Let's go PIC !

Digital I/O

Parte III: Nuovo corso di programmazione C16 per PIC basato sul sistema di sviluppo Micro-GT PIC versatile IDE

Nell'immagine una bella realizzazione della Micro-GT PIC versatile I.D.E. sviluppata da Luca, affezionato utente di Grix che si e' procurato un esemplare del PCB

Questa terza parte del nostro tutorial dovrebbe mettervi rapidamente in grado di leggere o pilotare ingressi o uscite digitali dei microcontrollori. Benché il corso sia specifico per il sistema di sviluppo Micro-GT PIC versatile IDE, potrà essere utilizzato anche da chi non possiede questa scheda, ma dovrà arrangiarsi nello sviluppare una opportuna demoboard. Dato che a magazzino dispongo ancora di alcuni esemplari della campionatura che ho fatto costruire in Cina, chi volesse averne una mi contatti che gliela fornirò a prezzo di costo. Gli schemi elettrici sono disponibili anche in formato PDF scaricabile, ma vi sconsiglio di tentare l'avventura di farla su mille fori. E' troppo ampia e piena di connessioni passanti tra i due layer.

Microcontrollore di riferimento

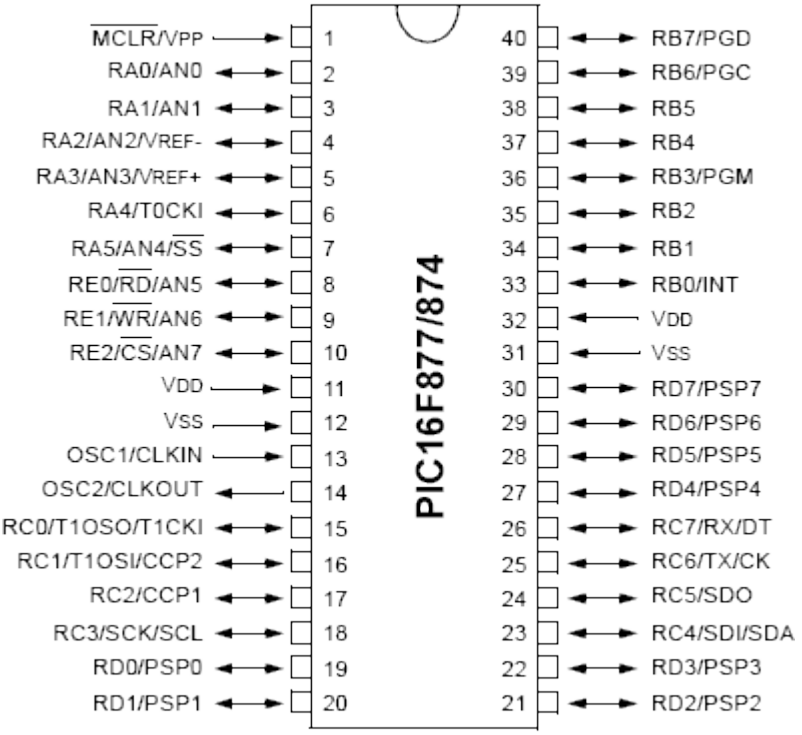
La famiglia di microcontrollori PIC è molto ampia e a sua volta suddivisa in sottofamiglie delle quali si indica con un numero la serie costruttiva, ad esempio 12, 16, 18, 32 ecc.

A seguito di questa prima cifra numerica c'è solitamente una lettera, ad esempio "C" o "F" la prima indicante la tecnologia CMOS e la seconda, di nostro diretto interesse, dato che operiamo in ambito di studio e sviluppo di prototipi, che indica la tecnologia "Flach", ovvero che contiene una area di eeprom in cui potremmo scrivere, cancellare, e riscrivere i programmi diverse migliaia di volte.

Alla lettera citata, segue una altro codice numerico che identifica il tipo di PIC, ad esempio 877, che è poi il prodotto di riferimento per i nostri esperimenti. Ovviamente potete usare un qualunque altro PIC riadattando di volta in volta quello che viene fornito negli esempi di programmazione.

Molti PIC altre a questi tre campi sigla, hanno anche una lettera, la "A" che sta ad indicare la revisione costruttiva del medesimo chip allo scopo di applicare le migliorie tecnologiche che si sono sviluppate nel corso degli anni, ecco che ad esempio potrete trovare un PIC16F877 come il PIC16F877A, che a causa delle tensioni necessarie per la programmazione, l'ampiezza delle aree di memoria e la velocità dell'area flasch o semplicemente del clock esterno, posso essere non pienamente supportati da alcuni programmer. Questa cosa dovrebbe essere risolta nei nuovi sistemi di sviluppo che dovrebbero, come nel caso della nostra Micro-GT PIC versatile IDE essere in grado di riconoscere e di flashare i PIC sia con la "A" che senza la "A".

Il PIC di riferimento per il nostro tutorial è il PIC16F877A , lasciando comunque ampia scelta agli utenti, in funzione di cosa trovano nei cassetti del proprio laboratorio.



Questo PIC per il momento rappresenta il miglior compromesso performance/costo nonché semplicità di utilizzo/difficoltà di attrezzassi per usarlo.

Bus interni

Con il termine "Bus" si vuole intendere un insieme di linee parallele di numero dipendente dall'applicazione, atte al collegamento interno, ma in qualche caso anche esteso all'esterno del processore tramite appositi connettori e cavi piatti, in grado di collegare punti diversi, entrambi interni o misti interni/esterni allo scopo di trasferire parallelamente dei dati.

Solitamente un bus (si legge bas) trova origine o terminazione in "registri", ovvero aree di memoria temporanea per un singolo dato di estensione variabile.

In presenza di uno schema a blocchi dell'architettura interna, i bus sono distinguibili in quanto disegnati

con linea simile a fascia larga e vuota, in qualche punto indicata con fascetta trasversale con vicino un numero indicante di quante linee è formato il suddetto bus.

I bus più noti sono:

- il bus dati
- Il bus indirizzi
- il bus di programma
- il bus di accesso diretto ai registri di lavoro.

L'estensione di questi bus corrisponde al numero chiaramente visibile sull'immagine dell'architettura interna.

Registri interni

Il **FLIP/FLOP** è una nota configurazione circuitale in elettronica digitale, in grado di memorizzare una volta ricevuto il comando, un bit, e di trattenerlo fino al sopraggiungere del comando di reset.

I FLIP/FLOP possono essere interconnessi tra di loro formando delle catene, più o meno lunghe assumendo il nome di registri.

I registri interni del PIC possono essere caricati a intera parola, da non confondere con word intesa come 16 bit, dato che in questo caso si intente la massima estensione del registro, oppure a singoli bit.

Quando di scrive su un registro di un pic lo si fa allo scopo di impostare particolari funzionalità, come ad esempio la velocità e il tipo di clock nel così detto registro dei fuse. La possibilità o meno di intervenire al timer di watchdog, nel medesimo registro.

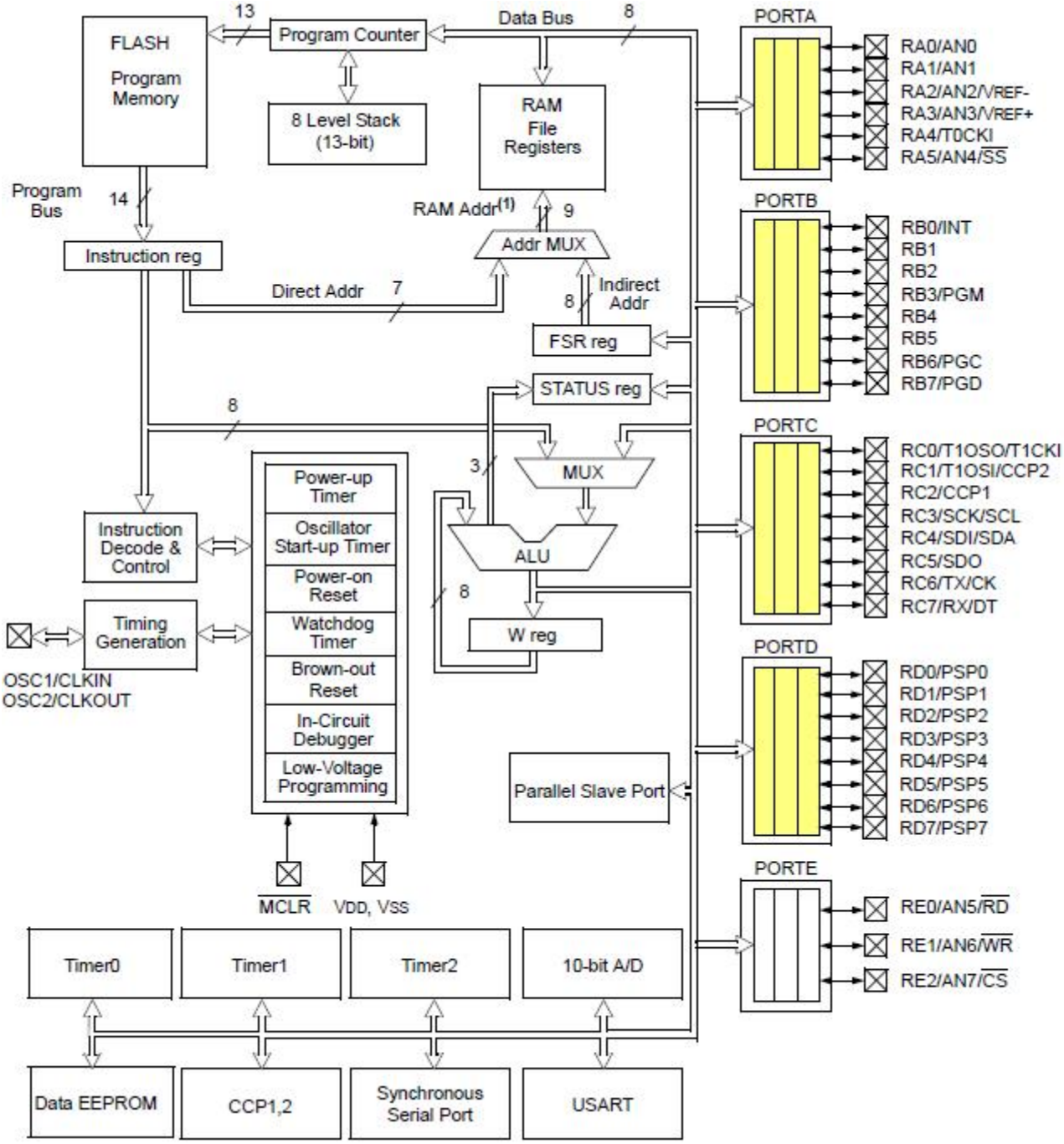
Si impostano dei bit nel registro **ADCON** per abilitare o meno certe funzioni sui convertitori analogici digitali, si setta una configurazione di bit nei registri **TRIS**, come anche si scrive una configurazione nei registri **PORT**.

Quando si programma in Assembly (con assembler si intende il compilatore e non il linguaggio che è appunto l'Assembly), è di fondamentale utilizzo il registro di lavoro, indicato con working register, dove transitano gli operandi prima di entrare nell'unita di processo, la ALU (aritmetic logical unit)...

In "C" il working register ha un ruolo apparentemente secondario dato che è gestito in trasparenza dal codice assemblato dal compilatore.e. E' già noto, a chi ha letto i precedenti capitoli, che l'unico formato accettabile ad essere trasferito nella memoria del PIC è il .hex che otterremo come risultato della compilazione tramite il compilatore online a Mp-Lab, prodotta dalla HITECH.

