
Esercitazioni C Enhanced Midrange

3 – LED comandato da pulsante

In questa esercitazione vediamo come

- utilizzare un ingresso digitale.
- effettuare il debounce di un pulsante

Nella pratica, vogliamo comandare il LED con un pulsante.

Essenzialmente il programma dovrà accendere il LED quando il pulsante è premuto. Questo richiede di far individuare dal microcontroller lo stato del pulsante.

L'hardware è lo stesso della seconda esercitazione, senza modifiche.

3.1 - I/O digitali come ingressi.

Abbiamo visto finora come è possibile comandare uscite digitali, ma, i pin possono funzionare anche come ingressi digitali.

La possibilità di definire i pin come uscite o ingressi è uno dei punti di forza dei microcontroller : leggendo lo stato degli ingressi possiamo introdurre nel programma informazioni derivate dalla situazione di contatti, pulsanti, commutatori, tastiere, fine corsa, encoder e sensori di moltissimi generi, oltre alla possibilità di ricevere segnali da linee seriali, convertitori di dati e periferiche di ogni tipo. Il comando dei pin come uscite renderà, poi, possibile governare i processi controllati, azionando LED, relais, motori, display, ecc.

La configurazione ingresso-uscita di un pin è variabile da programma e, dove è reso necessario dall'applicazione, può essere modificata dinamicamente durante l'esecuzione: non è obbligatorio mantenere la stessa funzione ad un I/O, ma è possibile senza problemi variane la direzione ingresso/uscita dove questo serve a gestire correttamente l'hardware.

Non è neppure un obbligo assegnare la direzione dei pin all'inizio del sorgente: si fa questo comunemente solo perché nella maggior parte delle applicazioni la direzione resta stabile durante l'esecuzione dell'intero programma.

Abbiamo visto come una uscita digitale abbia due livelli, 1 e 0: al primo corrisponde una tensione vicina alla Vdd, all'altro una vicina allo 0 (Vss).

Un "livello logico" valido è la tensione alla quale un segnale viene riconosciuto dal circuito digitale

come 1 o 0. Sappiamo che la logica binaria tratta dati le cui cifre hanno solo due valori, definibili in vari modi:

livello ALTO	high	set	1
livello BASSO	low	clear	0

Quando un livello è 0 (LOW) o 1 (HIGH) rispetto alla tensione di alimentazione?

Per i PIC, che sono costruiti in tecnologia MOSFET, questi livelli dipendono dalla tensione di alimentazione V_{dd} , che può andare da 1.8 a 5.5V, e dal tipo di gate che costituisce il circuito di ingresso: con questo si intende che i circuiti “dietro” ai pin, configurati come ingresso, possono essere di tre generi differenti, a seconda dell’impiego per cui il pin è stato progettato.

Per l' intero range di alimentazione V_{dd} si può usare questa tabella:

Livelli logici dei pin di ingresso digitale			
Tipo	V_{il} (min)	V_{ih} (max)	parametro
TTL	0.15 V_{dd}	0.25 $V_{dd} + 0.8 V$	D030/D040

Quindi, ad esempio, alimentando il micro a 5V, quando portiamo un pin di ingresso ad una tensione inferiore a 0.75V il processore considererà quel pin a livello basso (0). Se applichiamo una tensione superiore a 2.05V il processore considererà quel pin a livello alto (1).

E le **tensioni intermedie** tra questi due valori? Queste **sono da evitare tassativamente**, perché nel tratto compreso tra i due valori limite il circuito elettronico non è in grado di attribuire alla tensione un valore logico valido; si ha una banda di incertezza che non deve apparire mai al pin di ingresso, perché genererebbe risposte casuali (ricordiamo che stiamo parlando di logica digitale, che lavora su valori definiti e non può ammettere altri stati se non 1 e 0).

Per semplificare, possiamo dire che collegando il pin alla V_{dd} abbiamo sicuramente un valore 1 e collegandolo alla V_{ss} abbiamo sicuramente un valore 0. Per il collegamento possiamo usare contatti meccanici (interruttori, pulsanti, commutatori, ecc.) o interruttori elettronici (transistor). L’ importante è che all’ingresso si abbiano solamente valori nella fascia 1 o 0.

Per completezza, va detto che il microcontroller può accettare valori variabili nell’intera gamma tra V_{ss} e V_{dd} , ma esclusivamente attraverso ingressi non digitali, bensì “analogici”, come quelli di comparatori e convertitori AD (Analogico-Digitale). In questi caso, però, dati così raccolti devono venire convertiti in valori binari, definiti da stati 1 e 0.

Vediamo, dunque, di iniziare ad interfacciarci semplicemente con pulsanti, presenti su ogni demoboard o facilmente implementabili sulla breadboard.

La considerazione principale è, dunque, la seguente:

- **un pin di ingresso digitale deve avere sempre applicata una tensione a livello logico 0 o 1 per rendere una lettura corretta.**

Questo si otterrà applicando un **pull-up o un pull-down**.

Il pin di ingresso, per poter essere utilizzato, **non deve rimanere floating**. Con questa parola si

indica una situazione in cui al pin non è connesso nulla che fornisca un riferimento di tensione.

Dal punto di vista della gestione del programma, come abbiamo tratteggiato nelle esercitazioni precedenti, ogni gruppo di pin dispone di tre registri di controllo:

- **PORT**
- **LAT**
- **TRIS**

Per i **PIC 12F1571/2**, la struttura dei registri è questa:

bit	7	6	5	4	3	2	1	0
PORTA	-	-	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
LATA	-	-	LATA5	LATA4	-	LATA2	LATA1	LATA0
TRISA	-	-	TRISA5	TRISA4	-	TRISA2	TRISA1	TRISA0

- Ogni bit corrisponde ad un pin. Siccome il chip dispone di soli 6 pin di I/O, i bit 7 e 6 non sono implementati.
- Non sono implementati anche i bit 3 di **LATA** e **TRISA** perchè **RA3-MCLR** può operare solo come ingresso digitale.
Ricordiamo che i bit non implementati rendono in genere 0 in lettura e qualsiasi dato scritto va perso.
- I registri **PORT**, **LAT** e **TRIS** possono essere modificati in qualsiasi momento durante il funzionamento del programma.

E' fondamentale comprendere la diversa funzione, anche se può non essere immediato.

- **PORT**
 - in lettura, permette di leggere lo stato logico applicato ai pin configurati sia come ingressi che come uscite
 - in scrittura, permette di modificare lo stato logico dei pin configurati come uscite

Quindi,

- se leggo **PORT** mi troverò a leggere nel registro corrispondente il valore reale della tensione applicata ai pin.
- Se scrivo su **PORT** modificandolo, solo ai pin configurati come uscite si applicherà la modifica.

La scrittura di un valore su un bit di PORT (o di LAT) il cui corrispondente pin è configurato come ingresso non ha alcun effetto.

O meglio, ha l'effetto di modificare il latch che sta a monte dei pin e che conserva lo stato imposto in scrittura. Questo stato è trasferito al pin quando è configurato come uscita, altrimenti il dato resta nel latch fino ad una nuova scrittura.

- **LAT** permette
 - in lettura, di leggere il livello logico conservato nei latch
 - in scrittura, di modificare questo contenuto.

Il contenuto di **LAT** è trasferito al pin solo se questo è configurato come uscita. Se è configurato come ingresso, la scrittura del latch non ha effetti sul pin, ma solo sul contenuto del latch stesso.

Se leggo un port ora e trovo applicati certi livelli logici, rileggendolo dopo un certo tempo, se i livelli sono cambiati, il registro riporterà la nuova situazione.

Il latch, invece, è modificato solo dalla scrittura dello stesso. Se lo leggo, leggo non lo stato del pin, ma quanto scritto nel latch stesso l'ultima volta prima della lettura.

Il port sarà uguale al latch solo per quanto riguarda i pin configurati come uscite; quelli configurati come ingressi riporteranno lo stato delle tensioni applicate.

- **TRIS** permette di configurare i pin come ingressi o come uscite
In generale tutti i pin possono assumere funzione di I/O digitale e possono essere configurati come ingressi o come uscite (fa eccezione MCLR che può essere solo ingresso).

I registri PORT, LAT e TRIS possono essere modificati in qualsiasi momento durante il funzionamento del programma.

La regola è questa:

- **TRIS = 1** → pin configurato come ingresso-uscita
- **TRIS = 0** → pin configurato come uscita

Al default la condizione imposta è che tutti i bit di TRIS sono a livello 1, per cui tutti i pin sono configurati come ingressi (condizione di minor consumo).

Per ottenere uscite dovrò modificare il bit corrispondente portandolo a 0.

Al default, invece, il contenuto di **LAT** è casuale, mentre il contenuto di **PORT** riflette le tensioni applicate ai pin.

All'avvio del programma sarà necessario, quindi, inserire le istruzioni necessarie a configurare i pin nel modo voluto, come ingressi o uscite, e ad applicare i livelli logici corretti alle uscite.

