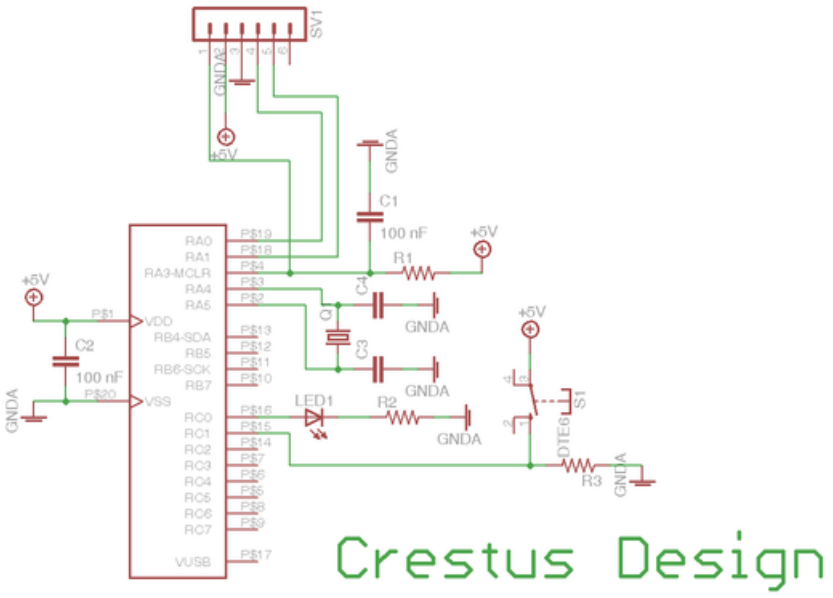
crestus

TUTORIAL PIC - HELLO WORLD

16 December 2010

Hello World - Schema elettrico

Questo è lo schema elettrico per poter provare i 2 programmi che vedremo successivamente.



Schema Elettrico by Crestus

I collegamenti sono stati pensati per testarne il funzionamento finalizzato ad un progetto più grande, ma in realtà le porte sono quasi indifferenti (quasi perché alcune caratteristiche si trovano solo su alcune porte, ma ne parleremo a tempo debito). Questo non è però vero, ad esempio, per i collegamenti dell'ICSP.

PIC e ICSP

Il **connettore ICSP** è stato disegnato al di sopra del micro e serve per collegare il **programmatore PICKIT2**; non è altro che una striscia di pin. Il connettore ha 6 collegamenti, ma noi ne utilizzeremo soltanto 5! sono MCLR, V+, GND, PGM, PGD:

- PGM e PGD sono i veri e propri collegamenti attraverso cui passano le informazioni: i canali attraverso cui sono inviati i bit che devono essere memorizzati nella memoria dedicata al programma.
- V+ e GND sono i pin per l'alimentazione servita dal programmatore: il PIC deve essere alimentato per poter essere programmato e il PICKIT gli può fornire l'energia necessaria.
- MCLR è il pin che di solito dà più problemi. Questo PIN ha un duplice comportamento: durante il regolare funzionamento permette di introdurre nello schema un pulsante di reset che sostanzialmente fa ripartire da capo il programma; è funzionale alla programmazione perché bisogna fornirgli una tensione elevata per abilitare la scrittura sulla memoria di programma. Per fare questo il pin deve ricevere circa 13 Volt. Quello che si fa è quindi di bloccare in maniera sicura la tensione in ingresso a questo pin: non basta collegarlo all'alimentazione, ma bisogna anche fare in modo di filtrare gli sbalzi di tensione che potrebbero portare ad un RESET indesiderato, oltre a limitare la corrente in ingresso. Ecco quindi che viene fuori questo schema: il pin MCLR è collegato in parallelo con una resistenza (circa $10\,\mathrm{k\Omega}$) a V+ e un condensatore (il 100 nF va benissimo) a massa secondo suggerimento dello stesso datasheet. Ovviamente questo pin non può essere utilizzato dal programmatore, non in questa configurazione: ecco che il connettore ICSP è collegato in parallelo a questa rete di filtraggio, comunicando direttamente con il PIC.

Alimentazione

A sinistra del PIC ci sono i suoi pin di alimentazione tra cui c'è un condensatore. Questo è un 100 nF ceramico che serve a eliminare eventuali sbalzi indesiderati della tensione di alimentazione; va montato il più vicino possibile ai piedini del PIC.

Blocchi funzionali

Al PIC sono collegati dei componenti che si possono dividere in 3 blocchi per la loro funzione: Quarzo, Led e Pulsante.

