riaccensione, il relè parta immediatamente con l'impulso o la pausa programmata in modo che i tempi siano rispettati anche all'istante iniziale. La traccia azzurra corrisponde alla tensione di alimentazione, mentre la gialla alla tensione sul Led.

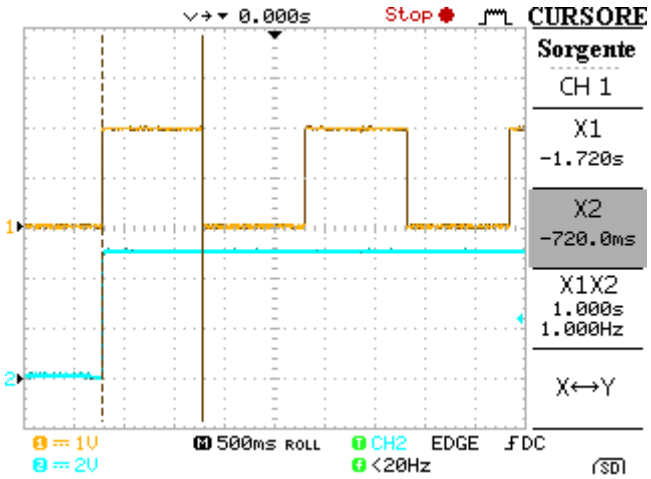

Figura 3 - Accensione del dispositivo in modalità First Pulse, Continuos Mode.

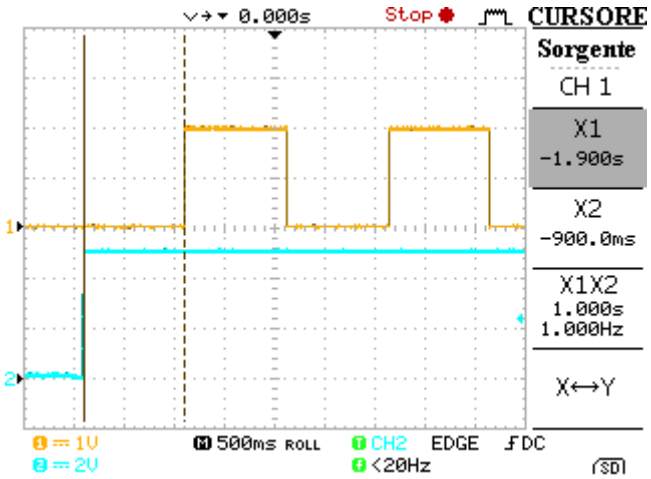

Figura 4 - Accensione del dispositivo in modalità First Pause, Continuos Mode.

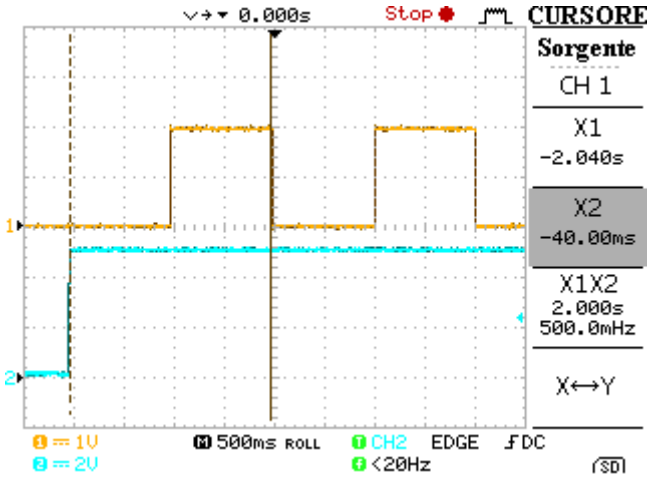

Figura 5 - Misura del periodo completo in modo continuo con Pause = Pulse = 1 secondo.

Di seguito un test di programmazione del tempo di pulse iniziale pari a 5 secondi con pausa 1 secondo, sempre nel modo continuo.

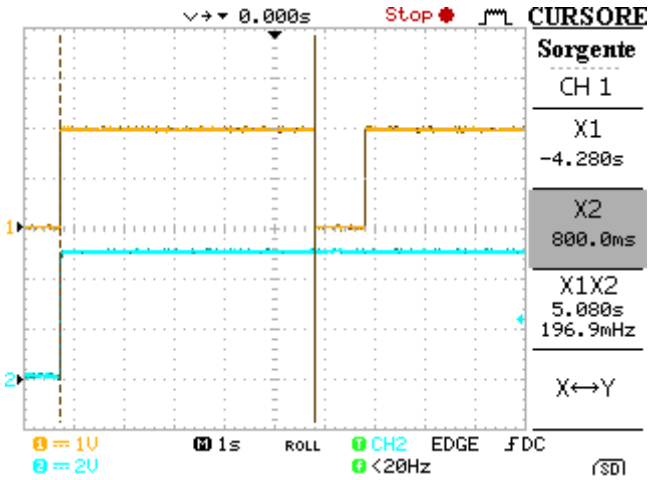

Figura 6 - Pulse = 5 sec. Pause = 1 sec. Continuos Mode.