Ad ogni reset o mancanza di alimentazione, la situazione dei registri degli I/O si riporta alla condizione di default.

Maggiori dettagli su come funziona un port le trovate [qui](#)^(www) e [qui](#)

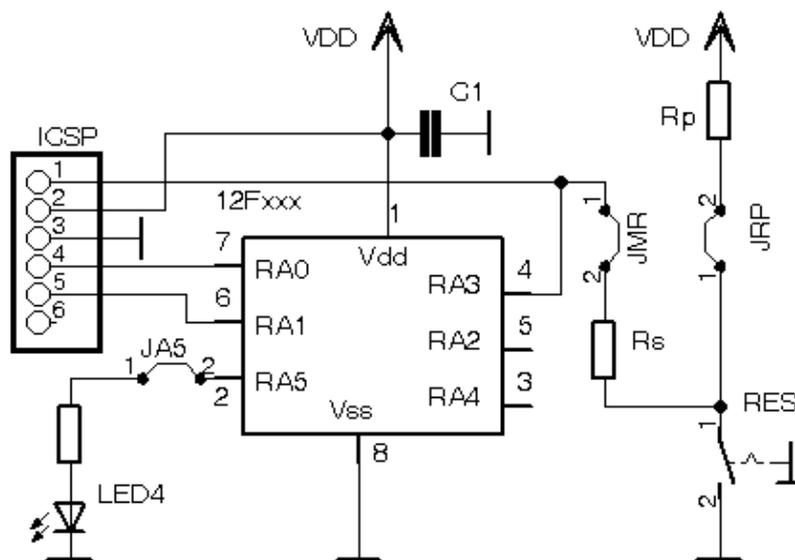
3.2 - Leggere un ingresso.

La possibilità di leggere la situazione di un ingresso è fondamentale per gli scopi per i quali è progettato il microcontroller.

La lettura di un ingresso è, in sostanza, l'identificazione del livello di tensione che è applicato a questo ingresso e che dipenderà da circuiti esterni, come pulsanti, contatti, ma anche linee di comunicazione.

Nel caso di un ingresso digitale, la tensione dovrà essere 1 o 0.

Per ottenere questo, sfruttiamo il circuito che collega il pulsante Reset al pin **RA3** come è realizzato sulla **LPCuB** (www).



Il pulsante **RES** è collegato da un lato alla V_{ss} -Gnd e dall'altro al pin **RA3** attraverso una resistenza **Rs**, mentre è "appeso" alla V_{dd} con il pull-up **Rp**.

Se il pulsante è aperto, al pin **RA3** arriva la tensione **Vdd** attraverso **Rp** e **Rs**; di conseguenza **RA3** è a livello 1. Leggendo lo stato del pin sul registro **PORTA** rileveremo il bit **PORTA3** corrispondente a 1.

Se il pulsante è chiuso, a **RA3** arriva il livello 0 della massa attraverso la **Rs**. Di conseguenza, sul registro **PORTA** leggeremo il bit corrispondente come 0.

Per conoscenza, i valori tipici per **Rp** e **Rs** sono rispettivamente 10K e 1K.

Rs non è strettamente indispensabile: il suo scopo è quello di isolare il pulsante dalla tensione di 12V circa che viene applicata al pin **RA3-Vpp** durante la programmazione del chip ed evitare danni al dispositivo di programmazione se inavvertitamente il pulsante è premuto durante questa operazione e, più in generale, a permettere ai diodi di protezione integrati di limitare la tensione in ingresso nel caso di disturbi elettromagnetici e statici. Questi disturbi possono apparire sulle linee che collegano i dispositivi di ingresso al microcontroller, soprattutto se sono lunghi più di qualche centimetro, ma anche dal contatto di un corpo col pulsante. Queste correnti sono inviate a massa o

al positivo di alimentazione da diodi di protezione del pin, interni al chip.

RA3 nelle esercitazioni precedenti ha effettuato la funzione di **MCLR**.

Però, in questi PIC, **RA3** può essere dedicato ad altre funzioni:

RA3	4	—	—	—	T1G ⁽¹⁾	—	—	—	IOC	Y	MCLR Vpp
-----	---	---	---	---	--------------------	---	---	---	-----	---	-------------

Il pin 4 può assumere la funzione di **MCLR** se nel config iniziale viene attivata questa opzione (**MCLRE = ON**). Se, invece, dichiariamo **MCLRE = OFF**, il pin assume la funzione di **RA3**.

Si tratta, quindi di un I/O digitale, ma con la limitazione che abbiamo indicato prima: **può operare solo come ingresso e non come uscita.**

Le altre funzioni del pin **RA3** elencate nella tabella sopra sono:

- **T1G** - gate del Timer1, funzione assegnabile con il registro APFCON e attiva se viene configurata nel Timer1. Dato che non programiamo questa area, la funzione può essere considerata inesistente
- **IOC** - interrupt on change che va abilitato agendo sui registri IOCAP/IOCAN. Non toccando questi registri, la funzione è, per default, disattivata e possiamo non considerarla
- **Y** – indica la possibilità di attivare un weak pull-up sul pin. Per default, il pull-up integrato è inattivo se il pin non è **MCLR** e va abilitato da programma.
- **MCLR**, ingresso per un reset esterno, va abilitato/disabilitato dal config iniziale. Da notare che, se abilitato, viene abilitata anche una resistenza di pull-up integrata che rende superfluo l'uso di un resistore esterno
- **Vpp** è l'ingresso della tensione di programmazione applicata dal tool di programmazione esterno. È una funzione automatica che non interessa l'esecuzione del programma, ma solo la programmazione del chip

Quindi, in sostanza, al default e senza altre azioni, basterà disabilitare MCLR dal config per poter accedere a **RA3** come ingresso digitale.

Dunque, nel sorgente, la prima considerazione da fare è la disabilitazione della funzione **MCLR** ottenendo l'ingresso **RA3**:

```
// #pragma config statements for PIC12F1571/2
// CONFIG1
// intosc, no clockout, no wdt, no pwrt, mclr, bor, no code prot.
#pragma config FOSC=INTOSC // INTOSC oscillator ECL ECM ECH
#pragma config CLKOUTEN=OFF // no clockout ON
#pragma config WDTE=OFF // no WDT SWDTEN NSLEEP ON
#pragma config MCLR=OFF // no MCLR ON
```

Il resto della configurazione resta inalterato.

Questa nuova configurazione è salvata nel file *config12f1572_1.h* che includeremo nel sorgente.

Va ricordato che, all'accensione, tutti i pin sono configurati come ingressi, quindi **RA3**, liberato dalla funzione **MCLR** passa ad essere un ingresso digitale senza altra operazione.

Quindi l'inizializzazione richiesta per l'I/O sarà la seguente:

```
/****** MAIN PROGRAM *****/  
void main(){  
    // Inizializza I/O  
    LATAbits.LATA5 = 1;    // preset RA5 high  
    TRISAbits.TRISA5 = 0; // RA5 come output  
                          // RA3 è ingresso digitale
```

Per accendere il LED in conseguenza dello stato del pulsante, una via del tutto immediata, ma semplicistica, è quella di portare il livello di **RA5** come quello di **RA3**:

```
// accende il LED in funzione dello stato di RA3  
LATAbits.LATA5 = !PORTAbits.RA3;
```

ovvero se il livello al pin **RA3** è uguale a 0 (pulsante chiuso), poni **RA5=1** (LED acceso) e viceversa.

La scrittura è identica a :

```
LATAbits.LATA5 = ~PORTAbits.RA3;
```

ma il segno ! è prontamente accessibile dalla tastiera italiana, mentre il segno ~ (tilde) non lo è e deve essere richiamato con *ALT+126*.

Da notare, per quanto detto in precedenza sui registri di controllo dei pin, **leggiamo lo stato del pin sul PORTA ed agiamo in uscita sul LATA.**

3.3 - Il debounce.

Abbiamo detto semplicistico perchè non viene preso in considerazione **il problema dei rimbalzi del contatto meccanico**. Praticamente tutti i contatti meccanici, ovvero pulsanti, relè, interruttori, ecc.(esclusi alcuni tipi speciali, ad esempio quelli a bagno di mercurio) sono soggetti a rimbalzi della parte mobile durante la chiusura e l'apertura, il che provoca una serie di stati 1-0 che variano casualmente per un certo tempo, fino alla stabilizzazione della condizione finale.

In presenza di rimbalzi, facendo corrispondere lo stato dell'uscita a quello del contatto in ingresso, si avrebbe in realtà un lampeggio veloce del LED che sarebbe acceso e spento fino al termine di rimbalzi.

Dato che questi rimbalzi solitamente hanno breve durata (qualche decina di ms), nella nostra applicazione la loro presenza non sarebbe avvertita. Però, se abbiamo a che fare con altre situazioni, il trovarsi con l'ingresso che varia inaspettatamente può essere fonte di molti problemi.