- Quarzo: Questo è un componente fondamentale! É vero che i microcontrollori sono forniti di un oscillatore interno, ma questi sono molto poco precisi, e non sono adatti a programmi il cui funzionamento ha tempistiche precise! Il quarzo dovrà essere collocato il più vicino possibile al PIC e con i suoi condensatori anch'essi molto vicini. I condensatori variano a seconda della frequenza del quarzo! Il valore è attorno ai 15 pF (pico Farad) ceramici: di più per un quarzo più lento, di meno per un quarzo più veloce. Quindi, scelta la frequenza operativa, andate sul datasheet del PIC per trovare maggiori informazioni!
- Led: Il led è collegato direttamente al PIC! Per fortuna la Microchip ha dimensionato le uscite per fornire un massimo di 20 mA di corrente. Noi ne utilizzeremo di meno, ma comunque in qualche maniera bisognerà limitare la corrente no? ecco il perchè della resistenza! Dovete pensare che la porta è logica: fornisce una tensione alta o bassa, ma non fa nessun controllo in corrente. Quando andremo ad accendere il led porteremo il segnale della porta alla tensione di alimentazione e questa tensione sarà sufficiente ad alimentare il Led; quando invece lo spegniamo, portiamo l'uscita alla tensione di massa, allo stato logico 0. L'assenza di differenza di potenziale ovviamente non permette il passaggio di corrente e il led resterà spento. Il calcolo della resistenza si fa nella solita maniera di cui troverete molta documentazione anche qui su ElectroYou
- Pulsante: La configurazione del pulsante può essere molto varia a causa del loro comportamento e delle tecniche che possono venire adottate per risolverlo; per il momento analizziamo il comportamento di un pulsante ideale. Quando il pulsante è aperto la tensione che il PIC riceve è quella della massa (livello logico 0) attraverso una resistenza di pull-down. Quando andiamo a chiudere il contatto, all'altro capo della resistenza viene collegata la tensione di alimentazione, ma la stessa tensione viene ricevuta in input dal pin che la interpreta come livello logico alto (1).La resistenza, oltre che connettere e allo stesso tempo dividere le 2 tensioni di alimentazione, permette di limitare la corrente che fluisce nel PIN. Per limitare la corrente in ingresso quando il pulsante è premuto, andrebbe collegata anche una resistenza tra il pulsante e il PIN.

Il Primo Programma

Forse l'avrete già visto mille volte e in tutte le salse, ma per iniziare non c'è niente di meglio. il programma è molto molto semplice in realtà:

```
#include <p18f14k50.h>
#include <Delays.h>

#pragma config FOSC = HS //Alta frequenza ( 12MHz )
#pragma config WDTEN = OFF //Disabilito WatchDog Timer
#pragma config LVP = OFF //Disabilito Low Voltage Programming

#define Led LATCbits.LATC0

void main (void) {

    LATA = 0x00; // Imposto PORTA tutti ingressi
    TRISA = 0xFF; // Imposto PORTA tutti ingressi

    LATB=0x00; // Imposto PORTB tutti ingressi
    TRISB=0xFF; // Imposto PORTB tutti ingressi

    LATC=0x00; // Imposto C0 uscita altri tutti ingressi
    TRISC=0b11111110; // Imposto C0 uscita altri tutti ingressi

    Led = 1; //LED acceso

    while (1) {
        Delay10KTCYx(150); //Delay di 0,5s
        Led = ~Led; // cambio lo stato del LED
    } //end while

} // end main
```

Se avete letto le guide precedenti dovreste riuscire a “vedere” il programma... Se è così dovreste notare a colpo d’occhio che la sola cosa che fa è accendere e spegnere un led sulla porta RC0, ogni mezzo secondo circa.

Il PIC

Pin Diagrams

20-pin PDIP, SSOP, SOIC (300 MIL)