Concetti Hardware fondamentali

In questa sezione parleremo in maniera semplice e diretta delle connessioni digitali disponibili nel nostro microcontrollore.
I "buffer" (aree di transito o parcheggio dei dati con effetto memoria) dedicati all'**I/O** digitale sono stati indicati dalla casa casa costruttrice Microchip con il nome di "PORT" seguito da una lettera che ne indica il collegamento e quindi l'accessibilità con il bus dati interno.
Sul lato sinistro del disegno dell'architettura interna vediamo evidenziato in giallo i PORT disponibile nel **PIC16F877A**. Come possiamo notare i buffer sono collegati ai quadratini "crocettati" ovvero i pin di connessione verso l'esterno.
La versione che impiegheremo sarà quella con piedinatura **DIL** (dual in line) con riferimento che l'housing del chip ha due file parallele di PIN, 20 ciascuna per un totale di 40, alla distanza standard di un decimo di pollice, quindi circa 2,54mm.
Come possiamo notare ogni pin è connesso al buffer interno tramite una linea bidirezionale. Effettivamente ognuno di questi punti di collegamento potrà essere un ingresso o una uscita. La destinazione finale di funzionamento deve essere dichiarata via software prima dell'utilizzo del punto di collegamento.



il registro delegato alla dichiarazione della direzione del segnale sul pin del PIC viene puntato e parametrizzato con il comando:

```
TRIS"lettera"=parametrizzazione;
il vero comando C16 sarà ad esempio:
TRISB=0;
TRISC=0xFF;
```

E' possibile, solo in prima istanza, usare il promemoria seguente:

```
uscita -> output -> "0" (è uno zero che somiglia alla o di output)
ingresso -> input -> "1" (perché assomiglia alla i di input)
```

In realtà, il comando che riguarda la direzione in "ingresso" è bene approfondirlo con ulteriori spiegazioni, altrimenti si rischia che sviluppare programmi non funzionanti o peggio ancora "pseudo-funzionanti" che accettano un segnale in input in maniera random o molto instabile.
Chiariamo subito che i registri TRIS hanno estensione 8 bit (in questa serie di processori) quindi parametrizzarli con "0" o "1" non è logicamente corretto dato che con una cifra ci si riferisce a un bit e non a un byte.

Cosa succede se parametrizzo a "1" il PORTA tramite il comando TRISA=1; ?

- il comando viene correttamente accettato dal compilatore.
- si parametrizza a 1 solo il punto di I/O RA0 che diviene un ingresso.
- i rimanenti bit da RA1 a RA7 sono instabili o addirittura random come ingresso o uscite.

Il comando coretto è quello che parametrizza tutta l'estensione del byte, come ad esempio i seguenti due che sono equivalenti per il **PORTC** e il **PORTB**;

```
TRISC=0xFF;  //parametrizza come ingresso tutti gli 8 bit del PORTC
TRISB=0b11111111;  //parametrizza come ingresso tutti gli 8 bit del PORTB
```

Il primo è impostato tramite la modalità binaria, quindi un bit alla volta, mentre per il secondo, essendo valida la conversione di base numerica **F=1111**, e ancora **F=1111**, si ha che "x" abilitando la modalità esadecimale, imposta gli otto bit del registro dichiarato prima del segno "=" a **FF->11111111**.
Ho visto molti programmi didattici non funzionare a causa di questa piccola ma grave insicurezza da parte dei programmatori.
Secondo questo metodo di funzionamento è possibile eseguire impostazioni miste dei bit del medesimo byte, quindi ad esempio, il comando:

```
TRISB=0b01010101;
```

ha l'effetto di impostare in maniera alternata gli 8 bit dello stesso PORTB come ingressi e come uscite digitali, a partire dal meno significativo RB0 che è impostato come ingresso.
Osserviamo il bit RC6 e RC7 del nostro PIC16F877A sullo schema a blocchi dell'architettura interna.
Notiamo che questi sono abbinati a una doppia funzione, oltre a essere in grado di fare da ingressi o uscite digitale sono collegati all'UART (universal asincronus receiver transmitter), ovvero a quella parte di hardware delegata alla comunicazione con protocollo seriale, ad esempio un EIA-RS232C. Nel caso impostassimo il PORTC in uscita questa funzione verrebbe a mancare, quindi il comando potrebbe essere ad esempio:

```
TRISB=0b11000000;
```

Si avrà lo stesso effetto impostando la porta in esadecinale, ad esempio convertendo il binario scritto sopra con la calcolatrice di windows:

```
TRISB=0xC0;
```

Ulteriori informazioni sul registro TRIS risulterebbero fuorvianti e quindi preferisco fermarmi qui, almeno fino a che il lettore non avrà ben fissato questi semplici ma fondamentali concetti.

Potenzialità dei PORT
La maniera più immediata per visualizzare le potenzialità dei port è quella di riferirsi allo schema a blocchi dell'architettura interna, ovvero la prima immagine del presente tutorial, nel nostro caso riferita al **16F877**.
Risaltano subito gli indirizzi dei pin, richiamabili in "C" detto anche "ansi C" se non si tratta di particolari rielaborazioni di alcune software house, con il loro nome indicato affianco di ogni quadrettino, esempio **RB0, RC2,RA3**, ecc.
Questo PIC presenta:

```
PORTA -> 6 punti di collegamento polifunzionali, la maggior parte degli
analogici è in questo PORT
PORTB -> 8 punti di collegamento non tutti polifunzionali
PORTC -> 8 punti di collegamento polifunzionali .
PORTD -> 8 punti di collegamento polifunzionali.
PORTE -> 3 punti di collegamento polifunzionali ma che bene riservare al
controllo dell'LCD.
```

Ogni pic è corredato dalla casa costruttrice di un PDF di spiegazione tecnica delle potenzialità, indirizzabilità, uso delle funzioni speciali, ecc.

Tensioni standard
Per come sono progettate le porte di connessione sono compatibili TTL, quindi accettano e propongono verso le periferiche 5V. Per interfacciare il pic al mondo esterno è bene sempre optoisolarsi e collegarsi ai carichi tramite dei BJT o MOSFET di opportuna potenza. I livelli possono essere adattati tramite dei comparatori fatti con amplificatori operazionali. Per il momento è meglio non dire che alcuni pin accettano segnali e tensioni diverse da quelle appena accettate perché il neofita quasi certamente romperà il suo primo pic. Facciamo i primi passi in maniera semplice, sicura e molto standard.

Hardware relativo all' I/O digitale della Micro-GT

Come possiamo notare dall'immagine sottostante, da tutti i pin dello zoccolo textool, chiamato anche ZIF (zero force insertion), sono messi a disposizione per eventuali connessioni da e per il mondo esterno, che chiameremo il bordo macchina.

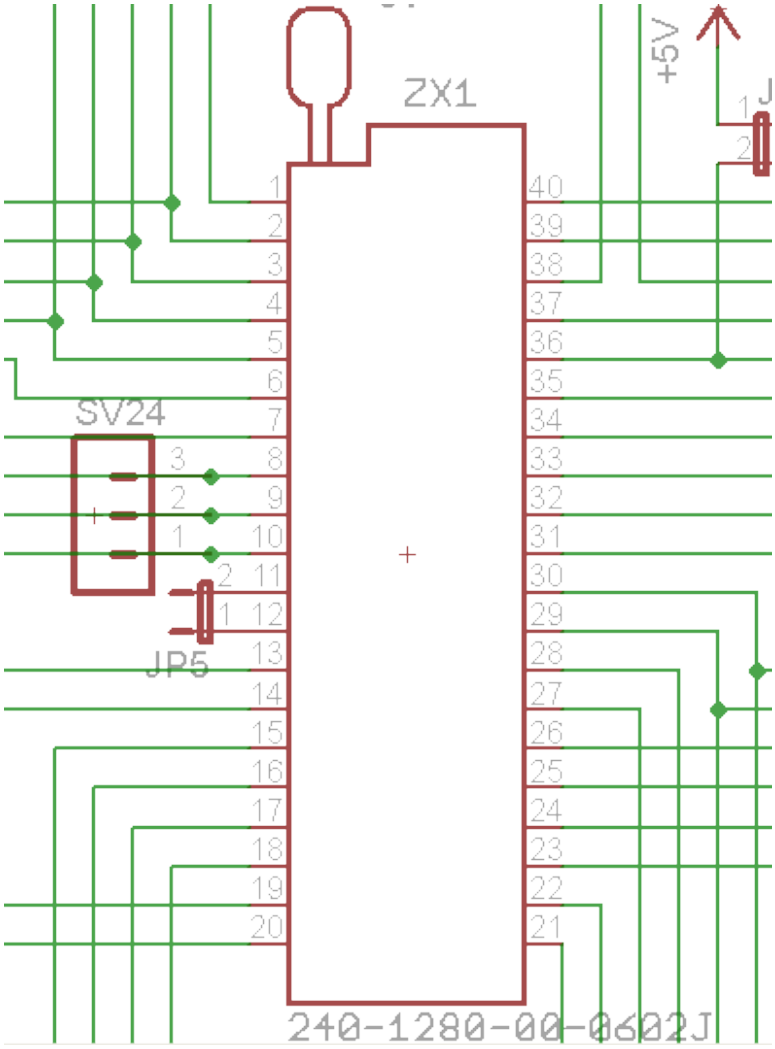
Lo schema completo, disponibile in PDF oltre che in formato Eagle, è molto ampio , va quindi stampato in formato A3, ma una volta suddiviso in blocchi funzionali, risulta molto più "focalizzabile" in merito ad ogni argomento trattato.

Fintanto che l'utente non ha buona familiarità con il layout della scheda sarà possibile muoversi agevolmente tra le sue potenzialità solo con gli schemi alla mano. Risulta così un ottimo esercizio per i tecnici in fase di formazione, ad esempio nelle scuole.

La scheda necessita di alcune impostazioni di base, a seconda del PIC interessato. Nel nostro corso tenderemo ad usare il 16F877A e non A, o processori simili a 40 pin, quindi una volta impostata la scheda tenderemo a non spostare o a farlo in maniera minima i jumper e deep switch.

Gli utenti che intendessero usare sempre e solo questo microcontrollore potranno ad esempio non montare il quarzo da 4Mhz dato che questi usa quello da 20.

Per quanto riguarda il connettore **ICSP**, lascio come utile esercizio al lettore il cercare di volta in volta in quale dei numerosi pin degli streep line maschio disponibili scoprire dove vanno connessi i 5 fili colorati, ovviamente si segue l'indicazione del software di programmazione come spiegato in precedenza.

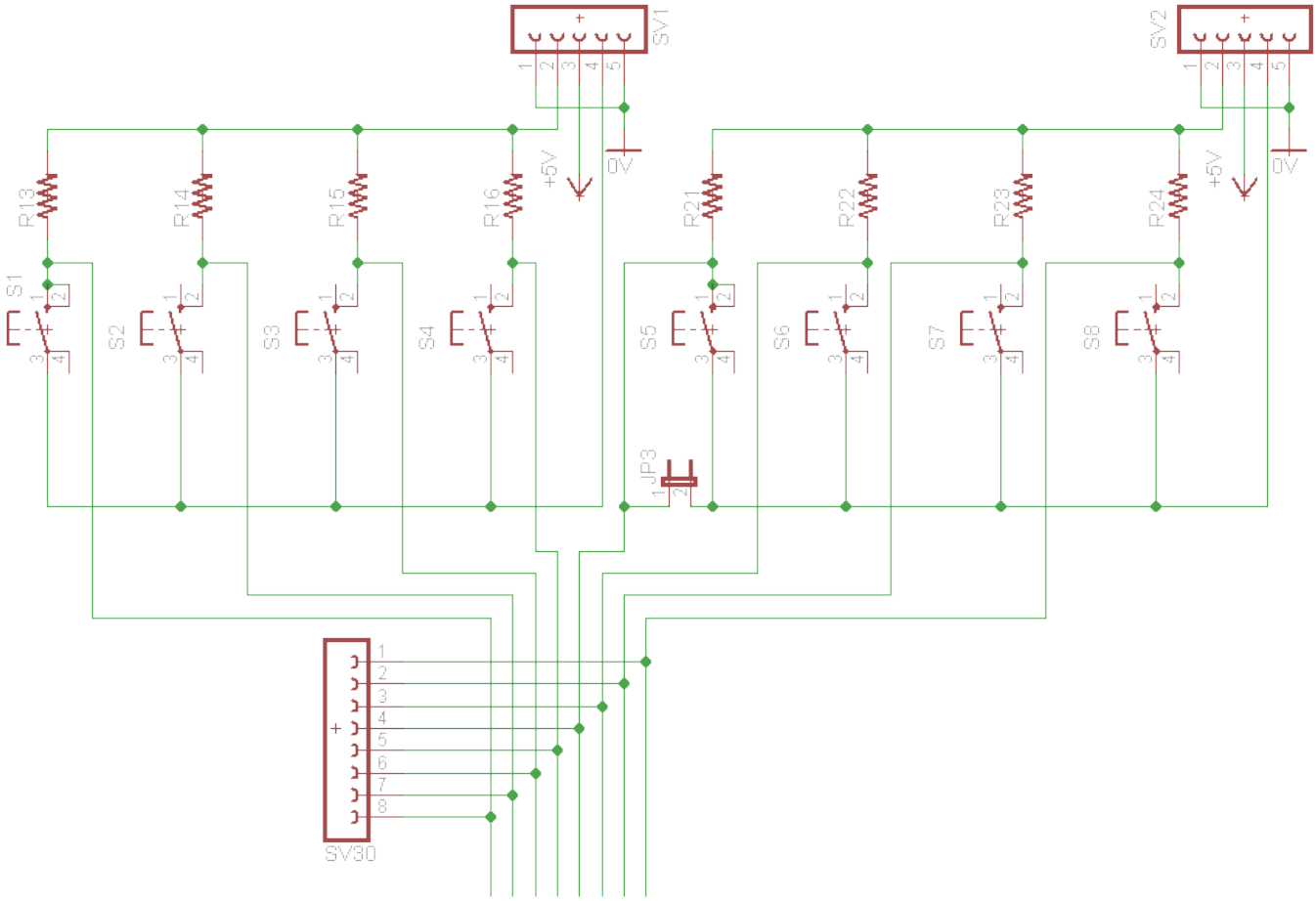


connettore TEXTOOL detto anche ZIF montato sulla Micro-GT PIC versatile I.D.E.