Se è possibile, in molti casi, applicare un qualche sistema di debounce hardware, questo può non esserlo in molti altri e, comunque, si tratta di aggiungere componenti esterni.

Ricorriamo allora ad un debounce software.

Una forma molto semplice di debounce software, dove è possibile applicarla, è la seguente:

- *attendo la chiusura del contatto*
- *commuto l'uscita*
- *attendo un tempo pari almeno alla massima durata dei rimbalzi*
- *attendo l'apertura del contatto*
- *commuto l'uscita*
- *attendo un tempo pari almeno alla massima durata dei rimbalzi*

In questo modo ho una risposta immediata all'azionamento del pulsante, ma prima di un'altra azione attendo che i rimbalzi siano esauriti.

Se presumiamo che sia possibile un disturbo sulla linea del pulsante che faccia assumere alla tensione sul pin un valore errato, possiamo modificare leggermente la sequenza:

1. *attendo la chiusura del contatto*
2. *attendo un tempo pari almeno alla massima durata dei rimbalzi*
3. *se il contatto è chiuso, commuto l'uscita*
se è aperto si è trattato di un disturbo e torno al punto 1
4. *attendo l'apertura del contatto*
5. *attendo un tempo pari almeno alla massima durata dei rimbalzi*
6. *se il contatto è aperto, commuto l'uscita*
se è chiuso si è trattato di un disturbo e torno al punto 5

Queste sono semplici sequenze di debounce, mentre esistono algoritmi più complessi dove un tempo di debounce fisso potrebbe non essere applicabile.

Dal punto di vista delle istruzioni:

```

while (1){
    // fino a che il contatto è aperto, attendi
    while (PORTAbits.RA3 == 1) ;
        LATAbits.LATA5 = 1;           // alla chiusura, accendi il LED
        __delay_ms (20);              // ritardo di debounce di 20ms

    // fino a che il contatto è chiuso, attendi
    while (PORTAbits.RA3 == 0) ;
        LATAbits.LATA5 = 0;           // all'apertura, spegni il LED
        __delay_ms (20);              // ritardo di debounce di 20ms
}                                     // loop continuo

```

Abbiamo deciso per un ritardo di debounce di 20ms perchè questo è il tempo che il costruttore dichiara come massimo per quel pulsante; per altri componenti il tempo potrà estendersi anche oltre 100ms.

Osserviamo alcuni particolari:

1. l'azione richiesta è continua e quindi il loop principale **while** è come già visto visto in precedenza
2. l'analisi dello stato del pulsante viene fatta rientrare in un altri loop **while** che sono da leggere come:

fino a che (while) la condizione indicata tra parentesi è vera...

nella condizione il segno di comparazione (uguale a) è == e non = che indica una uguaglianza imposta

3. ricordarsi sempre la chiusura delle parentesi graffe aperte

Il sistema di debounce visto sopra è molto pratico, ma presenta alcuni problemi che possono essere sentiti in applicazioni più complesse della presente.

In sostanza siamo di fronte al fatto che la durata dei rimbalzi non è un dato “scientificamente” esatto; per il costruttore dei contatti, il tempo dichiarato è in genere quello della massima durata dei rimbalzi, ma nella pratica questo tempo, a contatti nuovi, potrebbe essere molto più breve, mentre ha la tendenza ad allungarsi con il numero degli azionamenti. Inoltre, per molti contatti non di marca primaria, il tempo di debounce può non essere indicato e, per sicurezza, si tenderà ad introdurre tempi lunghi.

Ci troviamo di fronte da un lato ad un possibile spreco di tempo in ritardi superiori alla reale durata dei rimbalzi, mentre da un altro lato, riducendo questo tempo, rischiamo di trovarci ancora nell'area dei bounces.

Esistono altri algoritmi di debounce più avanzati che avremo occasione di vedere più avanti.

3.4 - WPU.

Abbiamo già visto come sia possibile sfruttare il pull-up automaticamente inserito quando **RA3** è **MCLR**. Questo pull-up è un elemento integrato e similmente a questo ne sono disponibili altri su tutti i pin. Sono chiamati weak pull-up (WPU).

Il loro uso consente di limitare il numero dei componenti esterni.

Non si tratta di resistori, ma di MOSFET con una elevata resistenza interna, che possono essere applicati o esclusi da programma.

La loro abilitazione dipende dalle necessità hardware dell'applicazione e sono gestiti in modo tale da poter essere abilitati e disabilitati per ogni singolo pin, attraverso un registro denominato **WPUx**.

Per 12F1572 abbiamo solo un **PORTA** e quindi ci sarà un **WPUA**.

bit	7	6	5	4	3	2	1	0
label	-	-	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0

Al default all'accensione (POR) i weak pull-up **NON sono abilitati** (bit **WPUAx=1**), ma non sono attivi in quanto esiste un ulteriore bit di controllo che permette di attivarli e disattivarli TUTTI globalmente. Questo bit, chiamato **nWPUEN** si trova nel registro **OPTION_REG**

bit	7	6	5	4	3	2	1	0
label	WPUEN	INTEDG	TMR0CS	TMR0SE	PSA	PS2	PS1	PS0

[Maggiori informazioni su OPTION_REG.](#)

Per rendere attiva una selezione effettuata con **WPUA**, occorre portare a 0 il bit **nWPUEN** ($\overline{\text{WPUEN}}$)
Per default questo bit è a 1 e di conseguenza **WPUA** non ha effetto e al POR nessun pull-up è inserito. (La funzione di altri bit di **OPTION_REG** verrà trattata in altre esercitazioni).

Quindi, l'attivazione di un pull-up integrato richiede due azioni:

- disattivare i wpu non necessari
- abilitare il controllo globale dei wpu

```
WPUA = 0b00001000;           // abilita WPU solo su RA3
OPTION_REGbits.nWPUEN = 0;   // abilita wpu globale
```

La label **nWPUEN** indica che il bit è negato $\overline{\text{WPUEN}}$ (condizione attiva per bit= 0 e viceversa) .

Se utilizziamo il pin RA3 come ingresso digitale può essere necessario, come nel nostro caso, abilitare la relativa wpu, il che evita l'aggiunta di una resistenza esterna di pull-up del pulsante. Se il pull-up integrato per la funzione MCLR è automatico, quello per RA3 non lo è e va attivato da programma.

Dal punto di vista dell'hardware sulla [LPCuB](#) ([www](#)) basterà togliere il **jumper Rp** per escludere la resistenza di pull-up esterna. Nel programma abilitiamo il wpu.

Se non lo abilitiamo ed eliminiamo il pull-up esterno, noteremo che il programma non funziona o si comporta casualmente; questo dipende dal fatto che il pin di ingresso, non polarizzato, assume uno stato ad alta impedenza che non è identificato come valido dal circuito logico e, nel contempo, è causa di rilevamenti di stato casuali, dipendenti dai disturbi elettromagnetici e statici catturati dalle piste del circuito collegate al pin.

[Altre informazioni sui pull-up integrati.](#)

es3C.c

```

/*****
*-----
*
*   Titolo           :   C Enhanced - Lesson 2
*                   :   Un LED collegato a RA5 è acceso se il pulsante
*                   :   collegato a RA3 è chiuso.
*                   :   WPU su RA3.
*   Data            :   01-05-2011
*   Modificato il   :
*   Versione        :   V0.0
*   Ref. hardware   :
*   Autore          :   afg
*
*-----
*****
*   Impiego pin :
*   -----
*       12F1571/2 @ 8 pin
*
*           |_____|
*           Vdd -|1   8|- Vss
*           RA5 -|2   7|- RA0
*           RA4 -|3   6|- RA1
*           RA3/MCLR -|4 5|- RA2
*           |_____|
*
*   Vdd                1: ++
*   RA5/OSC1/CLKIN     2: Out LED
*   RA4/OSC2/CLKOUT    3: In  nc
*   RA3/!MCLR/VPP     4: In  RA3
*   RA2                5: In  nc
*   RA1/ICSPCLK       6: In  nc
*   RA0/ICSPDAT       7: In  nc
*   Vss               8: --
*
*
*   RA5/T1CKI/PWM1/CW1G1A/[%RX-DT]/IOC
*   RA4/AN3/C1IN-/[T1G]/[PWM2]/[CW1G1B]/[*TX-CK]/IOC
*   RA3/[T1G]/IOC/MCLR
*   RA2/AN2/C1OUT/T0CKI/PWM3/CWG1A-!CW!G!FLT]/INT/IOC
*   RA1/AN1/Vref+/C1IN0-/PWM1/%RX-DT/IOC/ICSPCLK
*   RA0/AN0/DAC1OUT/C1IN+/PWM2/CWG1B/%TX-CK/IOC/ICSPDAT
*
*   [] rilocabili con APFCON
*   % solo 12F1572
*
*   Note: - INTOSC default @ 500kHz
*
*****/

// intosc, no clockout, no wdt, no pwrt, no mclr, bor, no code prot.
// no WRT, no pll, no stack error, bor low, no bor lp, no LVP
#include "C:/Corso_C/MyIncludes/conf1572_1.h"

#include <xc.h>
#define _XTAL__FREQ 50000

