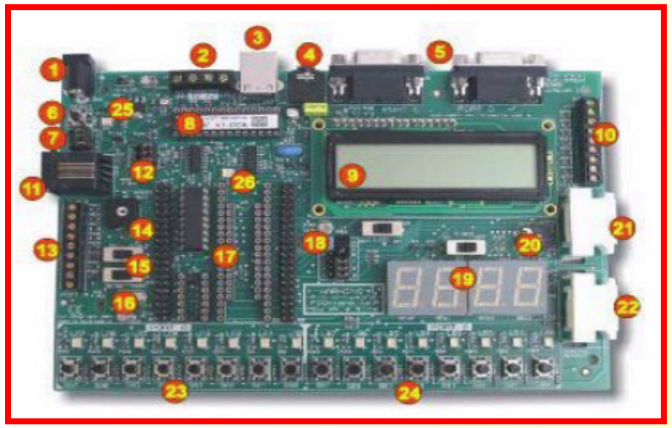
Università degli studi di Cassino

Corso di Laurea in
Ingegneria della Produzione Industriale



Corso di Informatica Applicata

Lezione 7

Ing. Saverio De Vito
e-mail: saverio.devito@portici.enea.it
Tel.: +39 081 7723364

Interfacciamento Sensori ed Attuatori

L' interfacciamento di sensori ed attuatori comporta l' analisi e la soluzione di numero di problematiche:

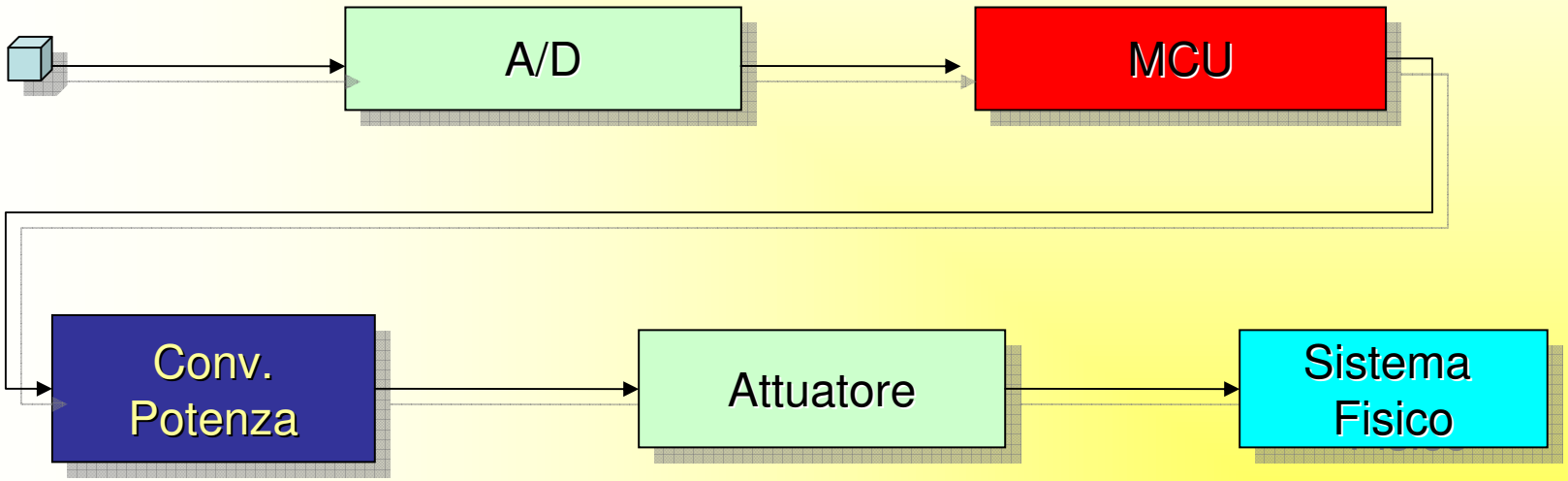
- Tipologia di sensore (Analogico, Digitale)
- Caratteristiche statistiche del sensore/attuatore (affidabilità, precisione, specificità, modello fisico dell' attuatore etc.)
- Metodologia di interfacciamento
- Campionamento
- Caratteristiche del mezzo di interfacciamento
- Etc.