Gli **ingressi digitali** a bordo della Micro-GT sono tanti e ampiamente configurabili. Lo schema elettrico è riportato nell'immagine successiva. Concetto base del controllo a microprocessore è che i segnali possono giungere ai pin di I/O sia in logica diretta che in logica negata, spesso con preferenza per la seconda opzione. In fase di Debug, come già accennato nel precedente capitolo di "Let's go pic!" è essenziale impostare a simulatore software, o hardware che sia, la corretta configurazione dei pulsanti o interruttori, ovvero normalmente aperti o normalmente chiusi.

Molte demoboard disponibili in internet non danno la grande potenzialità disponibile nella Micro-GT di poter configurare come pressato=0, normale=1 oppure viceversa i pulsanti.

Questo è possibile, a gruppi di 4 pulsanti.



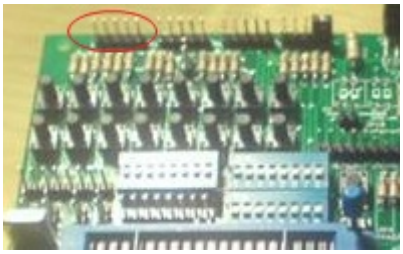
Osserviamo lo schema elettrico della sezione di ingressi digitali e focalizziamo l'attenzione sugli streep SV1 e SV2. La configurazione è progettata in modo che le medesime resistenze possa diventare pull-up o pull-down a seconda che i jumper siano innestati sugli streep allineati tutti a sinistra o a destra.

Le configurazioni sono:

- jumper allineati a sinistra -> Pull-down, (i pulsanti lanciano 1 logico quando pressati).
- jumper allineati a destra -> Pull-up, (i pilsanti lanciano 0 logico quando pressati, normalmente arriva 1).

I pulsanti sono isolabili dallo ZIF tramite l'apertura dei deep switch, e sono prelevabili dalla scheda per essere portati ove sia necessario, anche bordo macchina tramite dei cavi flat innestabili nel corrispondente streep line maschio (ad esempio SV30 e analoghi per gli altri PORT). Analogamente, si potrà collegare il cavo flat a monte del del deep switch, ovvero dal lato ZIF, dando la possibilità di entrare o uscire con segnali verso il relativo PORT del PIC. Questa modalità l'ho indicata come "direct I/O".

Abbiamo quindi massima duttilità di utilizzo.



Nella foto vediamo i 16 streep configurabili "up/down" suddivisi a gruppi di 4. In ogni gruppo di 4 vanno

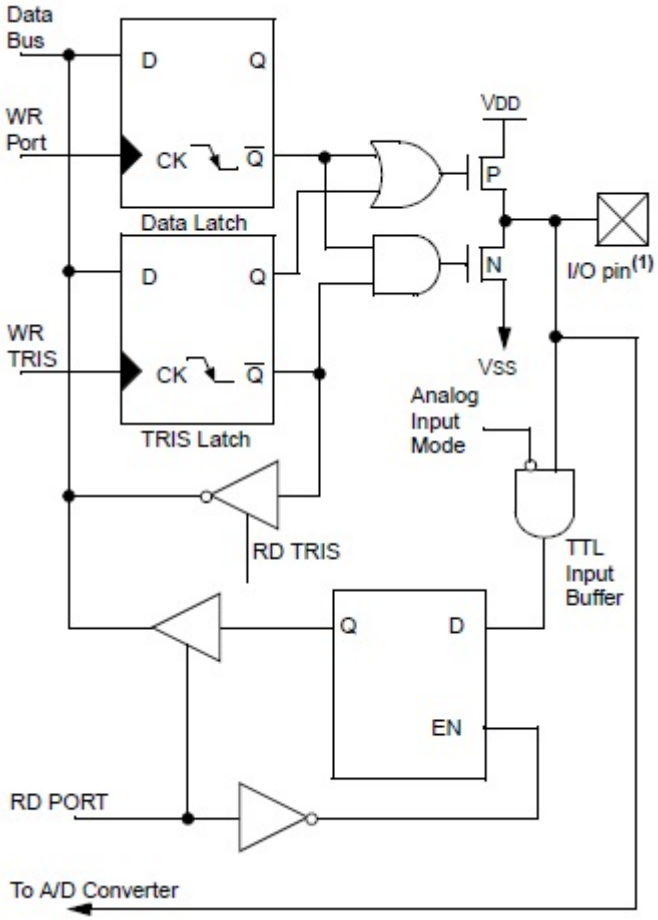
innestati almeno 2 jumper, o allineati a destra o allineati a sinistra per non avere pin flottanti rispetto alla massa.

In alcuni casi dipendenti dal tipo di housing (alloggiamento del controllore, o meglio scatolina da 40, 28, 18 pin ecc) il pulsante di reset risulta spostato in pin diverso dal numero 1. (è indicato con MCLR negato dalla barretta che sovrasta la scritta, ovvero attivo quando lancia al pin lo zero logico). Per questo motivo è stato inserito il jumper JP3, che in certi casi può anche portare semplicemente un segnale di alimentazione a specifici pin. Le casistiche sono davvero tante quindi il tecnico che utilizza questo sistema di sviluppo dovrà arrangiarsi quando si troverà davanti a un caso particolare. Sappiate che data la grande accessibilità ai pin, una soluzione c'è sempre. Bisogna studiarsi un po' l'hardware.

Hardware delle porte di I/O digitale

Gli ingressi

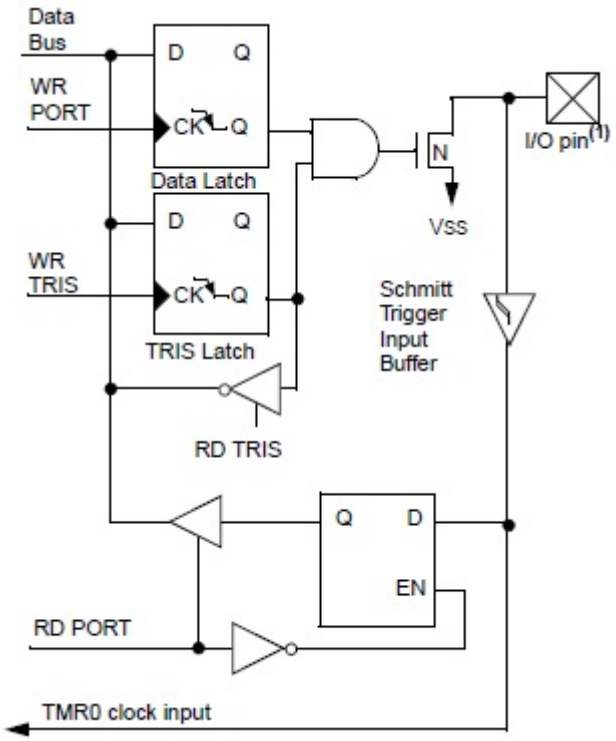
Facciamo una rapida analisi, ma non troppo superficiale del funzionamento delle porte quando configurate come digital I/O.



I Pin da 2 a 7, escluso il PIN 6, del processore di riferimento sono implementati internamente come rappresentato nello schema elettrico sovrastante. Facciamo una veloce analisi:

- Rettangoli grandi: Sono Flip/Flop di tipo "D" ovvero Delay. si ottengono dei FF di tipo Set/Reset, nella variante JK (che è una particolare cascata degli elementi base SR "set/reset"). Si ottiene un FF tipo D, collegando assieme gli ingressi JK della variante del SR, ma negando tramite una porta NOT l'ingresso K rispetto al J, e questo collegamento è interno così che, a parte il segnale di clock, il dispositivo presenta 3 terminali anziché i canonici 4 più i controlli di un normale flip flop. Il dispositivo ha l'effetto di latchare il segnare (tenerlo fermo o bufferizzarlo) aggiungendo una sorta di ritardo. I due flip flop in alto sono sincronizzati con il loro segnale di clock, mentre quello più in basso, non disponendo di questo ingresso ritarda e bufferizza il segnale rispetto a quanto si presenta in forma TTL al suo ingresso D.
- Le uscite Q e Q-negato di questi Flip flop sono pilotate dal segnale posto in ingresso dei due FF tipo D in alto, tali segnali corrispondo al comando di accensione o meno tramite **PORTA=0b00000001;** (riferito a RA0), e all'impostazione della direzione nel registro TRIS.

- Il mosfet a canale P si trova pilotato quando in gate quando siamo configurati come ingresso e non ci sono bit inviati in uscita tramite **PORTA=0b00000001;**
- Il mosfet a canale N avrà un uno logico in base quando il registro TRIS configura il pin come output, (TRISA=0;) e il PORTA è pilotato con tutte e uscite basse, ovvero **PORTA=0b00000000;**
- La configurazione complementare dei due mosfet determina poi lo stato dell'uscita se impostata come tale o la possibilità di leggere il bit quando ingresso. Da qui possiamo pretendere una discreta corrente, come già detto di circa 20mmA che non è poco. Facciamo comunque attenzione a non fare sciocchezze e di conseguenza bruciare la porta. Per dare un ordine di grandezza con questa corrente possiamo pilotare direttamente un led ad alta luminosità, o pilotare direttamente un display a sette segmenti, o un transistor di discreta potenza.
- La porta Nand (con la punta verso il basso nel lato destro dello schema) è quello che preleva il bit dal campo e gli permette l'ingresso verso i bus e il core del microcontrollore. questo è consentito se il segnale "analog input mode" è a zero, cosa che indirettamente deriva dal registro ADCON e altri. (si approfondirà in sede adeguata).
- I due flip flop "latchanno" (che storpiatura di lessico) i segnali, ovvero le tengono fisse in uscita anche se in ingresso il segnale non è più presente.
- Altri elementi circuitali sono di utilità intuitiva, ovvero invertono dove serve il segnale, e semplicemente fungono da driver.



Il pin 6 del processore 16F877 ha una configurazione Hardware leggermente diversa rispetto ai pin del medesimo PORT A, il cui schema è già stato analizzato. In effetti a questo pin (il 6 nel processore di riferimento ma potrà avere un'altra posizione in modelli diversi) è abbinato al "segnali di clock in entrata per il timer hardware 0", in inglese "timer zero clock input" abbreviato **T0CKI**.