```

```
/****** MAIN PROGRAM *****/
void main(void) {
    // inizializza I/O
    LATAbits.LATA5 = 0;           // preset RA5 high
    TRISA = 0b11011111;         // RA5 come output
    WPUA = 0b00001000;         // abilita WPU su RA3
    OPTION_REGbits.nWPUEN = 0;  // abilita WPU globale

    // main loop
    while (1) {
        // fino a che il contatto è aperto, attendi
        while (PORTAbits.RA3 == 1) ;
            LATAbits.LATA5 = 1;           // alla chiusura, accendi il LED
            __delay_ms (20);             // ritardo di debounce di 20ms

        // fino a che il contatto è chiuso, attendi
        while (PORTAbits.RA3 == 0) ;
            LATAbits.LATA5 = 0;         // all'apertura, spegni il LED
            __delay_ms (20);           // ritardo di debounce di 20ms
    }
}
}
```

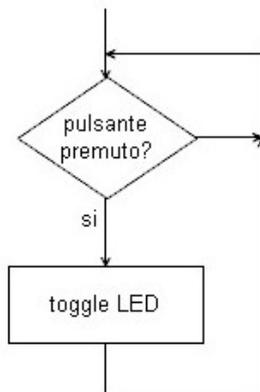
Questa versione è fornita come progetto completo (MPLABX v.4.13 o successivi, XC8 v.1.34 o successivi e **PIC12F1572**).

I file del progetto sono previsti per stare in una cartella *C:\Corso_Cles3C*.

3.5 – Toggle LED.

Vogliamo ottenere un funzionamento bistabile: ad una pressione del pulsante RES, il LED si accende, alla successiva si spegne

Lo schema elettrico e la situazione sulla scheda di sviluppo sono le stesse.

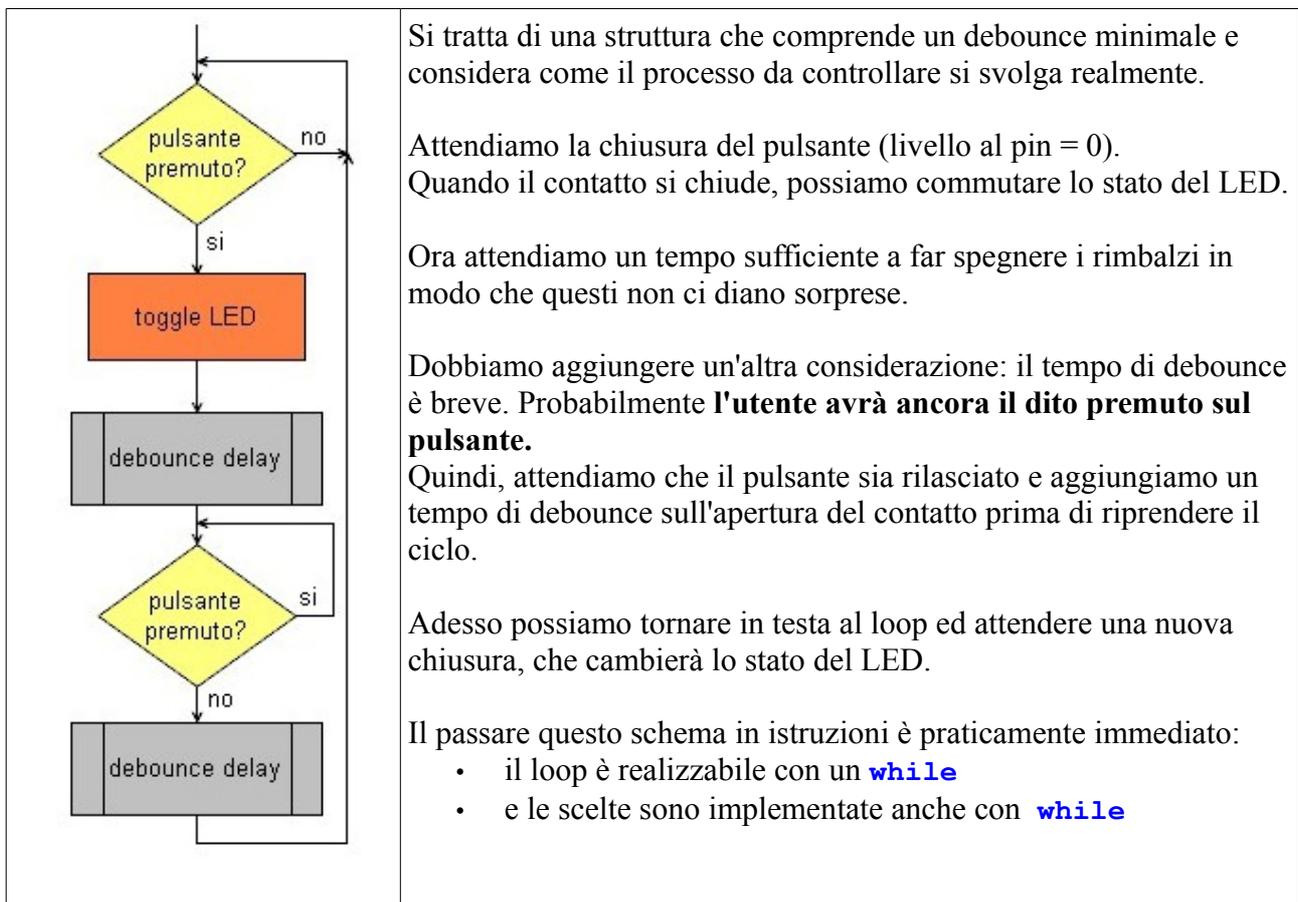


E' opportuno stendere un diagramma di flusso che ci consente di passare con facilità dall'idea alle istruzioni.

Questo a lato può essere un inizio, ma siamo ben lontani da quanto in effetti è necessario.

Nell'affrontare un problema occorre tenere presenti tutti i suoi termini: qui, non abbiamo considerato per nulla il problema dei rimbalzi del contatto del pulsante, né quello del comportamento dell'utente.

Allora, meglio questa versione più efficace:



L'uso di **while** per il loop primario lo abbiamo sufficientemente trattato nelle esercitazioni precedenti.

Utilizzarlo anche per i blocchi di scelta, dove parrebbe naturale inserire **if**, è dovuto al fatto che, se la scelta non è effettuata (condizione FALSO), il programma deve rimanere in attesa della condizione VERO, situazione svolta meglio da **while**.

La stesura di un diagramma di flusso è essenziale per individuare le vie più efficaci per realizzare quanto voluto.

Di conseguenza:

```
#define Pulsante PORTAbits.PORTA3
#define Led      LATAbits.LATA5

/***** MAIN PROGRAM *****/
void main(void) {

    // Inizializza I/O
    LATAbits.LATA5 = 1;           // preset latch RA5 =1
    TRISA = 0b11011111;         // RA5 come output
    WPUA = 0b00001000;          // abilita WPU su RA3
    OPTION_REGbits.nWPUEN = 0;   // abilita WPU
    OSCCON = 0;                  // clock a 31kHz

    while (1) {                  // loop continuo
        while (Pulsante==1);     // attesa pressione pulsante
        Led=~Led;                 // toggle LED
        __delay_ms(30);           // debounce 30ms

        while (Pulsante==0);     // attesa pulsante aperto
        __delay_ms(30);           // debounce 30ms
    }
}
```

Osserviamo che abbiamo definito una label **Pulsante** sostitutiva per **RA3** dove è applicato il pulsante e **Led** per **RA5** dove è applicato il LED. Questo rende più comprensibile il sorgente.

Come solito, **leggiamo su PORT** e **scriviamo su LAT**.

Abbiamo **ridotto anche il clock**: in una applicazione del genere **non c'è nessuna necessità di prestazione** e anche il clock minimo disponibile dall'oscillatore interno è adeguato.

La riduzione della frequenza, per contro, **riduce il consumo di corrente**.