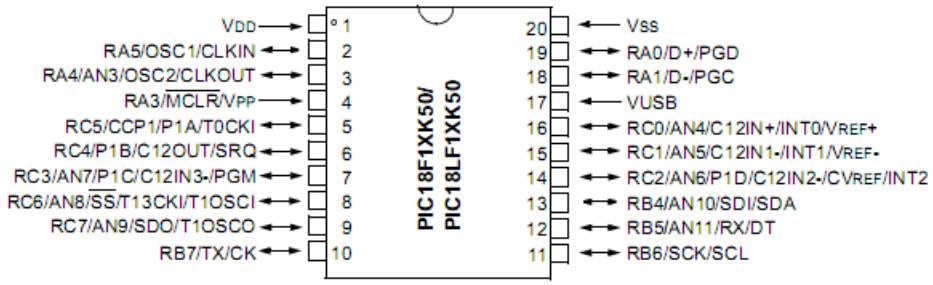


TABLE 1: PIC18F1XK50/PIC18LF1XK50 PIN SUMMARY

Pin	I/O	Analog	Comparator	Reference	ECCP	EUSART	MSSP	Timers	Interrupts	Pull-up	USB	Basic
19	RA0								IOC		D+	PGD
18	RA1								IOC		D-	PGC
4	RA3(1)								IOC	Y		MCLR/VPP
3	RA4	AN3							IOC	Y		OSC2/CLKOUT
2	RA5								IOC	Y		OSC1/CLKIN
13	RB4	AN10					SDI/SDA		IOC	Y		
12	RB5	AN11				RX/DT			IOC	Y		
11	RB6						SCL/SCK		IOC	Y		
10	RB7					TX/CK			IOC	Y		
16	RC0	AN4	C12IN+	VREF+					INT0			
15	RC1	AN5	C12IN1-	VREF-					INT1			
14	RC2	AN6	C12IN2-	CVREF	P1D				INT2			
7	RC3	AN7	C12IN3-		P1C							PGM
6	RC4		C12OUT		P1B							SRQ
5	RC5				CCP1/P1A			T0CKI				
8	RC6	AN8					SS	T13CKI/T1OSCI				
9	RC7	AN9					SDO	T1OSCO				
17											VUSB	
1												VDD
20												VSS

Note 1: Input only

Layout del PIC 18F14K50

Il nome del PIC è PIC18F14K50. Con questo diagramma potrete trovare facilmente tutte le porte e tutte le features di cui dispone il PIC contemporaneamente a dove

sono collocate, vi assicuro che è molto utile, soprattutto in fase di progettazione… Perchè non vi porto un esempio per l'onnipresente PIC16F84? Perchè è obsoleto, costoso e dalle ridottissime potenzialità. La serie 18F, al contrario, possiede molte caratteristiche avanzate e grandi capacità di calcolo ad un prezzo inferiore… inoltre permettono di programmare nativamente in ANSI-C.

Le direttive

Iniziamo ad analizzare il programma per capirne le parti principali, il loro significato specifico e la loro utilità. Le direttive

```
#include <p18f14k50.h>
#include <Delays.h>
```

Queste 2 righe indicano al compilatore dove trovare le istruzioni che corrispondono alle funzioni che saranno utilizzate più in basso nel programma vero e proprio. In questo caso indicano, rispettivamente:

- i nomi dei vari registri di controllo, a che indirizzo di memoria li potrà trovare (su questi registri si basa il funzionamento della famiglia dei PIC);
- cosa vuol dire una funzione "delay" in termini di codice realmente eseguito dal processore.

Queste informazioni sono contenute nei file ".h" disseminati in specifiche cartelle all'interno del PC. Per curiosità potete andarle a vedere (sia i file header sia i file sorgente di queste librerie)… l'importante è che non salviate modifiche! Andando avanti nel programma vanno definiti dei parametri di funzionamento fondamentali per il microprocessore. Queste le indico tramite le istruzioni:

```
#pragma config FOSC = HS //Alta frequenza ( 12MHz )
#pragma config WDTEN = OFF //Disabilito WatchDog Timer
#pragma config LVP = OFF //Disabilito Low Voltage Programming
```

Queste istruzioni sono fondamentali perchè preparano il PIC a lavorare nella maniera corretta con l'HW e il SW fornitogli. Notate che le istruzioni utilizzate sono le istruzioni "#pragma" di cui vi ho accennato nella lezione precedente.

I parametri che troverete sempre sono i primi 2, che sono le configurazioni indispensabili per qualsiasi PIC; ma in generale le configurazioni possibili variano da un microcontrollore ad un altro in base alle caratteristiche specifiche di ogni controllore. I parametri configurabili per ogni PIC li potete trovare in un file di help denominato: “hlpPIC18ConfigSet.chm” sotto la cartella: “MCC18/doc/”. Potete anche scegliere di configurare queste impostazioni tramite una tabella di configurazione del compilatore, ma sorge poi un problema di portabilità con altri PIC e di immediata comprensione. Se vengono scritti sul file sorgente è immediato vedere come viene impostato il micro.

- Nel caso specifico andiamo a settare il PIC per lavorare con un quarzo a 12MHz, con la prima istruzione, che va a configurare il modulo per la sorgente di clock esterna in maniera adeguata;
- nella seconda riga vado a disabilitare il WatchDog timer che permette di evitare impuntamenti del programma, lo disabilitiamo perché stiamo cercando di imparare, e a volte bisogna vedere bene come si blocca il programma per poter risolvere il problema;
- infine, nell’ultima riga, disattiviamo la caratteristica di programmazione a basse tensioni perchè non ci è utile, non con il PICKit2 almeno.

Dopodiché ci viene utile definire, per facilità di stesura/lettura del programma, che tutte le volte che scrivo “LED” lo sostituisca con una serie di altri caratteri; in particolare sarà l’istruzione che indica il pin a cui sarà collegato. L’utilità sta nello scrivere in maniera più “funzionale” senza dover scrivere tutte le volte un anonimo indirizzo di periferica, ed evitare di dover modificare tutte le istanze in caso di modifiche: si modifica la definizione e sono tutte modificate.

```
#define Led LATCbits.LATC0
```

Come vedete il led è stato posizionato sulla porta C0: Pin 16. Il motivo? E’ derivato dal progetto che sto sviluppando. Voi potete sceglierne un altro, è indifferente… o quasi…ma lo vedremo prossimamente!