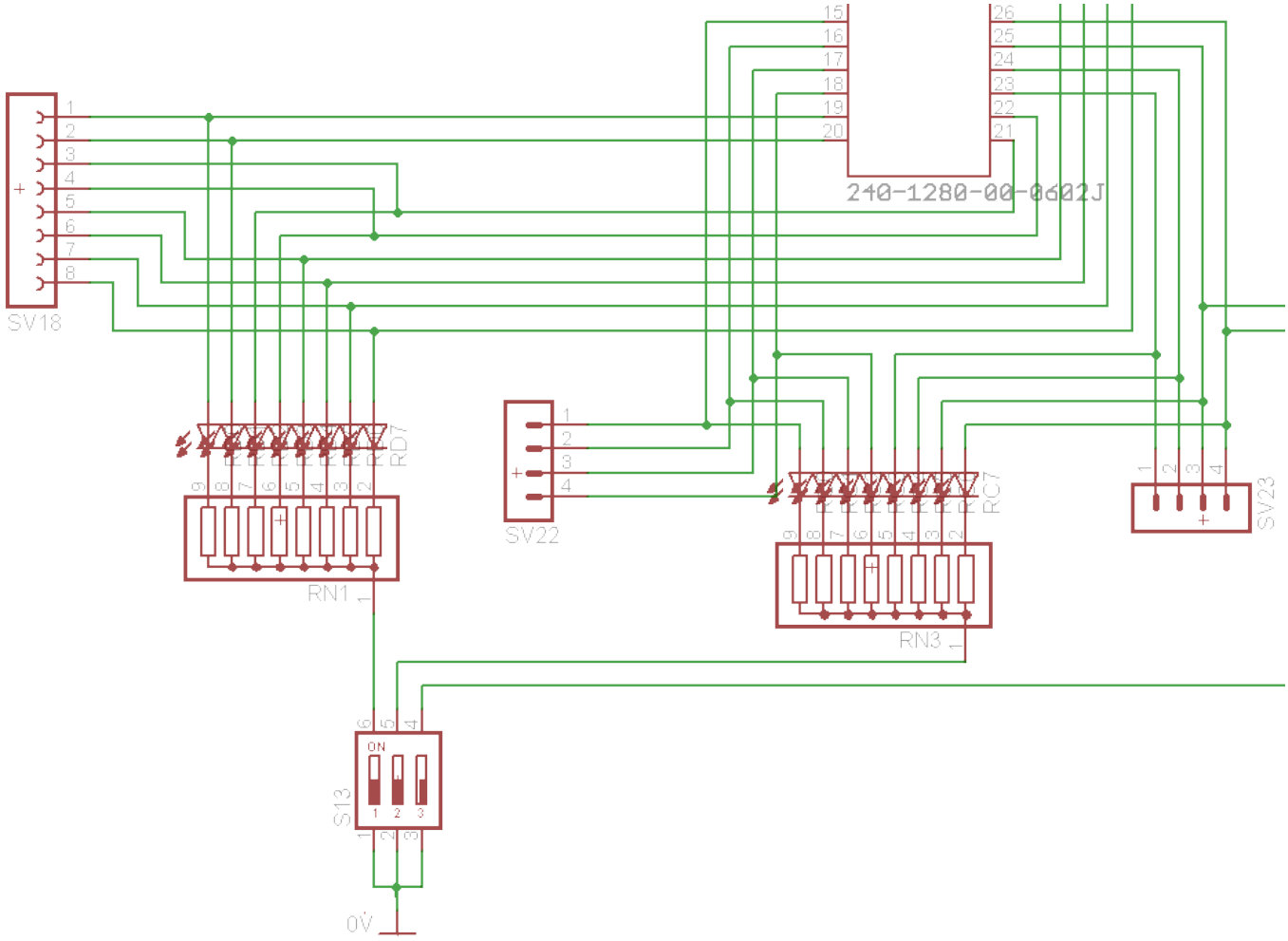
Dal punto di vista dell'architettura questo punti di I/O differisce dagli altri per le seguenti cose:

- Non è abbinato a un canale analogico.
- Funziona in modalità **open drain**, ovvero quando impostato come uscita la corrente del carico attraversa il mosfet e va verso la massa tramire l'unico canale "N" disponibile.
- E' abbinato come già detto a un segnale impulsivo esterno che pilota il TIMER hardware TMR0.
- Strutturalmente è più semplice ma ha delle funzionalità più complesse. A volte è meglio evitare l'uso di questa porta come ingresso digitale e anche come uscita per evitare problemi di interfacciamento o ambiguità di programmazione. Tutte le funzioni sono comunque disponibili.

Le uscite

Le **uscite digitali** a bordo sono 24 e accessibili sia tramite "direct I/O" esattamente come spiegato per gli ingressi, che in modalità free port, ovvero portabili in campo tramite cavo flat innestato negli streep line predisposti, ad esempio SV18, SV22, SV23, ecc.

Nei casi in cui la simulazioneED on board desse fastidio è possibile isolare la linea agendo nel deep switch S13, relativo al PORTB,C,D. I segnali saranno disponibili in freeport ma il led non si accenderà.



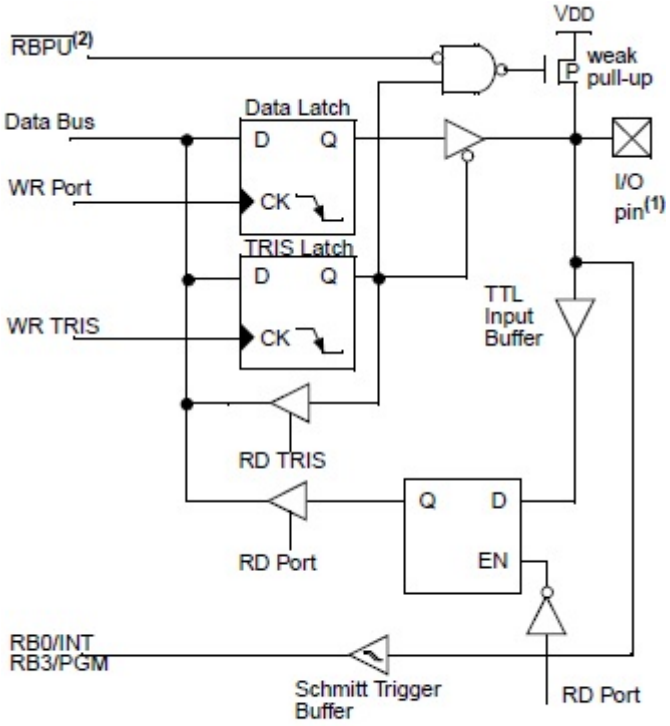
Il PORTE è di solito riservato alle linee di controllo degli LCD, è infatti composto da sole tre linee e verrà spiegato nell'apposito capitolo del tutorial "Let's go PIC".

Nota importante: benchè i PORT del PIC siano piuttosto robusti, arrivano a circa 20mA, corrente più che sufficiente al giorno d'oggi per pilotare qualunque cosa, è importante interfacciarsi comunque tramite transistor o comparatori operazionali, nonché optoisolare le sezioni di potenza.

I PORT B, C, D, E, hanno tutti delle caratteristiche aggiuntive o in meno rispetto a quella base, ma nella sostanza funzionano alla stessa maniera.

Al PORT B è ad esempio abbinata l'importante funzione, abilitabile tramite impostazione del registro... dei pull-up interni, che in qualche caso permette di sfruttare il mosfet interno come facente funzione della classica resistenza da 10K che solitamente "appendiamo" al segnale Vcc, o alla massa, al fine di non avere condizioni di segnale flottante prima dell'azione sul pulsante, spesso normalmente aperto (vedi Micro-GT) posto in serie ad essa.

La configurazione più stabile, e quindi da preferirsi è, per quanto riguarda gli input digitali è "normale 1 -> comando 0" che si ottiene ad esempio con una resistenza di pull-up con in serie un pulsante normalmente aperto. Il segnale prelevato dal centro di questa serie è portato direttamente al pin configurato come ingresso.

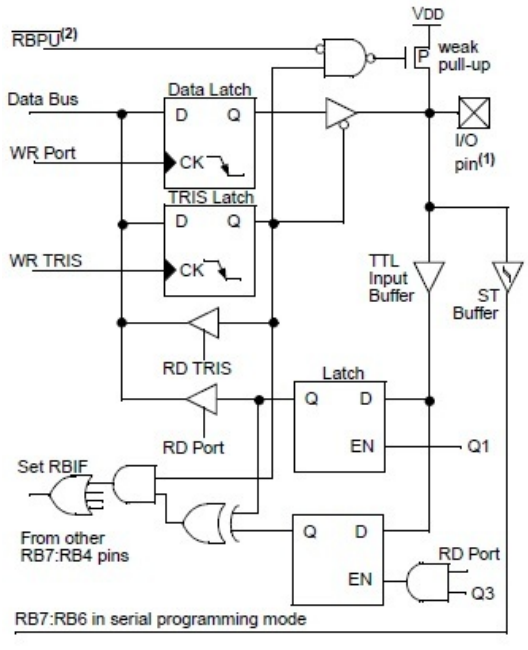


Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

I pin RB0 e RB3 hanno lo schema elettrico interno che vediamo nella figura sovrastante. La prima cosa che salta all'occhio è la presenza del segnale RBPU, attivo basso che come accennato, abilita la funzione di pull-up interni (non disponibili nel PORT A). Tali pull-up vengono automaticamente disabilitati quando la porta è commutata in uscita. La nota a piede dell'immagine chiarisce che esiste un diodo di ricircolo e di protezione, puntato verso Vdd che garantisce funzionamento. La presenza del TTL buffer, permette di non bruciare il punto di I/O nel momento in cui arrivano tensioni diverse dai 5V, come nel caso della programmazione. (questa questione sarà approfondita, per il momento prendiamola buona così).

RB0/INT è un ingresso che può acquisire un segnale di interrupt, ovvero quello che lancia una routine di servizio mettendo per pochi cicli machina in attesa il programma in corso (tecnica della commutazione del contesto) dando l'impressione che siano in esecuzione contemporaneamente due compiti dati da svolgere al microcontrollore. L'abilitazione e la configurazione dell'interrupt ha questa sintassi `INTDEG bit(OPTION_REG<6>)` suggerita dal manuale di microchip, in realtà vedremo presto come fare usando il C16.

I pin RB6 e RB7 sono invece delegati alla programmazione in circuit.

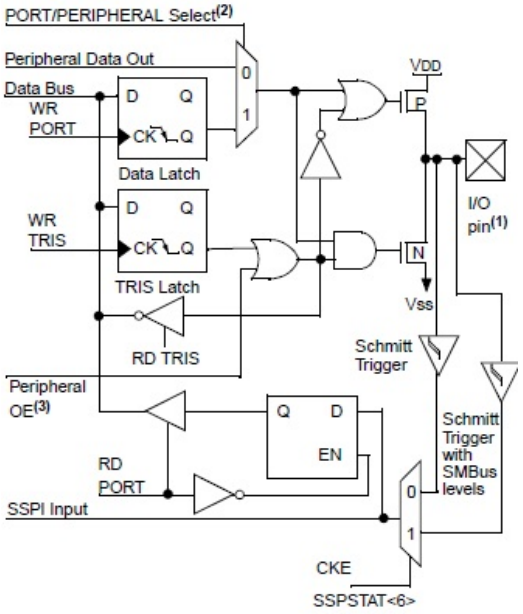


Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

Quando si usa la modalità di programmazione seriale in circuito a bassa tensione ICSP tipo LVP e sono abilitati i pull-up interni, bisogna azzerare nel registro TRISB il pin RB3.

E' importante sapere che il PORT B non è il più adatto a funzionare in modalità multiplexig a causa della presenza nell'estensione dei suoi 8 bit di queste funzionalità aggiuntive.

E' più adatto al multiplexing e al polling il PORT C di cui riporto sotto lo schema elettrico relativo, come per A e B ad un solo pin.

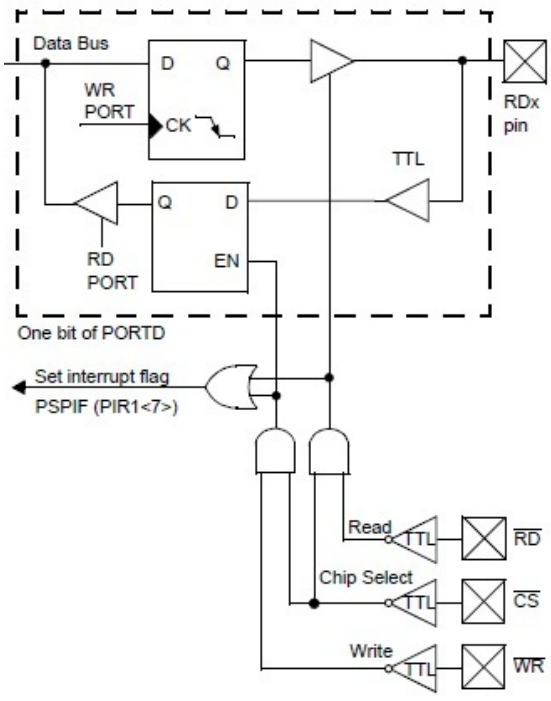


Note 1: I/O pins have diode protection to VDD and VSS.
2: Port/Peripheral select signal selects between port data and peripheral output.
3: Peripheral OE (output enable) is only activated if peripheral select is active.