Occorrerà, però, adeguare la definizione di **_XTAL_FREQ** al valore di frequenza utilizzato in modo che la macro del delay generi il ritardo voluto..

es3C_1.c

```

/*****
*-----
*
*   Titolo           :   C Enhanced - Lesson 2
*                   :   Un LED è acceso e spento dal pulsante su RA3.
*   Data             :   01-05-2011
*   Modificato il    :
*   Versione         :   V0.0
*   Ref. hardware    :
*   Autore           :   afg
*
*-----
*****
* Impiego pin :
* -----
*           12F1571/2 @ 8 pin
*
*           |_____|
*           Vdd -|1   8|- Vss
*           RA5 -|2   7|- RA0
*           RA4 -|3   6|- RA1
*           RA3/MCLR -|4 5|- RA2
*           |_____|
*
*   Vdd                1: ++
*   RA5/OSC1/CLKIN     2: Out LED
*   RA4/OSC2/CLKOUT    3: In  nc
*   RA3!/MCLR/VPP      4: In  RA3
*   RA2                5: In  nc
*   RA1/ICSPCLK        6: In  nc
*   RA0/ICSPDAT        7: In  nc
*   Vss                8: --
*
*
* RA5/T1CKI/PWM1/CW1G1A/[%RX-DT]/IOC
* RA4/AN3/C1IN-/[T1G]/[PWM2]/[CW1G1B]/[*TX-CK]/IOC
* RA3/[T1G]/IOC/MCLR
* RA2/AN2/C1OUT/T0CKI/PWM3/CWG1A-!CW!G!FLT]/INT/IOC
* RA1/AN1/Vref+/C1IN0-/PWM1/%RX-DT/IOC/ICSPCLK
* RA0/AN0/DAC1OUT/C1IN+/PWM2/CWG1B/%TX-CK/IOC/ICSPDAT
*
* [] rilocabili con APFCON
* % solo 12F1572
*
* Note: - INTOSC default @ 500kHz
*
*****/

// intosc, no clockout, no wdt, no pwrt, no mclr, bor, no code prot.
// no WRT, no pll, no stack error, bor low, no bor lp, no LVP
#include "C:/Corso_C/MyIncludes/conf1572_1.h"

#include <xc.h>

#define _XTAL_FREQ 31000 // clock 31kHz

```

```
#define Pulsante PORTAbits.PORTA3
#define Led LATAbits.LATA5

/***** MAIN PROGRAM *****/
void main(void) {
    // Inizializza I/O
    LATAbits.LATA5 = 1;           // preset latch RA5 =1
    TRISA = 0b11011111;         // RA5 come output
    WPUA = 0b00001000;          // abilita WPU su RA3
    OPTION_REGbits.nWPUEN = 0;  // abilita WPU
    OSCCON = 0;                  // clock a 31kHz

    while (1) {                  // loop continuo
        while (Pulsante==1);     // attesa pulsante premuto
        Led=~Led;                // toggle LED
        __delay_ms(30);          // debounce 30ms
        while (Pulsante==0);     // attesa pulsante rilasciato
        __delay_ms(30);          // debounce 30ms
    }
}
```

Questa versione è fornita come progetto completo (MPLABX v.4.13 o successivi, XC8 v.1.34 o successivi e PIC12F1572).

I file del progetto sono previsti per stare in una cartella *C:\Corso_C\esIC_1*.

3.6 – Alcune osservazioni.

Compilando questa esercitazione possiamo rilevare alcuni punti interessanti.

Un primo punto riguarda le parentesi graffe, principalmente dove la funzione non esegue nulla. Le righe

```
while (Pulsante==1){ // attesa pulsante premuto
}
```

possono essere sostituite con

```
while (Pulsante==1); // attesa pulsante premuto
```

dove la coppia di parentesi è sostituita da ; (null), meno chiaro, ma più compatto. Si tratta di uno dei tanti shortcut offerti dal C.

Nella lista della finestra di Output ci ritroviamo due warnings

```
:: warning: (1273) Omniscient Code Generation not available in Free mode
../../es3C_1.c:70: warning: (752) conversion to shorter data type
```

Il primo, già visto, riguarda il fatto che la versione del compilatore usata è quella free e quindi priva dell'ottimizzatore Omniscient Code; in sostanza, più un bug del compilatore che un problema dell'utente. Non viene evidenziato nel testo.

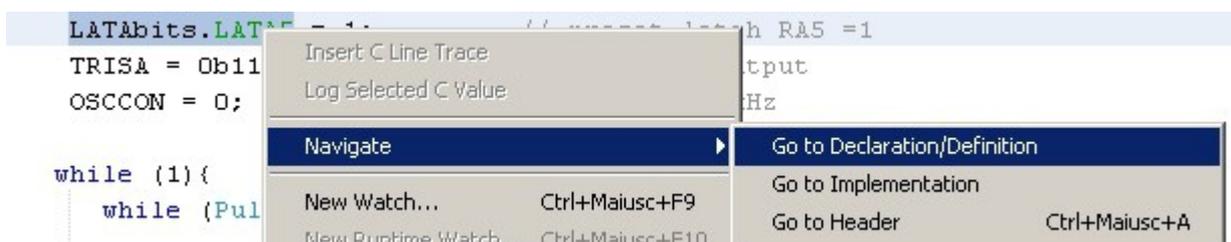
Il secondo avviso, meno chiaro, ma evidenziato, riguarda la riga `Led=~Led; // toggle LED` che troviamo evidenziata con un triangolino giallo:



Questo dipende dalla definizione `#define Led LATAbits.LATA5`

`LATAbits` è una struttura che troviamo pre definita nell'header `pic12f1572.h`; possiamo accedere a questa definizione direttamente dal testo del sorgente:

1. evidenziamo nel main la label `LATAbits.LATA5`
2. click destro: si apre un menù a tendina dove scegliamo *Navigate*
3. si apre un altro sotto menù da cui scegliamo *Goto Declaration/Definition* che ci porta nel file `pic12f1572.h` al punto in cui è originata la definizione



Nella riga `Led=~Led;` utilizziamo la label e il compilatore indica che l'ha declassata automaticamente ad un tipo più breve, per risparmiare risorse.

Entrambi i warning non sono pericolosi e non determinano problemi nell'eseguibile. In particolare, se non si desiderano i warning 752, è possibile disabilitarli nelle proprietà del progetto:

Project Properties > XC8 global options > Additional options

e inserire **--MSGDISABLE=752**

La selezione è valida solo per il progetto in corso.

Anche selezionando un diverso livello di warning, ad esempio 0 (quello di default del progetto è -3), questo genere di indicazioni viene soppressa.

In ogni caso, non è consigliabile eliminare tutti i warning, ma piuttosto comprenderne le cause ed eliminarle dove possibile.

Sempre a riguardo delle parentesi graffe, questo sorgente ci permette di osservare bene il meccanismo dell'IDE che aiuta l'utente ad individuare le coppie.

Se puntiamo col mouse una parentesi, questa viene evidenziata in giallo, mentre viene pure evidenziata quella che il compilatore ha inteso come seconda della coppia

```

61 void main(void) {
62     // Inizializza I/O
63     LATAbits.LATA5 = 1;           // preset latch RA5 =1
64     TRISA = 0b11011111;         // RA5 come output
65     OSCCON = 0;                 // clock a 31kHz
66
67     while (1) {                 // loop continuo
68         while (Pulsante==1) {   // attesa pulsante premuto
69             }
70         Led=~Led;               // toggle LED
71         __delay_ms(30);         // debounce 30ms      }
72
73         while (Pulsante==0) {   // attesa pulsante rilasciato
74             }
75         __delay_ms(30);         // debounce 30ms
76     }
77 }

```

Nello stesso tempo, sul lato sinistro, si evidenzia un indicatore che comprende tutte le righe all'interno delle parentesi. Sopra osserviamo l'indicatore in grigio, fisso, del blocco logico della funzione **main** e l'indicatore in blu del blocco del primo **while**.

Questo automatismo è di grande aiuto nell'identificare e correggere errori nell'uso delle parentesi.

La cosa funziona anche con le parentesi rotonde, anche se, solitamente, la coppia contiene un'area estesa ad una sola riga.

3.7 – E se non avessi un Enhanced, ma solo un Midrange ?

In queste pagine vogliamo parlare essenzialmente degli Enhanced Midrange. Questo non toglie, che, come abbiamo visto in precedenza, è possibile riportare gli esempi per qualsiasi altro PIC a 8bit senza particolari problemi.

Qui, stiamo ancora utilizzando gli I/O digitali, che hanno lo stesso comportamento: si devono solo tenere presente le differenze tra i vari chip. In Midrange e Baseline occorrerà considerare che:

- **LAT** non esiste ed occorre utilizzare **PORT** (o **GPIO**) con la necessità di curare il problema **RMW**
- è possibile che in alcuni chip **si debba escludere la funzione analogica dei pin usati**
- è possibile che **si debba disabilitare il comparatore** per accedere alla funzione di I/O digitale di alcuni pin

Inoltre, Midrange e Baseline non dispongono di oscillatore interno elaborato come quello degli Enhanced e non si potranno effettuare manovre di cambio del clock.

Se state usando PIC così antichi da avere l'obbligo di cablare un oscillatore esterno, valutate la possibilità di investire un paio di euro (o meno) in un chip più attuale.

Dunque se volete passare il sorgente al vostro vecchio PIC, seguite quanto detto e gli esempi precedenti.

In tal senso questo è l'ultima “partecipazione” dei vecchi PIC a questo corso.

Schede informative dettagliate. **Esercitazione 3**

Qui di seguito alcune pagine di informazioni più dettagliate sui argomenti trattati nella prima esercitazione. Che possono essere estratte e conservate separatamente.