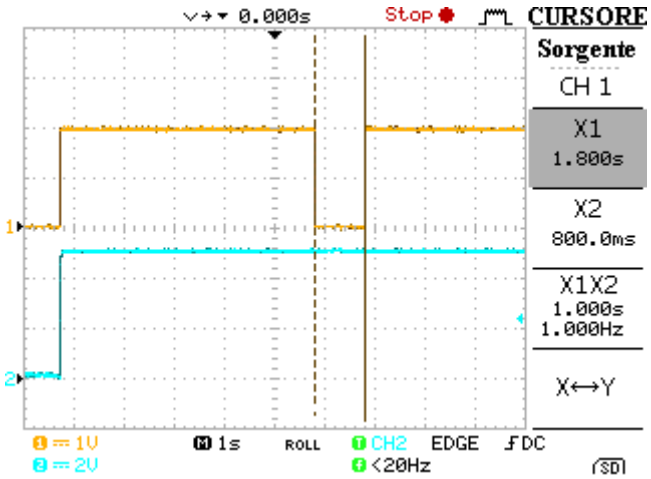

Figura 7 - Misura del tempo di pausa del caso precedente.

Concludo i Test provando la modalità "Single Mode", con i tempi settati entrambi a 1 secondo.
Di seguito potete osservare le acquisizioni:

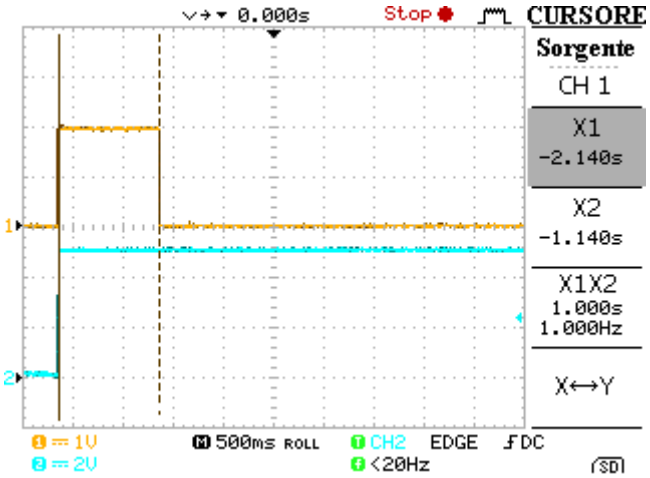

Figura 8 - Riaccensione del dispositivo in modalità First Pulse, Single Mode.

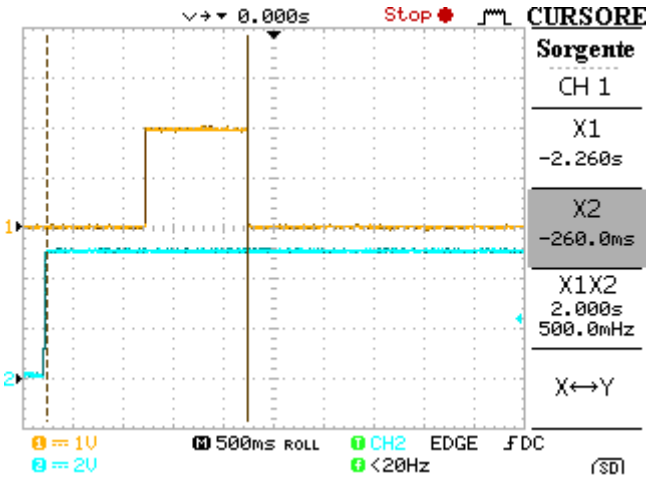

Figura 9 - Riaccensione del dispositivo in modalità First Pause, Single Mode.

Conclusioni.

Il codice è ricco di commenti, per abitudine scritti in inglese (mi scuso quindi in anticipo per eventuali errori).
Credo che sia tutto abbastanza chiaro e leggibile, e spero che questo possa essere utile a coloro che si apassionano di programmazione e ai fantastici microcontrollori PIC di casa Microchip.
Tutti i Software citati per questo progetto sono proprietari di Microchip, disponibili per il download su www.microchip.com.
Ringraziandovi per la lettura, per commenti e consigli rimango a disposizione, buona programmazione a tutti.

Enzo Carollo