Il PORT C contiene anche le porte seriali di comunicazione, e nei pin RB3 e RB4 è possibile abilitare il protocollo di comunicazione I^2C.

Vedremo invece che la porta RS232C è collegabile ai pin `RX->RC7` e `TX->RC6`.

Il PORT E, che possiede solo 3 punti di connessione, è predisposto per il protocollo PSP, parallel slave port, particolarmente indicato per la comunicazione tra due processori, ad esempio tra il nostro microcontroller e quello a bordo dei display LCD ad esempio a 2 righe per 16 caratteri, secondo lo standard Hitachi.



Note: I/O pin has protection diodes to VDD and VSS.

Vedremo al momento opportuno come questo PORT E sia collegato al funzionamento del PORT D che con i suoi 8 bit può implementare un efficace scambio di dati con le periferiche.

- `RD -> Attivo basso:RE0`
- `WR -> Attivo basso:RE1`
- `CS -> Attivo basso:RE2`

La programmazione inizia da qua

Come creare un alias

Quando il programma ha una certa estensione può diventare complesso o perlomeno scomodo usare i nome canonici dei punti di I/O, ad esempio RA2, RC3,RB0 ecc. pensiamo ad esempio a un programma in cui vengano citati un migliaio di volte e con la sequenza imposta dalla logica del programma. Vediamo ad esempio il seguente costrutto if:

```
if((RA0&&RA1)^RA2){ }
```

Se lo compilate all'interno di "main(){ }" (il corpo del main è dentro le parentesi graffe) la compilazione va a buon fine e viene creato un .hex, ma francamente è poco leggibile dato che le variabile sono anonime.

Lo stesso costrutto, rivisto secondo gli alias assume il seguente aspetto, palesemente più significativo:

```
if ((start && no_allarme) ^ bypass_OK) { }
```

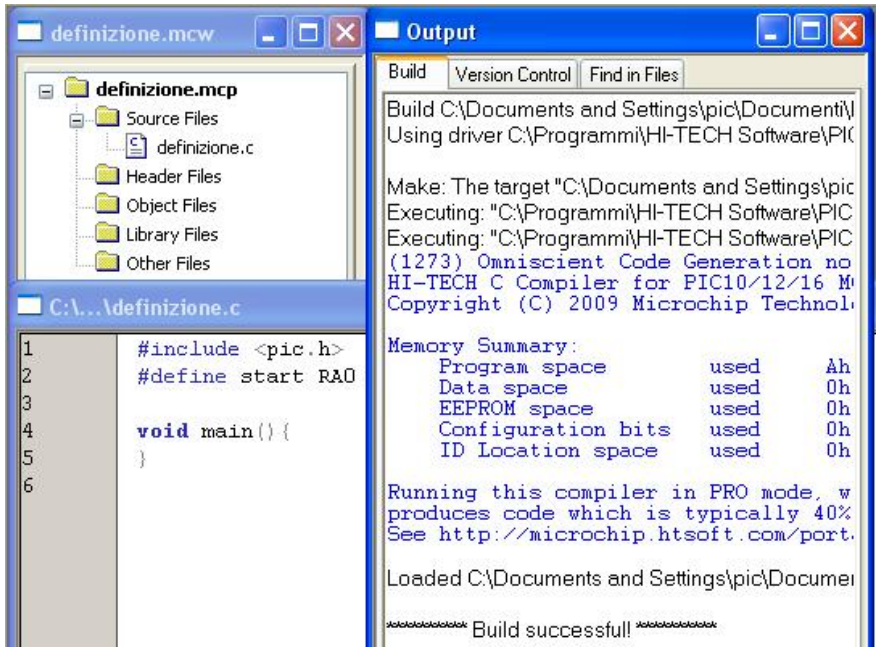
Il programma esegue il corpo dell' IF, ad esempio l'avvio di una macchina industriale, "se" `(if = se)` (è stato premuto il comando di start `"&&"` (and, significa "e anche"), oppure presente un segnale di consenso prioritario, che bypassa le prime due condizioni, ("oppure" ^ = or logico)

Notevolmente più leggibile !!!

Rimane il problema su come impostare questi alias. La sintassi è la seguente:

```
#define start RA0
```

Notiamo che la riga non è chiusa con il terminatore standard dell'ansi C, ovvero il " ; " questo è possibile perché il costrutto "#" (leggasi "number") seguito dal costrutto `"fine"` abbina la label (etichetta) seguente al punto di I/O, d esempio RA0 che termina la riga di comando.



Nell'immagine sovrastante vediamo il risultato di una compilazione su piattaforma MP-Lab e compilatore Hitech "C16", ovvero l'universal tool suite che è lo strumento ufficiale della casa costruttrice.

Ho compattato le finestre per ragioni di spazio, ma in alto a sinistra abbiamo l'albero del progetto, in basso a sinistra il programma minimale che esegue solo un alias sul pin RA0 il quale assume il nome "start", dato che probabilmente a bordo macchina è il pulsante di avvio, e lo manterrà per tutto il programma.

Nella colonna di destra vediamo la finestra del "sommario" della compilazione. Ci addentreremo di più nelle pubblicazioni future di "Let's go PIC". Per il momento accenniamo al fatto che ci mostra la gestione delle aree di memoria e, fondamentale, se la compilazione è andata o no a buon fine. Vediamo nel nostro esempio la scritta ***Build successful !***, che auguro a tutti di vedere sempre o perlomeno il più frequentemente possibile.

Esercizio: Creazione dell'alias

Copia il sorgente qui sotto e incollalo su un file notepad di windows. Salvalo con nome "definizione.c" poi leggi la prima edizione di "Let's go PIC !" e crea un progetto *definizione.mcw*. Esegui la compilazione fino ad ottenere il file *definizione.hex*

```
#include <pic.h>
#define start RA0 //implementa l'alias

void main(){
   if((RA0&&RA1)^RA2){
   }
}
```

Attenzione: Durante lo svolgimento dell'esercizio potrai incontrare dei problemi con le estensioni dei file, in modo particolare quando li crei con notepad anziche con l'editor Hitech integrato in MP-Lab. Risolvi la questione editando le proprietà della cartella che contiene il file e togliendo l'opzione, in proprietà cartella, nascondi estensione file. Se non fate questo potreste avere l'impressione che un file prova.c abbia in realtà la doppia estensione nascosta prova.c.txt che vi farà impazzire con errori strani di compilazione e conseguente nonon creazione del file .hex a noi essenziale per flashare ilel PIC.

Ricompila il programma definendo RA1 come FTC (che indica una fotocellula) e RA2 con il nome bypass.

Come possiamo notare la programmazione che utilizza gli alias è molto più elegante di quella con indirizzi reali, e tutto sommato aggiunge semplicità e leggibilità al codice sorgente. Una volta presa mano non ne potrete più fare a meno.

non proseguite la lettura se prima non siete riusciti a portare a buon fine questo esercizio.

L'antirimbalzo software

I contatti meccanici, diversamente di quelli allo stato solido simulati con transistor o mosfet, sono soggetti a inerzie e ancor peggio a reazioni elastiche che si riassumo in "rimbalzi" in fase di chiusura.

Se la lettura di questo segnale di input fosse molto rapida, la curva di elongazione e successive oscillazioni smorzate possono essere interpretati come commutazioni dello stato logico, trasformando l'immagine del gradino di heviside in un pettine di impulsi rapidi in grado di genera instabilità all'automazione.

Dal punto di vista elettrico la cosa è risolvibile ponendo un piccolo condensatore, qualche decina di picofard, in parallelo al contatto che si porta in chiuso. Questi, avendo funzione di filtro passa basso, cortocircuitando a massa le armoniche ad alte frequenza del fenomeno, smorzando o eliminando le elongazioni e le successive oscillazioni di assestamento.

Dopo il transitorio il condensatore si troverà carico alla tensione del segnale logico risultando quindi trasparente. Viene introdotta una incertezza di stato in fase di apertura che dipenderà dalla velocità di scarica del condensatore, anche se la curva essendo molto rapida potrà assomigliare fortemente a una rampa. Non sarà comunque mai identica al fronte di discesa del gradino in "spegnimento" (transizione ON -> OFF del segnale logico).

La questione dei rimbalzi può essere gestita anche via software eseguendo una sorta di controllo sulla "permanenza alta" del segnale che si è presentato per un tempo breve ma comunque presente dopo il rilevamento della prima presenza.

Ovvero: Chiudo il contatto e ne rilevo la presenza tramite il costrutto IF, ma non ne consento subito l'effetto. Lascio passare un tempo che stimo essere quello necessario a stabilizzare i rimbalzi, verifico se il segnale è ancora presente.

Se dopo questa sorta di "filtro" il segnale è presente allora lo lascio passare consentendone l'effetto.

il trucco è:

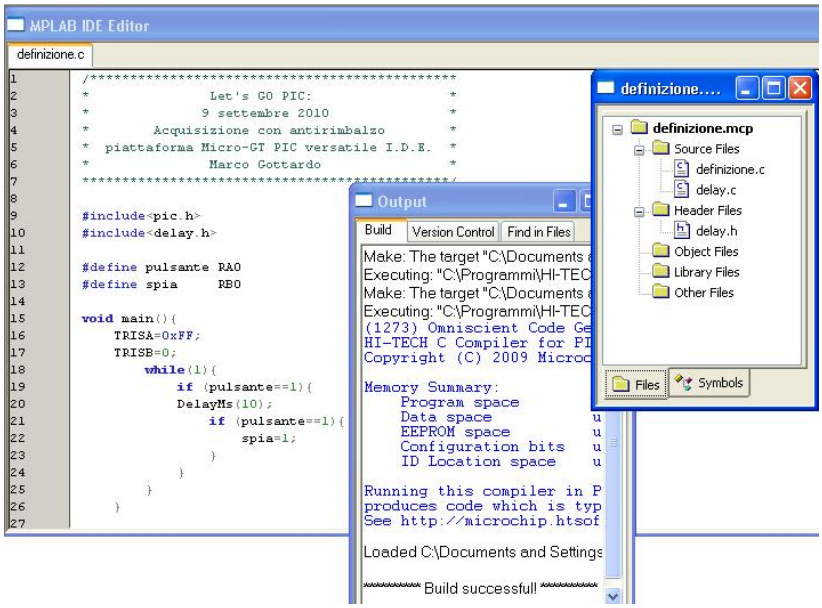
- controllo presenza con IF
- lancio di un tempo di stabilizzazione
- ricontrollo presenza con IF
- se il segnale è ancora presente consento l'effetto eseguendo il corpo del secondo IF.

Vediamo l'esempio ed eseguiamo il secondo esercizio

```
/************************************************
* Let's GO PIC: *
* 8 settembre 2010 *
* Acquisizione con antirimbalzo "bozza" *
* piattaforma Micro-GT PIC versatile I.D.E. *
* Marco Gottardo *
************************************************/

#include<pic.h>
#include<delay.h>

#define pulsante RA0
#define spia RB0

void main(){
   TRISA=0xFF;
   TRISB=0;
   while(1){
      if (pulsante==1){
         DelayMs(10);
         if (pulsante==1){
            spia=1;
         }
      }
   }
}
```

Nell'immagine vediamo l'ambiente di programmazione e la compilazione andata a buon fine. A questo punto sarete molto felici e contenti dato che tutto è segnalato, parole testuali del compilatore, come build successfull !, ...ma...

qualcosa non va. Prima di Leggere il seguito vi invito, a titolo di esercizio complementare al presente, di provare a ragionare su cosa sto cercando si segnalarvi come non trascurabile anomalia.