Interfacciamento di Sensori e Attuatori

- Ad esempio l' interfacciamento di un sensore analogico richiede la presenza nel canale di acquisizione di un dispositivo per la conversione analogico/digitale….
- L' interfacciamento di un attuatore di potenza richiede la presenza sul canale di attuazione di un dispositivo di conversione in potenza….

Interfacciamento di Sensori e Attuatori

- Semplificando:



Interfacciamento di Sensori e Attuatori

Analizzeremo la struttura di alcuni semplici snippet per l' interfacciamento di sensori ed attuatori.

In particolare analizzeremo l' interfacciamento di un sensore analogico per illuminotecnica e di un display a 7 segmenti.

Interfacciamento di Sensori e Attuatori

In primis abbiamo bisogno di un PIC con capacità di conversione analogica digitale ad esempio il PIC16F874 o il PIC 16F88.

- Convertitore sulla porta A
- Sample and Hold
- SAR (Successive Approximation Register)
- Conversion in Sleep Mode (Interrupt/Interferenze)

Interfacciamento di Sensori e Attuatori

```
unsigned char ADCON0@0x1f;
unsigned char ADCON1@0x9f;

unsigned char ADCRESL@0x9e;
unsigned char ADCRESH@0x1e;

void setup_hardware ( void )
{
   /* set all of PORTB for output   */
   set_tris_b ( 0x00 ) ;

   /* select A0 for analogue input  */
   /* and all other bits for        */
   /* digital I/O                   */
   /* right justified value         */
   ADCON1 = 0x8E;
   /* bits 1-3 of PORTA for output  */
   /* using bit 0 for analogue      */
   /* input                         */
   set_tris_a ( 0xf1 ) ;
}
```

Set up di registri come variabili (estensione non-standard del C2C)

Setup dell' Hardware

Interfacciamento di Sensori e Attuatori

```
void main ( void )
{
   unsigned char h, l ;
   setup_hardware () ;
   while (1)
   {
      /* enable the converter and start    */
      /* a conversion                      */
      ADCON0 = 0x05;
      /*  spin while the conversion runs   */
      while ( ADCON0 & 0x04 );
      /* can read the high nibble as it    */
      /* in bank 0                         */
      h = ADCRESH;
      /* now need to flip to bank 1 to     */
      /* read the low byte.                */
      /* flip to the other bank            */
      asm bsf STATUS, RP0
      /* load the byte into the W register */
      asm movf _ADCRESL, W
      /* flip back home                    */
      asm bcf STATUS, RP0
      /* store the W register in l         */
      asm movwf _l_main
      output_port_b ( l );
      output_port_a ( h * 2 ) ;
   }
}
```

Routine di Acquisizione

(potremmo convertrla in una funzione read_adc0)

Display Bar

```
#define ONE_LED 114
#define TWO_LEDS 227
#define THREE_LEDS 342
#define FOUR_LEDS 456
#define FIVE_LEDS 570
#define SIX_LEDS 684
#define SEVEN_LEDS 798
#define EIGHT_LEDS 912

void display_bar ( int value )
{
    if ( value < ONE_LED ) {
        output_port_b ( 0 ) ;
        return;
    }
    if ( value < TWO_LEDS ) {
        output_port_b ( 1 ) ;
        return;
    }
    if ( value < THREE_LEDS ) {
        output_port_b ( 3 ) ;
        return;
    }
    if ( value < FOUR_LEDS ) {
        output_port_b ( 7 ) ;
        return;
    }
    if ( value < FIVE_LEDS ) {
        output_port_b ( 15 ) ;
        return;
    }
    if ( value < SIX_LEDS ) {
        output_port_b ( 31 ) ;
        return;
    }
    if ( value < SEVEN_LEDS ) {
        output_port_b ( 63 ) ;
        return;
    }
    if ( value < EIGHT_LEDS ) {
        output_port_b ( 127 ) ;
        return;
    }
    output_port_b ( 255 ) ;
}
```