Il Programma

Il programma vero e proprio è veramente banale e se per voi non lo è ancora, lo diventerà presto:

```
Led = 1; //LED acceso

while (1) {
    Delay10KTCYx(150); //Delay di 0,5s
    Led = ~Led; // cambio lo stato del LED
} //end while
```

Possiamo tradurre queste righe con questa sequenza di concetti: faccio iniziare il programma con il led acceso e dopodiché inizia un loop infinito. All'interno del Loop mi giro i pollici fino a che non è passato mezzo secondo, dopodiché cambio lo stato del led. Infine ricomincio da capo a contare. Se colleghiamo alla corrente elettrica il led lampeggerà tranquillo e sereno.

Un paio di approfondimenti: Un programma è un'entità lineare. Vuol dire che avrà un inizio ed una fine; ciò signfica che quando avrà compiuto le operazioni per cui è stato programmato, il programma terminerà e si chiuderà. Quello che vogliamo fare noi,invece, è tenere acceso il programma all'infinito: dobbiamo quindi ingannare la programmazione al fine di ottenere quello che vogliamo. La tecnica universalmente utilizzata è quella di utilizzare un LOOP con una condizione fasulla:

```
while (1) {

...

}
```

La condizione "1" è sempre verificata, perchè in realtà non è una condizione: risultato avverrà sempre che il ciclo verrà ripetuto, all'infinito. Ma che fine fa la fine del programma? sparisce.. non ci interessa. Potremmo anche scriverla, ma non verrebbe mai eseguita. In realtà per programma complessi che richiedono altissima affidabilità, si scrivono anche istruzioni destinate ad interrompere il programma in caso di errori, ma noi non le affronteremo, non adesso.

Avete anche visto che Led corrisponde alla dichiarazione *LATCbits.LATC0* che corrisponde allo stile adottato da Microchip per la dichiarazione dei registri e in particolare ai singoli BIT dei registri: in particolare significa che stiamo considerando solo il singolo bit del registro LATC che risponde al nome di LATC0. questo particolare registro regola le funzionalità della porta C0 impostata come uscita.

Definito il programma vero e proprio ci rendiamo conto che in realtà c'è qualcosa che viene eseguito prima: bisogna configurare il PIC! bisogna inizializzare le porte, altrimenti il PIC non riuscirebbe a far funziare ciò a cui è collegato. Per questo dobbiamo impostare gli appositi registri in maniera appropriata:

```
LATA = 0x00; // Imposto PORTA tutti ingressi
TRISA = 0xFF; // Imposto PORTA tutti ingressi

LATB=0x00; // Imposto PORTB tutti ingressi
TRISB=0xFF; // Imposto PORTB tutti ingressi

LATC=0x00; // Imposto C0 uscita altri tutti ingress
TRISC=0b11111110; // Imposto C0 uscita altri tutti ingressi
```

come abbiamo detto, ogni bit corrisponde ad un pin. Ma le impostazioni da fare sono 2: dobbiamo impostare la porta per funzionare come ingresso di segnale, o come uscita. Impostando 1 nel relativo bit del registro TRIS settiamo la porta come ingresso, mentre con lo 0 impostiamo un'uscita. Bene, sapendo che il bit a destra è lo LSB (less significant bit cioè il bit dal valore più basso di tutti gli altri), capiamo che la porta RC0 è impostata come uscita, mentre tutte le altre , a salire, sono tutte ingressi. É quindi evidente che possiamo impostare indifferentemente ogni singola porta per conto proprio senza dover impostare tutte le porte di una stessa famiglia (A B C D E ecc.) allo stesso modo come ingresso/uscita.

Il Registro LAT definisce lo stato assunto da una particolare porta: se la porta è a livello logico alto, allora comparirà un 1 sul suo relativo bit; al contrario se è uno 0 logico. I registri LAT vengono impostati a 0 per lo stesso motivo per cui si inizializzano le variabili: per partire da uno stato conosciuto dal momento in cui inizia il programma; e per evidenziare subito eventuali problemi all'accensione o errore nei registri. Forse di questo registro non avete mai sentito parlare nel vostro girovagare su internet alla ricerca di informazioni. E' esattamente equivalente al registro PORT, che è l'unico registro disponibile sulla famosa famiglia 16F (e quindi anche sul famosissimo 16F84)

Il registro LAT sfrutta, però, una circuiteria differente: la ragione sta nel fatto che permette di evitare errori di rilevamento dello stato della porta in certi casi specifici. Per evitare confusioni e incomprensioni vi lascio una "regola d'oro" per la scelta dei registri da utilizzare nelle varie situazioni:

Il registro PORT per gli ingressi, LAT per le uscite

Non ci resta altro che compilare il programma, Caricarlo sul PIC tramite il PICKit2. Colleghiamo la corrente e scopriamo che: **non funziona!**

Dannazione! che succede? eppure il programma è identico carattere per carattere a quello della guida! Beh.. il problema non è vostro! Il programma è giusto, tant'è che non avete avuto problemi a compilarlo, ma non abbiamo tenuto conto di una cosa: Andate a guardare la tabella che vi ho proposto sopra alla riga della nostra porta C0. Alla 3° colonna notate che associata a quel pin c'è un convertitore analogico. "Mamma" Microchip imposta i propri PIC per avere abilitate le porte analogiche di default! Per poter utilizzare la porta C0 e tutte le altre con funzionalità analogiche, dobbiamo prima configurarle per funzionare come porte digitali!

Note:	On a Power-on Reset, RA4 is configured as analog inputs by default and read as '0'; RA<1:0> and RA<5:3> are configured as digital inputs.

riquadro evidenziato

Per far questo andiamo a controllare il datasheet e troviamo in un riquadrino evidenziato nei capitoli delle porte (capitolo: "I/O Ports") l'indicazione di quali porte sono analogiche. Il riferimento ad un registro "ANSEL" lo trovate successivamente. Se andiamo a vedere questo registro scopriamo che è proprio il registro adibito alla selezione della modalità di funzionamento di queste porte.

9.4 Port Analog Control

Some port pins are multiplexed with analog functions such as the Analog-to-Digital Converter and comparators. When these I/O pins are to be used as analog inputs it is necessary to disable the digital input buffer to avoid excessive current caused by improper biasing of the digital input. Individual control of the digital input buffers on pins which share analog functions is provided by the ANSEL and ANSELH registers. Setting an ANSx bit high will disable the associated digital input buffer and cause all reads of that pin to return '0' while allowing analog functions of that pin to operate correctly.

The state of the ANSx bits has no affect on digital output functions. A pin with the associated TRISx bit clear and ANSx bit set will still operate as a digital output but the Input mode will be analog.

REGISTER 9-15: ANSEL: ANALOG SELECT REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	U-0	U-0	U-0
ANS7	ANS6	ANS5	ANS4	ANS3	—	—	—
bit 7							bit 0

Registro ANSEL

In realtà i registri sono 2, ma il principio di funzionamento è identico: se posto a zero il bit, la porta associata verrà impostata come digitale. Come regola di massima io disabilito sempre le porte analogiche all'inizio del programma! Se, ad un certo punto del programma avrò bisogno di una porta analogica allora la riattiverò. Questo vuol dire che imposterò tutti i bit a zero scrivendo queste 2 righe di codice:

```
ANSEL = 0x00;
ANSELH = 0x00;
```

Il programma corretto sarà quindi (cliccare per aprire):

```
#include <p18f14k50.h>
#include <Delays.h>

#pragma config FOSC = HS //Alta frequenza ( 12MHz )
#pragma config WDTEN = OFF //Disabilito WatchDog Timer
#pragma config LVP = OFF //Disabilito Low Voltage Programming

#define Led LATCbits.LATC0

void main (void) {

ANSEL = 0x00;
ANSELH = 0x00;

LATA = 0x00; // Imposto PORTA tutti ingressi
TRISA = 0xFF; // Imposto PORTA tutti ingressi

LATB=0x00; // Imposto PORTB tutti ingressi
TRISB=0xFF; // Imposto PORTB tutti ingressi

LATC=0x00; // Imposto C0 uscita altri tutti ingressi
TRISC=0b11111110; // Imposto C0 uscita altri tutti ingressi

Led = 1; //LED acceso
```

```
while (1) {
Delay10KTCYx(150); //Delay di 0,5s
Led = ~Led; // cambio lo stato del LED
} //end while

} // end main
```

Gestione di un Pulsante

Dopo aver visto come realizzare il nostro primo progettino con un pic della serie 18F, possiamo iniziare ad aumentare la difficoltà.