Il programma è sintatticamente corretto ma ha un grave difetto funzionale. Se avete letto attentamente la parte introduttiva che riguarda l' hardware delle porte di I/O non vi sarà infatti sfuggito il fatto che esse sono implementate (significa eseguite o costruite) tramite dei Flip/Flop e quindi c'è un effetto memoria. Questo effetto memoria è noto in elettronica digitale come "Latch" (tradotto letteralmente chiavistello o serratura). Al lato pratico rende stabile in uscita quello che sarebbe un impulso monostabile in input.

Testate il programma con RealPic simulator, tenendo presente che nella sezione visual dovete disporre un pulsante abbinato a un pulsante all'indirizzo RA0, impostato come normal -> 0, pressed -> 1 (agite con il tasto destro sul tab di settaggio del box pulsante), mentre in uscita metterete un LED abbinato a RB0.

Vediamo l'esempio ed eseguiamo il terzo esercizio

```
/************************************************
* Let's GO PIC: *
* 9 settembre 2010 *
* Acquisizione con antirimbalzo *
* piattaforma Micro-GT PIC versatile I.D.E. *
* Marco Gottardo *
************************************************/

#include <pic.h>
#include <delay.h>

#define pulsante RA0
#define spia RB0

void main(){
   TRISA=0xFF; //imposta come ingressi tutto il PORT A
   TRISB=0; //imposta come uscita tutto il PORT B
      while(1){ //inizio del ciclo infinito
         if (pulsante==1){ //test della condizione di comando
            DelayMs(10);
            if (pulsante==1){
               spia=1; //accende uscita
Definizione.zip            }
         }
      if (pulsante==0){ //test della condizione negata
         DelayMs(10);
         if (pulsante==0){
            spia=0; //spegne uscita
         }
      }
   }
}
```

Dal sottostante link puoi scaricare uno zip contenente, le librerie delay.h e delay.c, il sorgente definizione.c e due file .hex, il primo serve per capire che se non si mette il comando antagonista ovvero di spegnimento del punto di I/O 1 bit resta latch-ato (che brutto temine, non usatelo mai). Il secondo è invece la versione completa ovvero il compilato del sorgente sovrastante. Fate questo utile esercizio, caricate prima uno e poi l'altro nel simulatore "RealPIC simulator" come spiegato nel primo tempo del tutorial, e confrontate gli effetti.

Scarica definizione.zip

Giochiamo con le uscite

Benché nel paragrafo precedente si siano citate le uscite digitali, dato che alla pressione di un tasto collegato all'ingresso abbiamo acceso il LED collegato a RB0, ciò era molto poco rispetto a cosa si aspetta un lettore un po' più esperto riguardo all'uso delle uscite digitali.

Come precedentemente spiegato, i PORT, quando configurati come uscite tramite la corretta parametrizzazione del registro TRIS, hanno scopi simili, non identici.

Abbiamo detto che alcuni sono più adatti a pilotare gli LCD, altri sono predisposti per certi tipi di

segnale, alcuni sono open drain, ecc. ecc. dipenderà fortemente dal modello che useremo.

Abbiamo anche detto che una ventina di milliampere gli possiamo tirare fuori o drenare negli I/O digitali, quindi non dovremmo avere paura di collegare direttamente un display LED, se abbiamo porte disponibili ovviamente, e non sarà uno spreco dato che questo ci permetterà di eliminare il chip di decodifica, cosa che ho preferito non fare nella Micro-GT la quale monta a bordo un decoder BCD-7seg tipo CD4511.

E' di dovere cominciare a prendere confidenza con le uscite con le esperienze classiche ...segue il coro dei veterani che comunque hanno deciso di leggermi....NOOOOOOO !!! SUPERCAR NOOO!!!!!

Ma che ci posso fare io? è un passaggio obbligato, e al dire il vero anche molto richiesto e gettonato sopratutto dai ragazzi nella fascia di età a cui insegno a scuola.

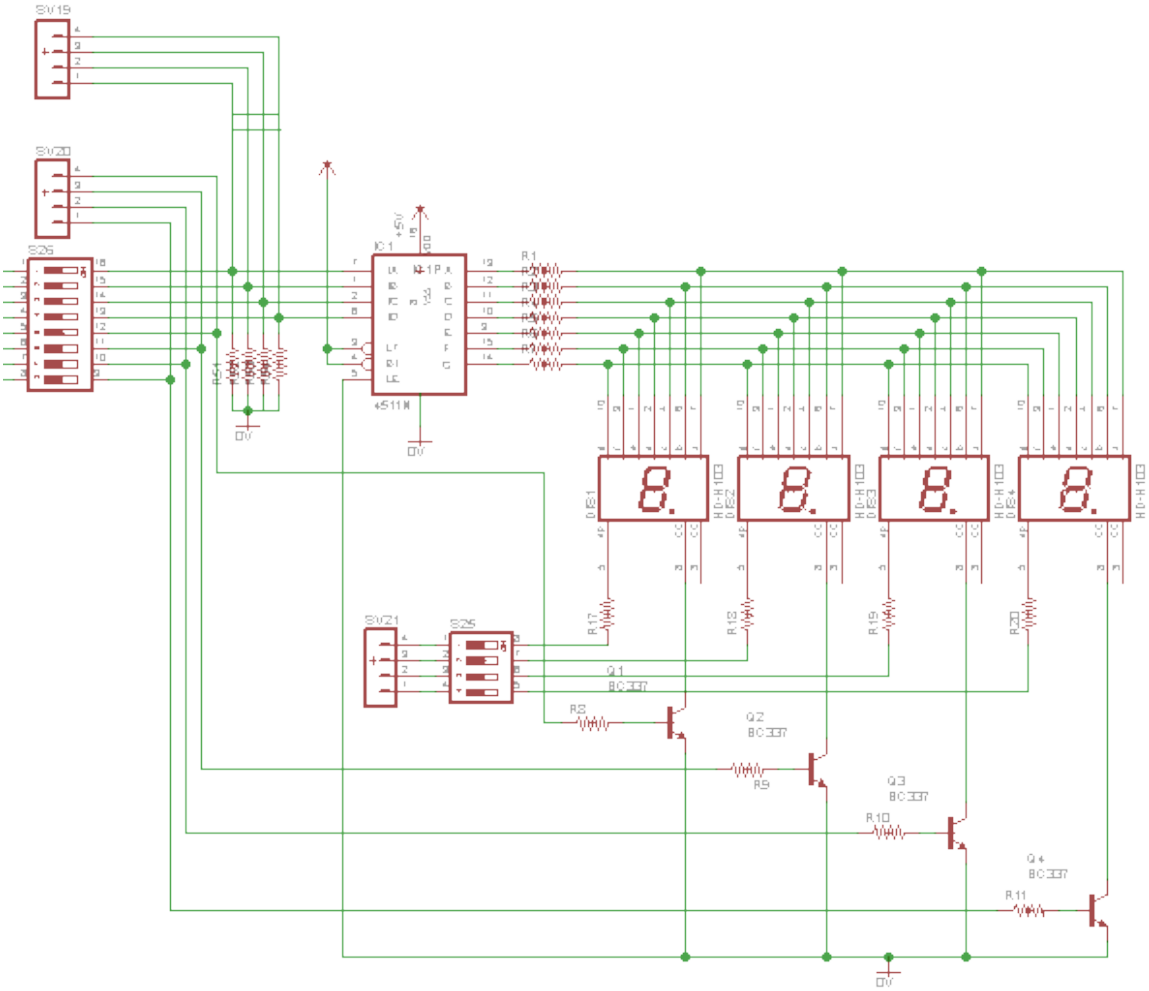
Facciamo così.... dato che insegno da molti anni anche nella formazione aziendale, quindi anche a persone che è già uscita dall'università, accontentiamo prima loro, e poi facciamo contenti i miei ragazzi con il tante ambito SUPERCAR.

Esercizio numero 4

La scheda Micro-GT, integra un decoder BCD-7segmenti LED pilotata dal circuito integrato **CD4511**. Questo integrato accetta su 4 linee di ingresso il codice BCD e lo decodifica nelle combinazioni necessarie ad accendere i segmenti del display in maniera corrispondente, ecco ad esempio che se gli presentate la combinazione `0011` lui imposterà le sue linee di uscita in modo compatibile all'accensione del numero 3, se gli proponiamo la combinazione 0100 lui decodifica sulle 7 linee una combinazione tale da accendere il numero 4, e così via per le cifre da 0 a 9. Le altre 6 combinazioni disponibili, infatti 2^4=16, come previsto dall'algebra combinatoria, sono dette ridondanti, ovvero prive di significato. Gli esperti potranno sfruttare queste combinazione per inviare in un bus condiviso dei segnali di controllo o informazioni "criptatate". o meglio trasparenti al display che le lascerà passare senza accendersi.

Scarica la documentazione tecnica dell'integrato CD4511

L'esercizio che propongo è abbastanza semplice, ma fortemente applicativo. Nella mia filosofia infatti è basilare la semplicità sopratutto in fase di apprendimento. Non nascondo che per un esperto questi programmini sono sulla soglia del ridicolo, ma come si dice dalle mie parti, nel Veneto, Nessuno nasce imparato... IO PER PRIMO.



schema elettrico della sezione display 7 segmenti della Micro-GT PIC versatile I.D.E.

Lo schema parla molto chiaro, i 4 display sono a catodo comune il quale viene connesso alla massa, dando la possibilità alla corrente di segmento di accendere il medesimo tramite un "interruttore elettronico", realizzato con un BJT di tipo **BC337**, classico transistor per segnale, NPN come si vede nello schema. Le resistenze R8,R9,R10,R11, limitano la RB al valore sufficiente a fare saturare il componente. In questa condizione la caduta di tensione tra collettore ed emettitore è prossima allo zero (a parte un piccolo residuo pari a circa 0,2V) e si comporta di conseguenza come un interruttore. A monte delle resistenze di base ci sono le 4 connessioni sia in modalità rt, ovvero collegando al PORT che vogliamo tramite un cavo flat, oppure semplicemente chiudendo il deepswitch accettando connessioni del bus al PORT che ho scelto in fase di progettazione, vincolandovi via software quello specifico PORT. *(vedi schemi elettrici)*

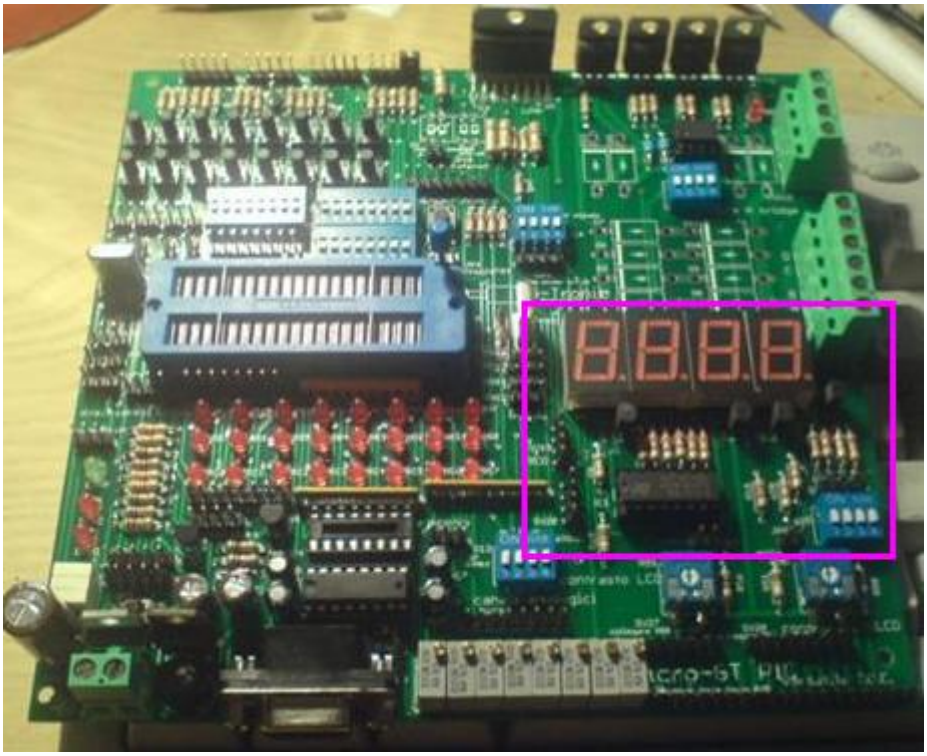
Se usiamo il bus connesso di default l'accensione del display avviene solo abilitando le linee del bus agendo sul deep SV26.