Accensione a Soglia

```
#define THRESHOLD 300
void main ( void )
{    int v;
setup_hardware () ;
while (1)
 {
        v = read_adc0();
     if ( v > THRESHOLD ) {
         output_port_b ( 255 ) ;        }
     else {
         output_port_b ( 0 ) ;          }
 }
}
```

Approfondimenti & Richiami

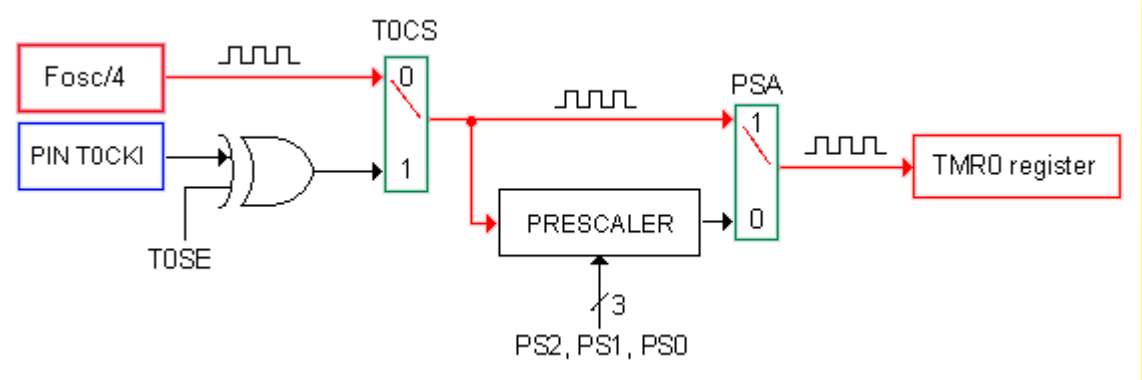
- Controllo ed uso del Timer0
- Controllo del Display LED a 4 Cifre e 7 segmenti
- Interrupts in Assembler
- Debouncing

Controllare il TIMER con prescaler

Il registro TMR0 agisce come contatore attivato dal clock di sistema. Viene incrementato regolarmente con frequenza dipendente dal clock e dal prescaler. Il passaggio per lo 0 causa il lancio di un interrupt che se opportunamente servito può essere utile per problemi di timing, I/O (visualizzazione,) etc.

Il prescaler agisce da divisore di frequenza programmabile e di fatto controlla la frequenza di generazione dell' interrupt legato a TMR0.

Utilizzando il registro OPTION è possibile controllare l' inserimento del prescaler e la sorgente del clock (Esterno, Interno)



Controllare il TIMER con il prescaler

Il valore del prescaler ci permette di ridurre la frequenza di incremento che in modo base è pari alla frequenza di clock diviso 4 (ogni 4 cicli di clock il TMR0 viene incrementato). Ad esmpio in modo base, se partiamo da una fclock pari a 4Mhz, otteniamo una frequenza di incremento TMR0 pari a 1 Mhz.

I Bit PS0, PS1, PS2 del registro OPTION determinano il valore della ulteriore divisione in frequenza operata dal pre-scaler.

PS2	PS1	PS0	Divisore	Frequenza in uscita al prescaler (Hz)
0	0	0	2	500.000
0	0	1	4	250.000
0	1	0	8	125.000
0	1	1	16	62.500
1	0	0	32	31.250
1	0	1	64	15.625
1	1	0	128	7.813
1	1	1	256	3.906

Controllare il TIMER con il Prescaler

I prescaler va assegnato ad uno dei due registri timer presenti sul PIC target, TMR0 o Watch Dog Timer. Noi ci soffermeremo sul TMR0, ignorando il WDT.