- [I/O digitali- come funzionano](#)
- [OPTION_REG](#)
- [Usare i pull up integrati](#)

Files allegati

Sono allegati i seguenti files/cartelle:

- Cartella *es3C* : progetti e sorgenti

I/O digitali – come funzionano.

Abbiamo visto finora come è possibile comandare uscite digitali, ma, i pin possono funzionare anche come ingressi digitali.

La possibilità di definire i pin come uscite o ingressi è uno dei punti di forza dei microcontroller : leggendo lo stato degli ingressi possiamo introdurre nel programma informazioni derivate dalla situazione di contatti, pulsanti, commutatori, tastiere, fine corsa, encoder e sensori di moltissimi generi, oltre alla possibilità di ricevere segnali da linee seriali, convertitori di dati e periferiche di ogni tipo. Il comando dei pin come uscite renderà, poi, possibile governare i processi controllati, azionando LED, relais, motori, display, ecc.

La configurazione ingresso-uscita di un pin è variabile da programma e, dove è reso necessario dall'applicazione, può essere modificata dinamicamente durante l'esecuzione: non è obbligatorio mantenere la stessa funzione ad un I/O, ma è possibile senza problemi variane la direzione ingresso/uscita dove questo serve a gestire correttamente l'hardware.

Non è neppure un obbligo assegnare la direzione dei pin all'inizio del sorgente: si fa questo comunemente solo perché nella maggior parte delle applicazioni la direzione resta stabile durante l'esecuzione dell'intero programma.

Abbiamo visto come una uscita digitale abbia due livelli, 1 e 0: al primo corrisponde una tensione vicina alla Vdd, all'altro una vicina allo 0 (Vss).

Ci si può porre il seguente problema: quale tensione devo applicare all'ingresso perché il processore legga un valore 1 piuttosto che 0?

In linea di massima possiamo dire che la Vdd e la Vss sono comunque i riferimenti:

Ingresso collegato alla	Livello logico
Vdd	1
Vss	0

Da notare che la tensione applicata ad un pin non può superare in positivo la Vdd o andare al di sotto della Vss, pena la distruzione dei componenti integrati collegati al pin.

In effetti, però, non occorre una connessione diretta alla Vdd o alla Vss, ma basta che la tensione al pin di ingresso sia entro i limiti di riconoscimento dei circuiti logici.

Cosa si intende con questo?

Un "livello logico" è la tensione alla quale un segnale viene riconosciuto dal circuito digitale come 1 o 0. Sappiamo che la logica binaria tratta dati le cui cifre hanno solo due valori, definibili in vari modi:

livello ALTO	high	set	1
livello BASSO	low	clear	0

Quando un livello è 0 (LOW) o 1 (HIGH) rispetto alla tensione di alimentazione?



I produttori di chip generalmente fanno riferimento allo standard più comune che è **TTL** (*Transistor-Transistor Logic*).

Questo stabilisce un valore massimo per la tensione che viene interpretata come 0 ed un minimo per quella corrispondente a 1.

Data la tensione di alimentazione TTL che è 5V, si formano tre bande:

- Da 0 a 0.8V tutte le tensioni applicate all'ingresso sono interpretate come 0 logico
- Da 2 a 5V tutte le tensioni applicate all'ingresso sono interpretate come 1 logico
- **Le tensioni comprese tra 0.8V e 2V non devono mai essere inviate all'ingresso digitale TTL** in quanto non darebbero origine a incertezza nell'attribuzione ad uno a all'altro livello.

Per i PIC, che sono costruiti in tecnologia MOSFET, questi livelli dipendono dalla tensione di alimentazione V_{dd} , che può andare da 1.8 a 5.5V, e dal tipo di gate che costituisce il circuito di ingresso: con questo si intende che i circuiti "dietro" ai pin, configurati come ingresso, possono essere di tre generi differenti, a seconda dell'impiego per cui il pin è stato progettato.

Per una alimentazione "TTL" a 5V abbiamo:

Livelli logici dei pin di ingresso digitale per $4.5V < V_{dd} < 5.5V$			
Tipo	V_{il} (min)	V_{ih} (max)	parametro
TTL	0.8 V	2.0 V	D030A/D040A
Schmitt trigger	0.3 V_{dd}	0.7 V_{dd}	D034/D044
SMBUS	0.6 V	1.4 V	D034A/D044A

Per una alimentazione inferiore, i parametri si spostano leggermente. Per l'intero range di alimentazione V_{dd} si può usare questa tabella:

Livelli logici dei pin di ingresso digitale			
Tipo	V_{il} (min)	V_{ih} (max)	parametro
TTL	0.15 V_{dd}	0.25 $V_{dd} + 0.8 V$	D030/D040

Quindi, ad esempio, alimentando il micro a 5V, quando portiamo un pin di ingresso ad una tensione inferiore a 0.75V il processore considererà quel pin a livello basso (0). Se applichiamo una tensione superiore a 2.05V il processore considererà quel pin a livello alto (1).

E le **tensioni intermedie** tra questi due valori? Queste **sono da evitare tassativamente**, perché nel tratto compreso tra i due valori limite il circuito elettronico non è in grado di attribuire alla tensione un valore logico valido; si ha una banda di incertezza che non deve apparire mai al pin di ingresso, perché genererebbe risposte casuali (ricordiamo che stiamo parlando di logica digitale, che lavora su valori definiti e non può ammettere altri stati se non 1 e 0).

Per semplificare, possiamo dire che collegando il pin alla Vdd abbiamo sicuramente un valore 1 e collegandolo alla Vss abbiamo sicuramente un valore 0. Per il collegamento possiamo usare contatti meccanici (interruttori, pulsanti, commutatori, ecc.) o interruttori elettronici (transistor). L'importante è che all'ingresso si abbiano solamente valori nella fascia 1 o 0.

Per completezza, va detto che il microcontroller può accettare valori variabili nell'intera gamma tra Vss e Vdd, ma esclusivamente attraverso ingressi non digitali, bensì "analogici", come quelli di comparatori e convertitori AD (Analogico-Digitale). In questi caso, però, dati così raccolti devono venire convertiti in valori binari, definiti da stati 1 e 0.

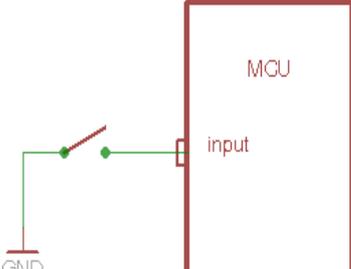
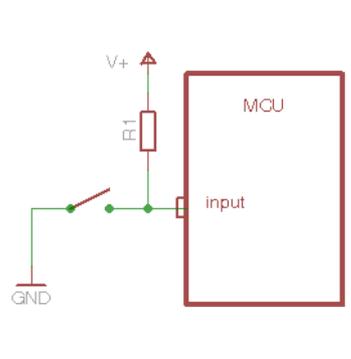
Vediamo, dunque, di iniziare ad interfacciarci semplicemente con pulsanti, presenti su ogni demoboard o facilmente implementabili sulla breadboard.

La considerazione principale è, dunque, la seguente:

- **un pin di ingresso digitale deve avere sempre applicata una tensione a livello logico 0 o 1 per rendere una lettura corretta.**

Questo si otterrà applicando un **pull-up o un pull-down**.

Il pin di ingresso, per poter essere utilizzato, **non deve rimanere floating**. Con questa parola si indica una situazione in cui al pin non è connesso nulla che fornisca un riferimento di tensione. Quindi, ad esempio, applicando un contatto all'ingresso, la seguente situazione è fortemente errata:

	<p>Sicuramente quando il contatto è chiuso, il pin è a livello 0, ma quando il contatto è aperto, il pin è floating e la tensione che appare dipende più che altro dai disturbi presenti nell'intorno.</p> <p>La lettura dello stato del pin col contatto chiuso renderà 0, ma quella col contatto aperto sarà casuale e quindi non utilizzabile praticamente.</p>
	<p>La correzione da introdurre sarà l'applicazione di un pull-up sul pin oppure attivare i pull-up integrati, dove siano disponibili.</p> <p>In questo caso, con il contatto aperto, il pin non è più floating, ma riceve una tensione a livello logico 1 attraverso la resistenza di pull-up.</p> <p>Ora, correttamente, la lettura dello stato del pin col contatto chiuso renderà 0 e quella col contatto aperto renderà 1.</p>

Dal punto di vista della gestione del programma, come abbiamo tratteggiato nelle esercitazioni

precedenti, ogni gruppo di pin dispone di tre registri di controllo:

- **PORT**
- **LAT**
- **TRIS**

E' fondamentale comprendere la diversa funzione, anche se può non essere immediato.

- **PORT**
 - in lettura, permette di leggere lo stato logico applicato ai pin configurati sia come ingressi che come uscite
 - in scrittura, permette di modificare lo stato logico dei pin configurati come uscite

Quindi,

- se leggo **PORT** mi troverò a leggere nel registro corrispondente il valore reale della tensione applicata ai pin.