Le linee di controllo, ovvero le basi dei transistor, e le line dati che vanno al decodificatore BCD/7 segmenti sono ulteriormente separati su due streep line maschi indicati nella serigrafia con SV19 *(linea dati)* e SV20 *(linea controllo accensione)*.

Anche i punti decimali, se fossimo interessati ad accenderli sono disponibili nella modalità freeport, quindi tramite cavetto flat e chiudendo nel deep S25 le vie di interesse.

Nota importante: Se sviluppate una demoboard con lo schema che vi ho fornito sopra ricordatevi di connettere le masse alla massa della vostra mainbord, ovvero quella in cui avete il processore, altrimenti non si accende nulla.

La sezione interessata si trova nell'area indicata con il rettangolo rosa nella foto della Micro-GT PIC riportata qui sotto.



sezione del display a 7 segmenti della Micro-GT PIC versatile I.D.E.

Testo dell'esercizio:

Alla pressione di un tasto vogliamo accendere a display il numero corrispondente, ad esempio tasto 1 sul display compare 1, tasto 2 sul display compare 2, ecc.

Soluzione

```
/*********************************************************************
*                 Let's GO PIC !! 9 settembre 2010                   *
*                          Marco Gottardo                            *
*                   MICRO-GT PIC versatile I.D.E.                    *
*                  programma di selezione numerica                   *
*    abbina il tasto numero n al display numero n che visualizza n    *
*********************************************************************/

// imposta la frequenza del quarzo
```

```
#define XTAL_FREQ 20MHZ
#include <pic.h>

// fuse di configurazione
__CONFIG (HS & WDTDIS & PWRTEN & BORDIS & LVPDIS & DUNPROT & WRTEN &
DEBUGDIS & UNPROTECT);

#include <delay.h>
#include "setting.h"

//programma di test dell'interrupt

void main(void){
   settings();
   while(1){
      if(P1==0){
         DelayMs(10);//ritardo
         if(P1==0){
            PORTC=disp1;
            PORTB=1;
         }
      }

      if(P2==0){
         DelayMs(10);//ritardo
         if(P2==0){
            PORTC=disp2;
            PORTB=2;
         }
      }

      if(P3==0){
         DelayMs(10);//ritardo
         if(P3==0){
            PORTC=disp3;
            PORTB=4;
         }
      }

      if(P4==0){
         DelayMs(10);//ritardo
         if(P4==0){
            PORTC=disp4;
            PORTB=8;
         }
      }

   }
}
```

Questo programma contiene già molti punti di discussione. Ad esempio la riga di codice:

```
__CONFIG (HS & WDTDIS & PWRTEN & BORDIS & LVPDIS & DUNPROT &
WRTEN & DEBUGDIS & UNPROTECT);
```

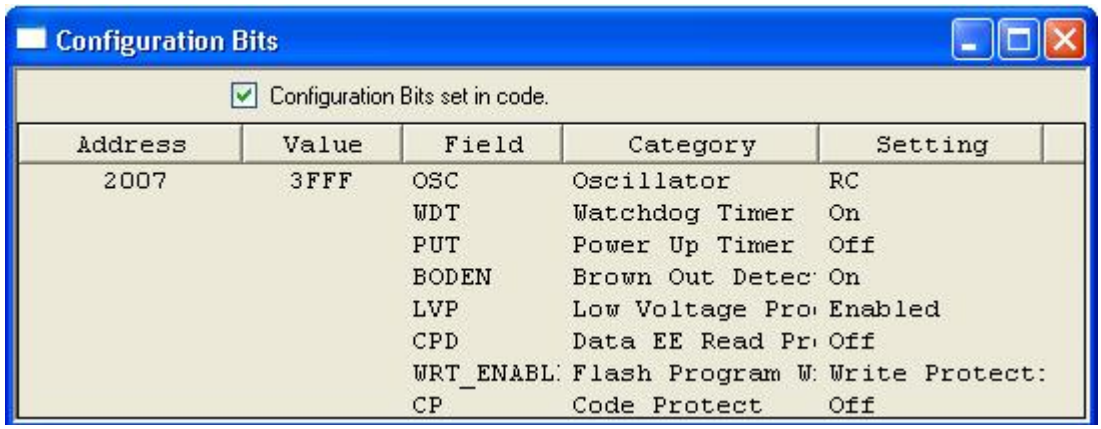
La incontriamo per la prima volta. Questa imposta il registro dei FUSE che sono settaggi indispensabili per il corretto funzionamento del PIC. La riga inizia con due "underscore" che l'editor unisce dando l'impressione che sia uno, dopodiché ognuno degli elementi separati da una "&" corrispondono ad un BIT che abilita o disabilita una funzione, chiamiamola di sistema.

vediamo ad esempio i primi due: HS e WDTDIS. sono rispettivamente l'impostazione dell'hardware interno in modo che possa accettare un quarzo esterno ad alta velocità, highspeed, e la "DIS" ovvero disabilitazione del "WDT" il famoso watchdog timer. Do solo un breve accenno al fatto che questo timer viene resettato in automatico ogni volta che il contatore di programma passa per la prima riga di codice. Se il timer dovesse scadere prima di un successivo riavvio significa che il processore e quindi il programma è impallato, e quindi non più in grado di rispondere alle sollecitazioni o stimoli sulle sue terminazioni di I/O. In questo caso l'intervento del WDT blocca tutto il sistema per evitare i gravi danni che una automazione può fare quando è privo di "controllo"

Il registro dei FUSE, il loro approfondimento e uso merita un capitolo specifico del nostro tutorial "Let's go PIC". Appena ho un momento libero ci lavoro un po.

Se questa riga non è inserita nel codice sorgente allora è possibile impostare i fuse direttamente dal software di sviluppo.

dal menu 'configuration' di MP-Lab richiamare 'bits di configurazione'. compare il seguente box di impostazioni.



A questo punto facciamoci una domanda: Perché impostare i fuse sul codice dato che lo possiamo fare esternamente sul software di sviluppo?

Risposta: Esiste una strategia software che permette l'inserimento del microcodice, ovvero dell'hex all'interno del PIC senza utilizzare ne il software di programmazione e sviluppo (ad esempio MP-Lab), ne una scheda di sviluppo ma semplicemente una sheda "target" che ovviamente metta almeno a disposizione le porte di comunicazione.

Questa modalità di flashare il PIC non richiede neppure l'alta tensione (i 13,2V che mette a disposizione la Micro-GT) perché il tutto avviene in modalità bassa tensione, ovvero quella che è già disponibile per alimentare il PIC, guardiamo ad esempio la quinta voce sull'immagine "configuration bits"sovrastante.

Dedicherò un capitolo del tutorial alla programmazione tramite bootloader. Per il momento è meglio non deviare troppo dal focus dell'articolo che sono i digital I/O.

Quando il codice diventa molto articolato è bene sfruttare la capacità del "C" di essere suddiviso in moduli esterni successivamente linkati all'interno di un unico file prima di essere assemblato.

La direttiva di #include è utilizzabile oltre che per i file di libreria anche per eventuali moduli esterni del nostro programma, sia essi di intestazione "header" salvati con estensione .h, oppure veri e propri sorgenti contenti dei metodi e funzioni salvati .C

Nella filosofia di programmazione del "C" i file header (.h) contengono i prototipi e le definizioni di eventuali comandi non di libreria.

Ecco il sorgente del modulo esterno setting.h

```
// programma base sviluppato per 16F877A
// chiudere i canali del DIP SW1 da 5 a 8
// aprire i canali del DIP SW1 da 1 a 4
// collegare con cavo flat a 4 conduttori lo streap SW19 con SW22
// gli ingressi del decoder CD4511 corrispondono a:
// A -> RC0
// B -> RC1
// C -> RC2
// D -> RC3


#define A RC0 // -> pin 15
#define B RC1 // -> pin 16
#define C RC2 // -> pin 17
#define D RC3 // -> pin 18

#define digit4 RB0   // pin 33 (LSB) cifra piu a destra
#define digit1 RB1   // pin 34
```

```
#define digit2 RB2   // pin 35
#define digit3 RB3   // pin 36 (MSB) cifra piu a sinistra

//pulsanti normal=1, pressed=0
//usare pull-up da 10Kohm
//SV2 con jumper allineati a destra
//il PORTA viene commutato in ingresso

#define P1 RA0   // comanda scrittura (1) su digit 1
#define P2 RA1   // comanda scrittura (2) su digit 2
#define P3 RA2   // comanda scrittura (3) su digit 3
#define P4 RA3   // comanda scrittura (4) su digit 4

char disp1=1;    // usate nella funzione scriviBUS nel modulo writeBUS
char disp2=2;
char disp3=3;
char disp4=4;

void settings(void){
   TRISA=0xFF;
   TRISB=0;
   TRISC=0;
   TRISD=0;
   TRISE=0;
}
```

Scarica la soluzione di questo esercizio

Seconda versione:

Vogliamo eseguire un ciclo automatico di visualizzazione dei numeri a diplay. Ogni digit viene multiplexato

```
/
*************************************************************************
*                      MICRO-GT PIC versatile I.D.E.                    *
*                     programma di selezione numerica                   *
*                   versione sia automatica che manuale                 *
*      abbina il tasto numero n al display numero n che visualizza n    *
*     nel funzionamento automatico premendo P5 genera il numero a 1Hz   *
*************************************************************************
/

// imposta la frequenza del quarzo
#define XTAL_FREQ 20MHZ
#include <pic.h>

//fuse di configurazione
__CONFIG (HS & WDTDIS & PWRTEN & BORDIS & LVPDIS & DUNPROT & WRTEN &
DEBUGDIS & UNPROTECT);

#include <delay.h>
#include "setting.h"

void main(void){
   settings();
   while(1){
      if(P1==0){
         DelayMs(10);//ritardo
         if(P1==0){
            PORTC=disp1;
            PORTB=1;
         }
      }

      if(P2==0){
         DelayMs(10);//ritardo
```

```
         if(P2==0){
            PORTC=disp2;
            PORTB=2;
         }
      }

      if(P3==0){
         DelayMs(10);//ritardo
         if(P3==0){
            PORTC=disp3;
            PORTB=4;
         }
      }

      if(P4==0){
         DelayMs(10);//ritardo
         if(P4==0){
            PORTC=disp4;
            PORTB=8;
         }
      }

      if(P5==0){
         DelayMs(10);//ritardo
         while(P5==0){
         // if(P5==0){
               PORTC=disp1;
               PORTB=1;
               DelayMs(200);
               PORTC=disp2;
               PORTB=2;
               DelayMs(200);
               PORTC=disp3;
               PORTB=4;
               DelayMs(200);
               PORTC=disp4;
               PORTB=8;
               DelayMs(200);
         // }
         }
      }
   }
}
```

il sorgente successivo è il setting.h, specifico per questa versione automatica e per la scheda Micro-GT PIC. per accendere correttamente i display dovete fare in modo che i segnali viaggino correttamente lungo i bus abilitano i deepswitch e settando gli opportuni jumper.

Il da farsi è indicato a commento, ovvero dopo il doppio slach "//" sulle prime righe del file setting.h che vedete qui sotto.