Ponendo il bit TOCS del registro OPTION a 0 selezioniamo il Clock del PIC, mentre con il bit PSA posto a 0 assegnamo il prescaler al TMR0.

Per ottenere ad esmpio un ritardo pari a circa un secondo possiamo porre il prescaler a 32, per ottenere una frequenza di 31250/256 Hz = 122.07.

Contando 122 passaggi per lo 0 possiamo ottenere un ritardo di circa 1.0006 secondi.

Per essere + precisi potremmo andare a porre a 6 il TMR0 ogni volta che effettua il round up. Avremo così 250 cicli anzichè 256 e quindi un ritardo di 1 secondo.

Gestione Interrupt in Assembler

- Nella scorsa lezione avevamo visto come il PIC, in corrispondenza di un interrupt saltava, cambiando PCL in una speciale locazione di memoria, rappresentante un vettore di interrupt di dimensioni unitarie.
- Questa locazione di memoria è la locazione 4 esadecimale della memoria programma.
- Da questa locazione parte la routine di gestione dell' interrupt del PIC target.

Gestione Interrupt in Assembler

- E' conveniente effettuare i soliti passi tra i quali :
  - Salvare il contenuto di eventuali registri sottoposti a cambiamento
  - Identificare il tipo di interrupt
  - Effettuare il clear del bit interrupt corrispondente in INTCON
  - Saltare in opportune locazioni di memoria dove effettivamente si implementa la routine di servizio terminante con un RETFIE

Gestione Interrupt in Assembler

- Esempio Interrupt Handling:

```
ORG     04H
    ;*****************************************************************
    ; Interrupt handler
    ;*****************************************************************
     ;Toc, Toc, Chi  è? Determinazione della tipologia
     btfsc   INTCON,T0IF
     goto    IntT0IF
     btfsc   INTCON,RBIF
     goto    IntRBIF
     ;Terminazione, Reset degli interrupts
End_ih
     bcf     INTCON,T0IF
     bcf     INTCON,RBIF
     ;Ritorna al programma principale
     retfie
```

Salto all' opportuno segmento di codice di servizio dal quale si ritornerà in END_ih

Int Reset

Ritorno

Gestione Interrupt in Assembler

Esempio
DBLINT.ASM

Controllare il Display

Il display presente sulla scheda di sviluppo è dotato di 4 cifre a 7 segmenti.

Il numero di bit necessari al loro controllo è maggiore di quello disponibile con i nostri PIC.

Si ricorre ad un “trucco”:

In realtà ogni unità del display è connessa alla porta B in parallelo, mentre i bit della porta A selezionano l’ unità che si vuole controllare.

La visualizzazione di un numero a 4 cifre, si ottiene mediante visualizzazione in rapida successione delle 4 cifre su unità differenti sfruttando la persistenza del display e della retina umana.

Ogni numero può essere visualizzato quindi presentando un opportuno pattern sulla porta B e assicurando la selezione di una particolare unità a 7 segmenti del display a disposizione.

Controllare il Display

```
void display ( unsigned char digit, unsigned char POs )
{ /* turn on the required LED unit*/
  output_port_a ( enable [POs] ) ;
  /* set the pattern on the LED   */
output_port_b ( patterns [digit] );
}
```

Problema del ritardo : selezione -> presentazione -> selezione

Debouncing

Lo switching di un interruttore meccanico causa in generale piccole oscillazioni che possono generare multiple transizioni di stato in programmi del tipo appena sviluppato. Lo stato finale dipende dal numero di transizioni (pari o dispari) che il codice è in grado di rilevare, in pratica lo stato finale si randomizza. Una soluzione è continuare a leggere lo stato fisico dell' interruttore (nel nostro caso connesso ad un bit di porta I/O) fin quando non si stabilizza, ossia fino a quando non otteniamo un numero congruo di letture coerenti.