Il Passo successivo è infatti gestire un pulsante per dare istruzioni al nostro PIC. Ci sono alcune cose da sapere sul comportamento dei pulsanti: non sono perfetti! Questa affermazione si traduce nel fatto che al momento della chiusura del contatto il segnale che viene inviato non è istantaneo e nemmeno preciso e pulito. Il PIC rileva il cambio di tensione applicata ala porta a cui è connesso il pulsante, ma questa tensione avrà delle oscillazioni per svariati ms (millesimi di secondo). Per i nostri sensi sono inezie, ma per il pic che compie come minimo 4 milioni di istruzioni al secondo ogni spike (picco di tensione di breve durata...il disturbo in pratica) potrebbe essere interpretato come una pressione del pulsante. Quello che viene normalmente fatto è “filtrare” questo periodo istruendo il microcontrollore su quanto tempo dovrà passare ignorando i cambiamenti sulla porta dopo il primo segnale. Passiamo direttamente al codice!

```
#include <p18f14k50.h>
#include <Delays.h>

#pragma config FOSC = HS //Alta frequenza ( 12MHz )
#pragma config WDTEN = OFF //Disabilito WatchDog Timer
#pragma config LVP = OFF //Disabilito Low Voltage Programming

#define Led LATCbits.LATC0
#define Button LATCbits.LATC1
#define T1 0x96 //150
#define T2 70

char tempo=T1;

void main (void) {
ANSEL = 0x00; // elimino ingressi analogici 1
```

```
ANSELH = 0x00; // elimino ingressi analogici 2

LATA = 0x00; // Imposto PORTA tutti ingressi
TRISA = 0xFF; // Imposto PORTA tutti ingressi

LATB=0x00; // Imposto PORTB tutti ingressi
TRISB=0xFF; // Imposto PORTB tutti ingressi

LATC=0b00000000; // Imposto C0 uscita altri tutti ingressi
TRISC=0b11111110; // Imposto C0 uscita altri tutti ingressi

Led = 1;

while (1) {
if (Button==1) {  // Se il pulsante è premuto

Delay10KTCYx(30);  // FILTRAGGIO SPIKES

if (Button==1)  // è ancora premuto?
tempo = (tempo == T1) ? T2 : T1;  // cambio frequenza di lampeggio

}  //End if

Delay10KTCYx(tempo);
Led = ~Led;

} //end while

} // end main
```

Bene… Rispetto allo scorso programma possiamo notare alcune introduzioni: intanto Notiamo la riga info

```
#define Button LATCbits.LATC1
```

Questo ci dice che al nostro PIC sarà collegato un pulsante sulla posta C1. Inoltre notiamo le righe

```
#define T1 0x96 //150
#define T2 70
```

Queste righe mi definiscono 2 nomi a cui sono associati dei valori: quando il compilatore troverà questi nomi li sostituirà con i valori che gli abbiamo attribuito. questo è un metodo per poter utilizzare la stessa costante all'interno del programma utilizzando un nome che ci indica la sua funzione. inoltre, in caso di bisogno, cambiando il valore della definizione cambieremo in blocco tutte le istanze, evitando di dimenticare di cambiarne qualcuna. In questo caso possiamo vedere che le variabili si possono definire in diversi modi: T1 è definito come esadecimale, T2 come decimale. il risultato non cambia. Le righe:

```
ANSEL = 0x00; // elimino ingressi analogici 1
ANSELH = 0x00; // elimino ingressi analogici 2
```

sono fondamentali. Come scritto nel precedente articolo servono a configurare in digitale le porte con funzionalità analogiche! La configurazione dei registri delle porte non è stata modificata dallo scorso programma:

```
LATA = 0x00; // Imposto PORTA tutti ingressi
TRISA = 0xFF; // Imposto PORTA tutti ingressi

LATB=0x00; // Imposto PORTB tutti ingressi
TRISB=0xFF; // Imposto PORTB tutti ingressi

LATC=0b00000000; // Imposto C0 uscita altri tutti ingressi
TRISC=0b11111110; // Imposto C0 uscita altri tutti ingressi
```

Il registro TRISC configura sempre la porta C0 come uscita per pilotare il led, e la porta C1 come ingresso per poter leggere il pulsante.

Il Programma vero e proprio

Per capire meglio il programma è meglio utilizzare il diagramma di flusso:

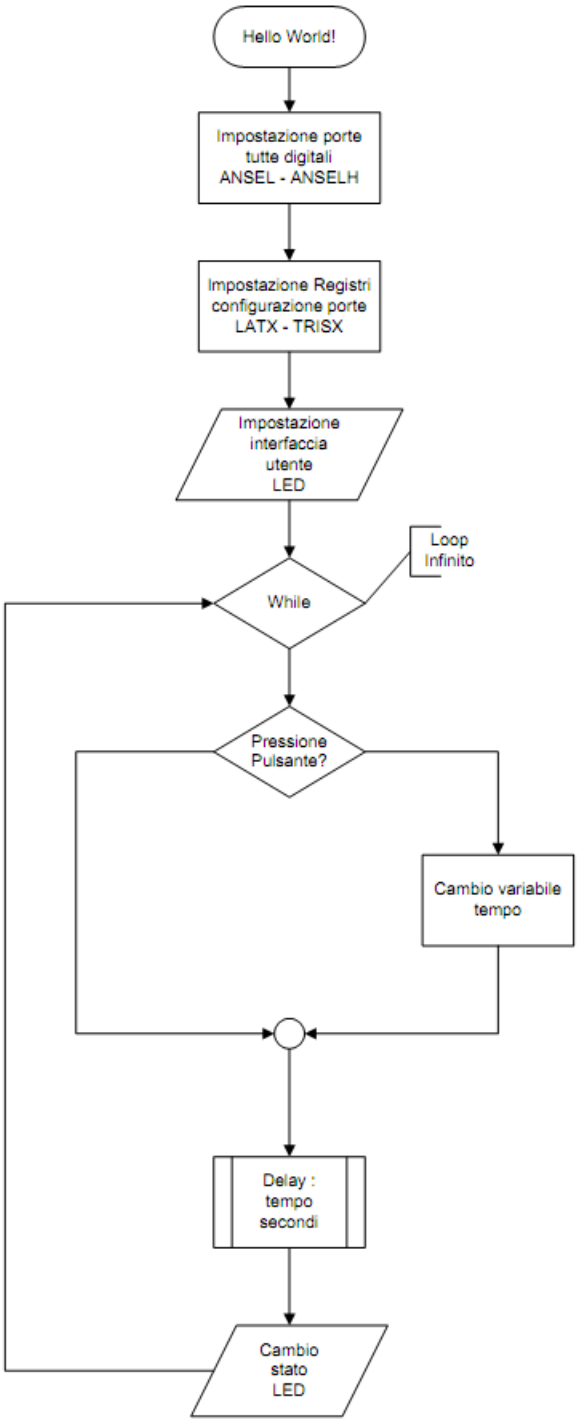


Diagramma di Flusso

Come potete vedere, entrando nel Loop Infinito passo il tempo a controllare se il pulsante viene premuto. In queso caso vado a modificare la variabile che controlla il tempo tra un cambio di stato del led e l'altro. L'ultima operazione che viene fatta

è il delay, dopodichè ricomincio tutto da capo all'infinito. Non è però finita quì. Se ci limitassimo a controllare la tensione presente sul pin potremmo incappare in vari errori. Come vi dicevo all'inizio di questo articolo, il pulsante reale si discosta dal funzionamento del pulsante ideale: al contatto la tensione non cambia immediatamente e nemmeno in maniera stabile: ci sono sempre delle fluttuazioni sulla tensione in ingresso al PIC.
Queste possono durare svariati millisecondi in maniera dipendente alla qualità del pulsante. Come risolviamo allora questo problema? Semplice, ignoriamo ciò che avviene sulla porta! Quando il pic rileverà la tensione corretta inizierà a ignorare le informazioni successive per un tempo prestabilito. Solitamente viene utilizzato un tempo di 100 ms (0,1 s) al termine del quale il PIC controllerà lo stato dell'ingresso, prendendo per buono e stabile ciò che trova. i tempi possono anche essere ridotti, se necessario, ma devono essere controllati i risultati facendo delle prove a priori per individuare il limite a cui si può arrivare con i vostri pulsanti. Come calcolare il Delay lo vedremo fra poco. Vediamo quindi il codice relativo:

```
while (1) {
    if (Button==1) {  // Se il pulsante è premuto
        Delay10KTCYx(30);  // FILTRAGGIO SPIKES
        if (Button==1)  // è ancora premuto?
            tempo = (tempo == T1) ? T2 : T1;  // cambio frequenza di lampeggio
     }  //End if

    Delay10KTCYx(tempo);
    Led = ~Led;

} //end while
```

Vedete che immediatamente dopo essere entrati nel loop inizio il ciclo in cui interrogo la porta per controllare se il pulsante è stato premuto. L'istruzione ci da anche un'altra informazione: il pulsante deve dare un livello logico alto (+V)come risultato della pressione: questo vuol dire che normalmente il pin dovrà leggere uno stato logico basso (GND). il collegamento in se lo vedrete nello schema. Questa situazione è anche ribaltabile: posso avere come risultato un livello logico basso, Ovviamente l'istruzione andrà modificata in questa maniera:

```
if (Button==0) {  // Se il pulsante è premuto
```

l'IF poteva anche essere scritta come:

```
if (Button) {  // Se il pulsante è premuto
```

Perchè l'if riconosce come risultato positivo o negativo rispettivamente un 1 o uno 0 fornitogli tramite le condizioni. Ma Button è già un BIT il cui valore è solo 0 o 1: quindi quando il pulsante sarà premuto Button varrà 1 che è già di per se una condizione sufficiente per entrare nel ciclo if. Avrete notato che ho scritto "==" e non "=" come condizione dell'if: questo perchè "==" è l'operatore di confronto per uguaglianza, mentre "=" è l'operazione di uguaglianza di 2 valori. Se avessi scritto

```
if (Button=1) {  // Se il pulsante è premuto
```

ogni volta che il programma passa per questa istruzione assegnerebbe 1 alla variabile Button rendendo così vera la condizione dell'if ed entrando nel ciclo ogni volta e non solo alla pressione del pulsante. Bene; Passiamo ira a vedere cosa succede all'interno dell'if. Subito ci troviamo davanti all'operazione di Delay la quale dura esattamente 100 ms. vi ricorda qualcosa? Esatto! il filtraggio degli spike provenienti dal pulsante. Dopodichè andiamo a controllare nuovamente lo stato della porta C1:

```
if (Button==1)  // è ancora premuto?
tempo = (tempo == T1) ? T2 : T1;  // cambio frequenza di lampeggio
```