Lasciate pure queste righe di commento all'interno del file. saranno molto utili.

```
// programma base sviluppato per 16F877A
// chiudere i canali del DIP SW1 da 5 a 8
// aprire i canali del DIP SW1 da 1 a 4
// collegare con cavo flat a 4 conduttori lo streap SW19 con SW22

// gli ingressi del decoder CD4511 corrispondono a:
// A -> RC0
// B -> RC1
// C -> RC2
// D -> RC3


#define A RC0 // -> pin 15
#define B RC1 // -> pin 16
#define C RC2 // -> pin 17
```

```
#define D RC3 // -> pin 18

#define digit4 RB0 // pin 33 (LSB) cifra più a destra
#define digit1 RB1 // pin 34
#define digit2 RB2 // pin 35
#define digit3 RB3 // pin 36 (MSB) cifra piu a sinistra

// pulsanti normal=1, pressed=0
// usare pull-up da 10Kohm
// SV2 con jumper allineati a destra
// il PORTA viene commutato in ingresso

#define P1 RA0 // comanda scrittura (1) su digit 1
#define P2 RA1 // comanda scrittura (2) su digit 2
#define P3 RA2 // comanda scrittura (3) su digit 3
#define P4 RA3 // comanda scrittura (4) su digit 4
#define P5 RA4 // comanda ciclo di loop su digit 1-2-3-4

char disp1=1; // usate nella funzione scriviBUS nel modulo writeBUS
char disp2=2;
char disp3=3;
char disp4=4;

void settings(void){
   TRISA=0xFF;
   TRISB=0;
   TRISC=0;
   TRISD=0;
   TRISE=0;
}
```

Questo programma include un secondo file di intestazione che chiamerete writebus.h il cui sorgente è qui sotto. Esso contiene solo la funzione di scrittura sul port e quindi l'invio dei dati al decodificatore CD4511 della scheda di sviluppo.

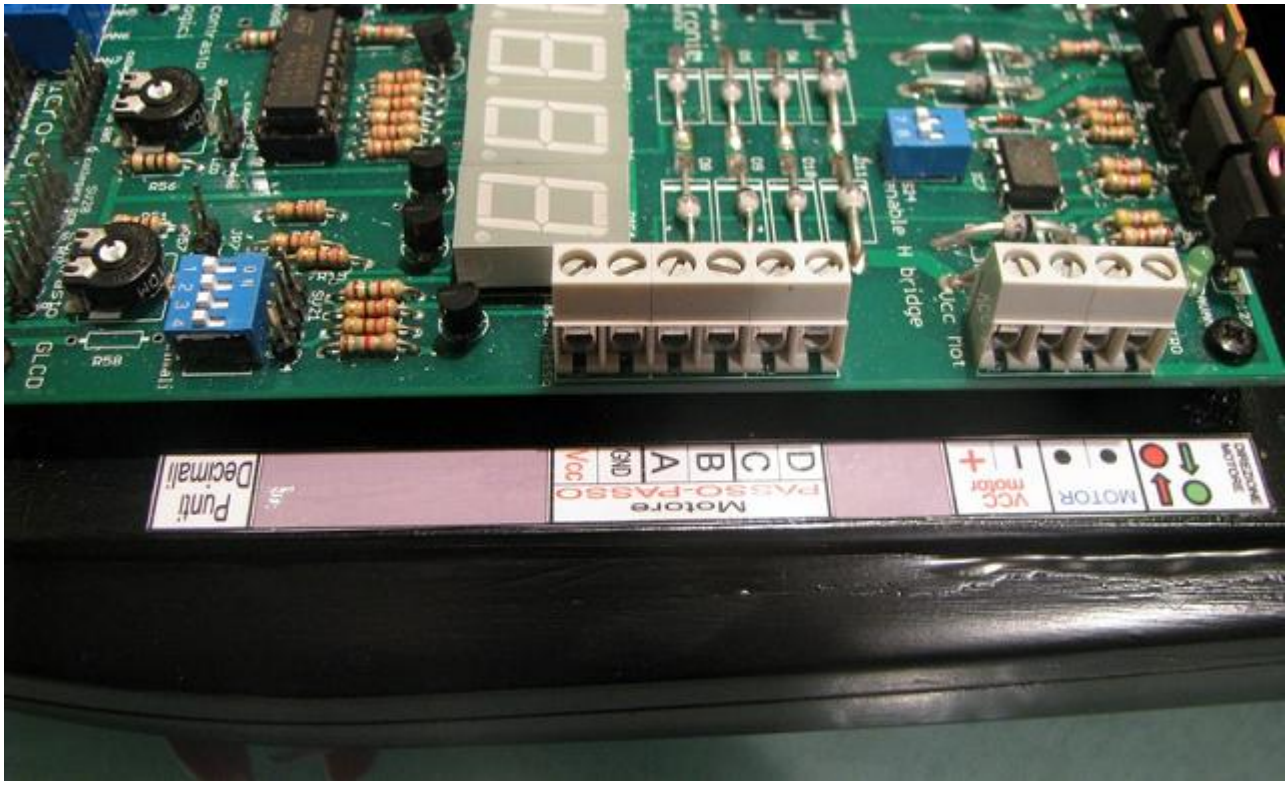
```
void scriviBUS(){
   if (PORTB==0b0001){
   }
   if (PORTB==0b0010){
   }
   if (PORTB==0b0100){
   }
   if (PORTB==0b1000){
   }
   PORTC=disp1;
}
```

In definitiva la versione automatica della scittura dei numeri sul display a 7 segmenti si compone di tre file:

1. writeBUS.h
2. select_number_auto.c
3. setting.h

ai quali aggiungo il quarto, il file: **select_number_auto.hex**

Scarica il file zippato



Nella foto un particolare delle morsettiere per il collegamento del motore passo situato proprio sopra al display a sette segmenti usato in questi esercizi.

Appendice:

Effetto luci supercar: L'hello world di chi inizia con i PIC.

Come promesso cercherò ora di fare contenti i principianti con qualche variante del classico esercizio "supercar". Chi è già esperto potrà saltare a piè pari questo paragrafo, anche perché, come ben sappiamo, la programmazione con le routine delay in line è sbagliata di concetto, dato che il processore è messo in attesa che scada un tempo e nel frattempo non è attento a cosa avviene nella sua periferia. Accenniamo, ma approfondiremo in un successivo capitolo che nella normalità delle cose il processore deve poter essere "interrotto" dal suo task per poter eseguirne uno prioritario con la tecnica detta dell'interrupt.

Credo che un principiante possa sicuramente afferrare il concetto ma non credo sia in grado di sviluppare un programma o anche solamente di gestirne uno già fatto.

Andiamo quindi per gradi ed una volta acquisita l'esperienza necessaria tutto verrà affrontato con maggior serenità.

Programma supercar per principianti:

Microcontrollore 16F877A, sistema di sviluppo Micro-GT PIC versatile I.D.E.

Selezionare il clock a 20Mhz con gli appositi jumper (vedi schemi elettrici).

Abilitare il PORTB e il PORTC agendo sui deepswitch corrispondenti.

Impostare i pulsanti come normalmente aperti, ovvero normali ->0, pressati ->1. Se realizzate il sistema reale saranno quindi "interruttori normalmente aperti con resistente di pull-down.

Se usate lampade di potenza ricordatevi di pilotare i triac con gli opportuni optoisolatori.

Scaricate il file hex che ho messo a disposizione e testatelo con Realpic simulator (vedere la prima parte di let's go pic per la spiegazione più accurata.

Data la natura didattica di questo semplice programma non mi assumo nessuna responsabilità per il suo eventuale uso al di fuori di questo ambito, del resto ho anche chiaramente detto che il metodo di programmazione con le routine di delay inline è sbagliato di concetto. Programmi analoghi vanno sviluppati con la tecnica dell'interrupt.

Ecco il codice sorgente:

```
 /*************************************************************************
                Let's GO PIC: gestione digital I/O
              sviluppato per Micro-GT PIC versatile I.D.E.
                   13 settembre 2010: Marco Gottardo
             Impostare correttamente i jumper sulla demoboard
              Gli ingressi sono bistabili (interruttori N.A.)
                                 * *
                           effetti disponibili:
                     car1 -> effetto supercar al PORT B
                     car2 -> effetto supercar al PORT C
          car1_2 -> effetto supercar dal PORTB prosegue sul PORTC
                                 * *
 *************************************************************************/

#include <pic.h>
#include <delay.h>

#define car1 RA0
#define car2 RA1
#define car1_2 RA2
#define stop RA3

void main(){
   TRISA=0x3F;   // 6 bit della porta A configurati come ingressi
   TRISB=0;      // tutta la porta B in uscita
   TRISC=0;      // tutta la porta C in uscita
   PORTB=0b00000000;   // inizializza B la porta tutta spenta
   PORTC=0b00000000;   // inizializza C la porta tutta spenta
   while(1){
      if (car1==1){ // effetto supercar al PORT B
         PORTC=0b00000000;   // azzera il port non in uso
         PORTB=0b00000001;
         DelayMs(255);
         PORTB=0b00000010;
         DelayMs(255);
         PORTB=0b00000100;
         DelayMs(255);
         PORTB=0b00001000;
         DelayMs(255);
         PORTB=0b00010000;
         DelayMs(255);
         PORTB=0b00100000;
         DelayMs(255);
         PORTB=0b01000000;
         DelayMs(255);
         PORTB=0b10000000;
         DelayMs(255);
         PORTB=0b01000000;
         DelayMs(255);
         PORTB=0b00100000;
         DelayMs(255);
         PORTB=0b00010000;
         DelayMs(255);
         PORTB=0b00001000;
         DelayMs(255);
         PORTB=0b00000100;
         DelayMs(255);
         PORTB=0b00000010;
         DelayMs(255);
      }
      if (car2==1){   // effetto supercar al PORT C
         PORTB=0b00000000;   // azzera il port non in uso
         PORTC=0b00000001;
         DelayMs(255);
         PORTC=0b00000010;
         DelayMs(255);
         PORTC=0b00000100;
         DelayMs(255);
```

```
         PORTC=0b00001000;
         DelayMs(255);
         PORTC=0b00010000;
         DelayMs(255);
         PORTC=0b00100000;
         DelayMs(255);
         PORTC=0b01000000;
         DelayMs(255);
         PORTC=0b10000000;
         DelayMs(255);
         PORTC=0b01000000;
         DelayMs(255);
         PORTC=0b00100000;
         DelayMs(255);
         PORTC=0b00010000;
         DelayMs(255);
         PORTC=0b00001000;
         DelayMs(255);
         PORTC=0b00000100;
         DelayMs(255);
         PORTC=0b00000010;
         DelayMs(255);
      }
      if (car1_2==1){   // effetto lungo dal PORT B prosegue sul PORTC
         PORTB=0b00000001;
         DelayMs(255);
         PORTB=0b00000010;
         DelayMs(255);
         PORTB=0b00000100;
         DelayMs(255);
         PORTB=0b00001000;
         DelayMs(255);
         PORTB=0b00010000;
         DelayMs(255);
         PORTB=0b00100000;
         DelayMs(255);
         PORTB=0b01000000;
         DelayMs(255);
         PORTB=0b10000000;
         DelayMs(255);
         PORTB=0b00000000; //pulisce portB per continuare su C
         PORTC=0b00000001;
         DelayMs(255);
         PORTC=0b00000010;
         DelayMs(255);
         PORTC=0b00000100;
         DelayMs(255);
         PORTC=0b00001000;
         DelayMs(255);
         PORTC=0b00010000;
         DelayMs(255);
         PORTC=0b00100000;
         DelayMs(255);
         PORTC=0b01000000;
         DelayMs(255);
         PORTC=0b10000000;
         DelayMs(255);
         PORTC=0b01000000;
         DelayMs(255);
         PORTC=0b00100000;
         DelayMs(255);
         PORTC=0b00010000;
         DelayMs(255);
         PORTC=0b00001000;
         DelayMs(255);
         PORTC=0b00000100;
         DelayMs(255);
         PORTC=0b00000010;
```

```
         DelayMs(255);
         PORTC=0b00000001;
         DelayMs(255);
         PORTC=0b00000000; // pulisce effetto memoria al portC
         PORTB=0b10000000;
         DelayMs(255);
         PORTB=0b01000000;
         DelayMs(255);
         PORTB=0b00100000;
         DelayMs(255);
         PORTB=0b00010000;
         DelayMs(255);
         PORTB=0b00001000;
         DelayMs(255);
         PORTB=0b00000100;
         DelayMs(255);
         PORTB=0b00000010;
         DelayMs(255);
         PORTB=0b00000001;
         DelayMs(255);
      }
      if (car1==0){
         PORTB=0b00000000; // pulisce memoria, porta B tutta spenta
      }
      if (car2==0){
         PORTC=0b00000000; // pulisce memoria, porta C tutta spenta
      }
   }
}
```

Scarica da questo link il programma completo con il file .hex da provare sul simulatore e da flashare nel PIC.

Contiene anche le librerie delay.h e delay.c da copiare nella directory include del compilatore.

Supercar MicroGT.zip