Se scrivo su **PORT** modificandolo, solo ai pin configurati come uscite si applicherà la modifica. Il diagramma seguente riporta la struttura di un port tipico degli Enhanced Midrange.

- **LAT** permette
 - in lettura, di leggere il livello logico conservato nei latch
 - in scrittura, di modificare questo contenuto.

Se leggo un port ora e trovo applicati certi livelli logici, rileggendolo dopo un certo tempo, se i livelli sono cambiati, il registro riporterà la nuova situazione.

Il latch, invece, è modificato solo dalla scrittura dello stesso. Se lo leggo, leggo non lo stato del pin, ma quanto scritto nel latch stesso l'ultima volta prima della lettura.

Il port sarà uguale al latch solo per quanto riguarda i pin configurati come uscite; quelli configurati come ingressi riporteranno lo stato delle tensioni applicate.

- **TRIS** permette di configurare i pin come ingressi o come uscite
In generale tutti i pin possono assumere funzione di I/O digitale e possono essere configurati come ingressi o come uscite (fa eccezione MCLR che può essere solo ingresso).

La regola è questa:

- **TRIS = 1** → pin configurato come ingresso-uscita
- **TRIS = 0** → pin configurato come uscita

Al default la condizione imposta è che tutti i bit di TRIS sono a livello 1, per cui tutti i pin sono configurati come ingressi (condizione di minor consumo).

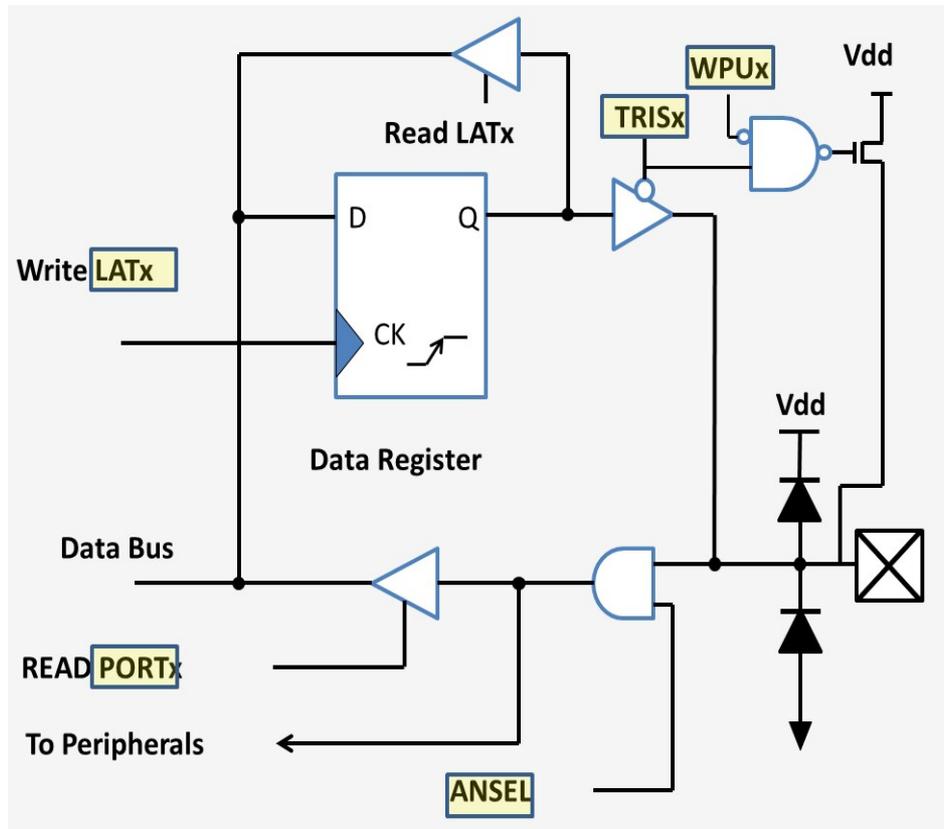
Per ottenere uscite dovrò modificare il bit corrispondente portandolo a 0.

Al default, invece, il contenuto di **LAT** è casuale, mentre il contenuto di **PORT** riflette le tensioni applicate ai pin.

All'avvio del programma sarà necessario, quindi, inserire le istruzioni necessarie a configurare i pin nel modo voluto, come ingressi o uscite, e ad applicare i livelli logici corretti alle uscite.

Ad ogni reset o mancanza di alimentazione, la situazione dei registri degli I/O si riporta alla condizione di default.

Vediamo la situazione graficamente attraverso un diagramma:



Vediamo come funziona il port in scrittura.

Il cuore dell'azione è il registro dati, che è un latch tipo D.

Quando scriviamo un dato su **LATx** o su **PORTx**, questo dato è presente sul bus dati e viene caricato nel latch dal clock generato dall'istruzione di scrittura.

Come si nota, l'uscita del latch è condizionata da un gate pilotato dal corrispondente bit del registro **TRISx**: se il bit è a 0, il gate è aperto e mette in contatto l'uscita **Q** con il pin.

Se il bit del **TRISx** è a 1, il gate è chiuso e il dato non viene passato al pin (ma resta nel latch fino ad una nuova scrittura).

Quindi, è possibile caricare i latch senza che questi dati vadano trasferiti ai pin configurati come ingressi. Il trasferimento del livello logico del latch avviene solo se il pin è configurato come uscita. Per i PIC dove i registri **LAT** non sono accessibili (Baseline e Midrange) la scrittura sul registro **PORT** ha lo stesso effetto, ma con il rischio del problema RMW.

In lettura, sul bus dati viene trasferito lo stato del pin o del latch, attraverso gate clockati dall'istruzione di lettura.

- Se leggo **LAT**, trasferisco sul bus non lo stato del pin, ma quello dell'uscita **Q** del latch (che, come abbiamo detto, può essere non collegato al pin).
- Se leggo **PORT**, trasferisco sul bus il livello di tensione presente sul pin. Se questo livello rientra nei limiti dell'interfaccia, sarà letto come 1 o 0.

Quindi, qui sta la differenza tra **PORT** e **LAT**:

- **LAT** legge l'uscita del latch in ogni caso
- **PORT** legge la tensione presente sul pin

Ovviamente, se il pin è configurato come uscita, la lettura di **LAT** e **PORT** sarà n uguale valore, a meno di condizioni di RMW.

Osserviamo che il pin dispone di altre opzioni:

- la possibilità di abilitare un weak pull-up integrato che in molti casi evita l'aggiunta di un pull-up esterno (registro **WPU**). I weak pull-up non sono resistori, ma MOSFET con resistenza elevata. Negli Enhanced sono inseribili singolarmente sui pin.
- la possibilità di dirigere il segnale in ingresso non al bus dati, ma ad una periferica analogica, come ADC o comparatori (registro **ANSEL**)

Per i **PIC 12F1571/2**, la struttura dei registri che interessano i pin digitali è questa

bit	7	6	5	4	3	2	1	0
PORTA	-	-	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
LATA	-	-	LATA5	LATA4	-	LATA2	LATA1	LATA0
TRISA	-	-	TRISA5	TRISA4	-	TRISA2	TRISA1	TRISA0

- Ogni bit corrisponde ad un pin. Siccome il chip dispone di soli 6 pin di I/O, i bit 7 e 6 non sono implementati.
- Non sono anche implementati i bit 3 di **LATA** e **TRISA** perchè **RA3-MCLR** può operare solo come ingresso digitale.
Ricordiamo che i bit non implementati rendono in genere 0 in lettura e qualsiasi dato scritto va perso.
- I registri **PORT**, **LAT** e **TRIS** possono essere modificati in qualsiasi momento durante il funzionamento del programma.

La scrittura di un valore su un bit di PORT il cui corrispondente pin è configurato come ingresso non ha alcun effetto. O meglio, ha l'effetto di modificare il latch che sta a monte dei pin e che conserva lo stato imposto in scrittura. Questo stato è trasferito al pin quando è configurato come uscita, altrimenti il dato resta nel latch fino ad una nuova scrittura.

Dallo schema precedente possiamo notare che il pin è protetto da due diodi che impediscono alla tensione applicata di superare la V_{dd} o, in negativo, la V_{ss}.

Questi diodi hanno un evidente limite di corrente, il cui superamento porterebbe alla loro distruzione. Per evitare questo genere di problemi, è molto consigliato inserire una resistenza in serie al pin (genericamente tra 100 e 1000 ohm) allo scopo di limitare la corrente nel caso di entrata in funzione di uno dei diodi.

Questa resistenza serie non ha influenza sul funzionamento del pin.

La sua inserzione è indispensabile se il collegamento tra pin e sensori di ingresso assume una lunghezza sensibile, dato che i collegamenti possono comportarsi come antenne per i campi elettromagnetici presenti nell'ambiente.