Come vedete questa volta non ho usato le parentesi. Questo perchè nella sintassi C vi è una regola, la quale dice che se l'istruzione associata ad una istruzione decisionale è una soltanto, si possono non mettere le parentesi perchè la riga successiva sarà considerata come l'istruzione in questione, ed eseguita all'occorrenza di una condizione positiva. in pratica posso evitare le parentesi se devo scrivere una sola linea di codice, come in questo caso. Ma cos'è esattamente l'istruzione alla linea 36? è una convenzione del C per esprimere il cambio di valore di una variabile tra 2 valori predefiniti. La possiamo interpretare inquesta maniera:

- variabile tempo è uguale a : (tempo è uguale a T1?) se sì allora deve diventare uguale a T2 : Altrimenti uguale a T1;

Abbastanza chiaro?

Istruzioni Delay

Ma come calcoliamo il valore da inserire nell'istruzioe di delay per ottenere il tempo desiderato? Beh, andiamo a leggerci il manuale del compilatore, nel quale troveremo che:

Delay10KTCYx	
Function:	Delay in multiples of 10,000 instruction cycles (TCY).
Include:	`delays.h`
Prototype:	`void Delay10KTCYx( unsigned char unit );`
Arguments:	***unit*** The value of ***unit*** can be any 8-bit value. A value in the range [1,255] will delay (***unit*** * 10000) cycles. A value of 0 causes a delay of 2,560,000 cycles.
Remarks:	This function creates a delay in multiples of 10,000 instruction cycles. This function uses the globally allocated variable, `DelayCounter1`. If this function is used in both interrupt and mainline code, the variable `DelayCounter1` should be saved and restored in the interrupt handler. Refer to the `save=` clause of the `#pragma interrupt` or `#pragma interruptlow` directives for more information. Note that other delay functions also use the globally allocated `DelayCounter1` variable.
File Name:	`d10ktcyx.asm`

Istruzione Delay 10000 cicli

Come leggete, ogni cifra inserita tra le parentesi corrisponde a 10000 istruzioni. Ma come sappiamo il numer di cicli da effettuare? Bisogna fare qualche calcolo!

$$F_{osc} = 12MH_z$$

Tempo di esecuzione 10000 istruzioni:

$$\left(\frac{1}{12 \cdot 10^6}\right) \cdot 4 \cdot 10000 = 0,00\overline{3}$$

100 ms corrispondo quindi a :

$$\frac{0,1\ \ (\mathrm{s})}{0,00\bar{3}} = 30$$

Ora sappiamo che per avere 100 ms dovremo scrivere 30! Partendo da questo risultato possiamo anche sapere agevolmente multipli e sottomultipli.. perchè 500 ms sarà $30 \cdot 5 = 150$. Informazione di base: la frequenza con cui vengono eseguite le istruzioni è diversa dalla frequenza dell'oscillatore: ogni istruzione richiede 4 colpi di clock. Quindi $MIPS = \left(\frac{F_{osc}}{4}\right) = \left(\frac{12 * 10^6}{4}\right) = 3\ MIPS$, cioè 3 milioni di istruzioni al secondo.

Video e Download

Flash

e quì potete scaricare il pacchetto zippato con i sorgenti e i progetti per questi 2 programmi..:)

Download

FONTI

Gli argomenti trattati in questa guida sono semplificati e adatti a lettori con una conoscenza di base. Questa comprende la conoscenza del linguaggio di programmazione C e la comprensione di base del funzionamento del microprocessore.

Inoltre è indispensabile andare a recuperare le informazioni necessarie sul DataSheet del proprio modello di microprocessore: sono tutti disponibili direttamente e gratuitamente sul sito del produttore: Microchip.

Questa guida è concepita come una versione semplificata e un aiuto passo-passo per i primi tentativi con la programmazione, l'argomento è trattato più approfonditamente nel testo "C18 Step by step" gratuito di Marco Laurenti disponibile sul sito laurtec.it. Altro testo assolutamente valido, ma a pagamento, è quello di Paolo Rognoni : "Pillole di microcontrollori PIC"che potete trovare a questo indirizzo. Trovate informazioni a riguardo sul sito : PicExperience. Potete inoltre trovare alcuni stralci del libro pubblicati dall'autore sul sito **ElectroYou** più precisamente all'indirizzo : PICcoli grandi PICMicro.

Lezione Precedente

Il mio sito : **Dreamasearcher**