Per quanto riguarda i pin configurati come uscite:



I buffer di uscita dei pin non dispongono di alcun sistema per limitare la corrente assorbita o resa. Quindi, una situazione di corto circuito o di eccessiva corrente, danneggeranno il buffer.

In generale, la corrente gestibile da un pin è tipicamente 25mA, anche se esistono Enhanced con pin speciali che possono trattare correnti di 100mA.

Maggiori informazioni sulla corrente massima nei pin qui ([www](#)).

E' utile sapere che negli Enhanced i pin possono essere configurati da programma

- in **ingresso** come **TTL** o **ST**, quindi con diverse soglie di riconoscimento della tensione 1/0 di ingresso.
- come **uscite**, possono assumere sia la struttura classica push-pull, sia diventare **open drain**.

Inoltre, è possibile ridurre la pendenza delle transizioni tra i livelli logici per ridurre le interferenze ad alta frequenza generate quando si commutano carichi con una certa corrente.

Alcuni registri consentono di applicare le opzioni ai singoli pin.

Questo offre una flessibilità che non è presente in Baseline e Midrange, adattando i pin alle più diverse situazioni.

Maggiori informazioni le trovate sul foglio dati alla sezione che tratta gli I/O.

Molto importante è tenere ben presente che:



I buffer di uscita dei pin, anche configurati come open drain, non possono gestire tensioni diverse da quella di alimentazione.

Quindi, per usare un pin per comandare un carico su una tensione maggiore della Vdd, ad esempio 12 o 24V occorrerà interporre un buffer adeguato.

Maggiori informazioni su circuiti di interfaccia le trovate qui ([www](#)).

Precauzioni per non danneggiare i PIC le trovate qui ([www](#)).

Dei pin fa parte anche il meccanismo di **Interrupt On Change (IOC)** che consente di generare un interruzione al cambio di stato di un pin.

Negli Enhanced il sistema è particolarmente sofisticato, in quanto è possibile abilitare per ogni pin la risposta sia ad una variazione 1-0 (falling edge) sia ad una variazione 0-1 (rising edge) o a entrambe.

OPTION_REG

Il registro **OPTION** è un elemento importante per la gestione dei weak pull-up, dell'ingresso INT e del Timer0

Si presenta composto da 8 bit attivi:

bit	7	6	5	4	3	2	1	0
label	WPUEN	INTEDG	TMR0CS	TMR0SE	PSA	PS2	PS1	PS0

- **WPUEN** (**nWPUEN**) consente di abilitare contemporaneamente tutti i wake pull-up che sono stati selezionati con i registri WPU relativi ai singoli port. E' attivo a livello 1. Per default al POR è a livello 1, ovvero tutti i wpu sono disabilitati
- **INTEDG** questo bit ha lo scopo di selezionare il fronte di risposta all'interrupt esterno applicato al pin **INT** (1= rising edge, 0= falling edge)
- **TMR0CS** questo bit permette di selezionare la sorgente del clock del Timer0 (1= clock dall'esterno sul pin **TOCKI**, 0=clock da Fosc/4)
- **TMR0SE** permette di selezionare il fronte di commutazione per il segnale esterno di clock del Timer0 (1=falling edge, 0= rising edge).
- **PSA** questo bit decide l'assegnazione del prescaler al Timer0 (1= no prescaler, 0=prescaler)
- **PS<2:0>** attraverso questi tre bit si seleziona il coefficiente di divisione del prescaler, da 1:2 a 1:256

A differenza di Baseline e Midrange, gli Enhanced non hanno il prescaler del Timer0 condiviso con il WDT; ognuna delle due periferiche dispone di un prescaler proprio.

Ogni bit di **OPTION_REG** è leggibile e scrivibile. Per default al POR tutti i bit sono a 1, quindi la situazione è:

- Wpu generale disabilitato
- **INT** per fronte di salita
- Clock del Timer0 dal pin **TOCKI**
- Fronte di discesa del clock esterno
- Prescaler non assegnato
- Prescaler 1:256

Qualsiasi altra selezione va operata nel programma.

Usare i pull-up integrati.

Abbiamo già visto come sia possibile sfruttare il pull-up automaticamente inserito quando **RA3** è **MCLR**. Vediamo ora come sfruttare la presenza dei weak pull-up in generale.

Se torniamo a considerare l'impiego dei pin di ingresso digitale, nella comune configurazione con un pulsante verso la massa, è richiesto che, a contatto aperto, l'ingresso sia comunque polarizzato verso la Vdd (livello logico 1). Questo si ottiene con una resistenza di pull-up applicata esternamente, il che vuol dire aggiungere un componente.

Questo non è necessario nei chip che integrano i pull-up (denominati weak pull-up perchè presentano un valore di resistenza ohmica abbastanza alto, con lo scopo di non consumare una corrente elevata quando il contatto è chiuso): queste resistenze sono interne al chip e possono essere abilitate da programma. L'uso dei **weak pull-up integrati** nel chip consente di limitare il numero dei componenti esterni.

Da un punto di vista pratico, i weak pull-up non sono veri e propri resistori, ma MOSFET P ad elevata resistenza che possono essere inseriti tra il pin, come ingresso, e la Vdd. In effetti si tratta di sorgenti di corrente, con un valore medio di 100/120uA (param. D070 del foglio dati), il che corrisponde ad un valore resistivo di 30/40kohm. Sono adeguati per molte applicazioni, anche se in alcuni casi è necessario utilizzare resistenze esterne di valore minore (linee open collector, I²C, ecc.) o dove valori resistivi bassi sono richiesti per la reiezione del rumore.

Quando vogliamo utilizzare i weak pull-up, dobbiamo abilitarli da programma.

Negli Enhanced Midrange sono organizzati in modo tale da poter essere abilitati e disabilitati per ogni singolo pin, attraverso un registro denominato **WPUx**, corrispondente ad un port.

Ad esempio, in 12F1572, dato che esiste solo il **PORTA**, ci sarà un **WPUA**.

bit	7	6	5	4	3	2	1	0
label	-	-	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0

I bit 7 e 6 non sono implementati in quanto il chip dispone di soli 6 pin utili.

I bit da 5 a 0 corrispondono ad ogni singolo pin.

Per default, questi bit sono a 1, ovvero tutti i weak pull-up sono abilitati. Per disabilitarli, basta portare i bit a 0.

Però, **al default all'accensione (POR) i weak pull-up NON sono attivi, in quanto esiste un ulteriore bit di controllo che permette di attivarli e disattivarli TUTTI globalmente.** Questo bit, chiamato **nWPUEN** si trova nel registro **OPTION_REG**

bit	7	6	5	4	3	2	1	0
label	WPUEN	INTEDG	TMR0CS	TMR0SE	PSA	PS2	PS1	PS0

Per rendere attiva la selezione effettuata con **WPUA**, occorre portare a 0 il bit **nWPUEN** (**WPUEN**)
Per default questo bit è a 1 e di conseguenza **WPUA** non ha effetto e al POR nessun pull-up è inserito. (La funzione di altri bit di **OPTION_REG** verrà trattata in altre esercitazioni).

Per riassumere:

- **i weak pull-up sono inseribili solamente se il pin è configurato come ingresso.**
Se il pin è configurato come uscita non c'è possibilità di attivare il pull-up.
- Su pin **RA3**, quando questo è configurato come **MCLR** il pull-up è attivato automaticamente; questo può rendere inutile l'aggiunta di una resistenza esterna.
Se il pin è configurato come input, il pull-up dipende dal bit **WPUA3** del registro **WPUA**.
- Nessun weak pull-up è presente (escluso quello automatico su **RA3=MCLR**) a meno che da programma non si azzeri il bit **nWPUEN** del registro **OPTION_REG**.
Il bit **nWPUEN** programmato a 1 (default) disabilita TUTTI i wpu contemporaneamente.
- Azzerando il bit **nWPUEN** si abilitano tutti i wpu che hanno il relativo bit nei registri **WPUx** uguale a 1. Tutti i pin che hanno il relativo bit nei registri **WPUx** uguale a 0 non disporranno del wpu.
- **Per i pin configurati come uscita il wpu sarà disabilitato qualsiasi sia il valore nel registro WPUx.**

Quindi:

- per abilitare un wpu occorre portare a 1 il bit del relativo registro **WPUx** (default) e poi il bit **nWPUEN**.
- Per disabilitare un wpu occorre portare a 0 il bit del relativo registro **WPUx**.



IMPORTANTE:

**I wpu vanno disabilitati dagli ingressi analogici.
In caso contrario, la lettura del segnale analogico non sarà corretta.**

La presenza dei wpu è automatica se questi sono abilitati e il pin è configurato come ingresso. Sono automaticamente esclusi dai pin configurati come uscite; però non c'è un automatismo che disconnetta i wpu se il pin è un ingresso a cui è associata una funzione analogica. La presenza del wpu altera il valore analogico. Quindi occorre ricordarsi di disabilitare i pull-up dove si stia utilizzando un qualsiasi ingresso analogico (ADC, comparatori, opamp, ecc).