

Esame di Stato 2006/2007
I.T.I.S. G.Silva

Davide Dal Frà
Classe V BInf

Argomento

Da Analogico a Digitale:
Conversione A/D ed Interfacciamento con un PC

Materie

Elettronica - Informatica

Sommario:

Che cos'è un dato analogico	Pag.	3
La conversione A/D	Pag.	3-4
Interfacciamento con il PC	Pag.	5
La porta parallela.....	Pag.	6
Gestione I/O con la porta parallela.....	Pag.	9
Parte elettronica di Conversione/Interfacc.	Pag.	10
Schema elettrico del Circuito.....	Pag.	11
Realizzazione del PCB	Pag.	11
Funzionamento del circuito	Pag.	13
Convertitore A/D con PIC 16F818.....	Pag.	14
Listato in "C" del PIC16F818	Pag.	15
Datasheet dei componenti.....	Pag.	16
La Libreria JNPOUT32 e relativa DLL.....	Pag.	22
Modifiche alla libreria:La classe ACQInterface...	Pag.	22
Codice dell'applicazione	Pag.	23
Spiegazione del codice/funzioni relativa	Pag.	24
Software sviluppato in java per l'acquisizione	Pag.	28
Foto di un acquisizione.....	Pag.	29
Screenshot Applicativo Java	Pag.	30
Sensori applicati agli ingressi	Pag.	31
Sensore di temperatura con AD590	Pag.	31
Schema elettrico e PCB	Pag.	32
Spiegazione del circuito	Pag.	35
Sensore di Umidità con TSG813 e spiegazione	Pag.	36
Schema elettrico e PCB	Pag.	37
Bibliografia	Pag.	41

Che cos'è un dato Analogico?

In natura ogni grandezza che sia velocità, temperatura, posizione etc..., è presente in forma analogica. Tali grandezze, velocità (Km/h), la temperatura (°C,°F), non sono in forma elettrica e quindi non sono interpretabili da un sistema a Microprocessore (che sia esso un µControllore o un computer).

La conversione A/D

Una grandezza fisica (temperature, velocità , etc...) viene trasformata in una grandezza elettrica mediante trasduttori. La grandezza elettrica ottenuta può essere una variazione *Tensione* , una variazione *Corrente* ,una variazione *Capacità*, etc... Per far “capire” al sistema a microprocessore queste grandezze è necessaria un'altra conversione, quella da Analogico a Digitale (A/D). La grandezza elettrica viene convertita in un insieme di bit (interpretabili dal computer) attraverso un componente chiamato A/D Converter.

Esistono diversi tipi di A/D Converter che convertono il dato per:

- Approssimazioni Successive
- Gradinata
- Integrazione
- Parallelo (Flash)

In qualsiasi caso di A/D converter utilizzato è necessario condizionare il segnale in uscita dal trasduttore poichè gli A/D converter hanno un range min-max da rispettare, usualmente 0÷5 V o 0 ÷10 V.

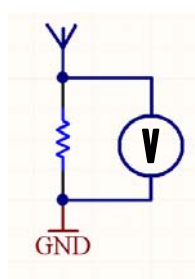
Per condizionare il segnale in uscita da un trasduttore si utilizzano nella maggior parte dei casi degli A.O. opportunamente configurati.

Nel caso un sensore risponda in corrente è necessario convertire questa corrente in uscita dal sensore in tensione. Vi sono due modi per fare questa conversione:

1. Far scorrere la corrente in uscita dal sensore in una resistenza R a noi nota e prelevare la caduta di tensione ai suoi capi.
2. Utilizzare un Convertitore Corrente/Tensione mediante l'utilizzo di un A.O.

Propongo le configurazioni dei due punti descritti sopra:

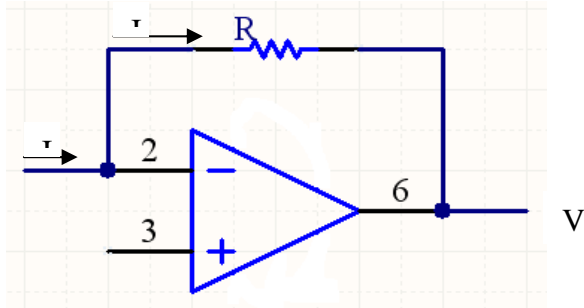
1) **I(trasduttore)**



Dove la tensione V sarà:

$$V = R \times I$$

2) Convertitore I/V



Una volta configurato il circuito di condizionamento del segnale, possiamo applicare la sua uscita ad un convertitore A/D il quale ci fornirà il relativo valore di tensione in binario.

Esempio:

- Se il convertitore A/D è un convertitore a 8 bit con range di tensioni in ingresso comprese tra 0 ÷ 5V avremo :
 - 0 V → 00000000b
 - 5 V → 11111111b

La conversione non sarà precisa in quanto legata alla risoluzione dell' ADC. Il valore minimo apprezzabile sarà:

$$Q = \frac{V_{FS}}{2^{nBit}}$$

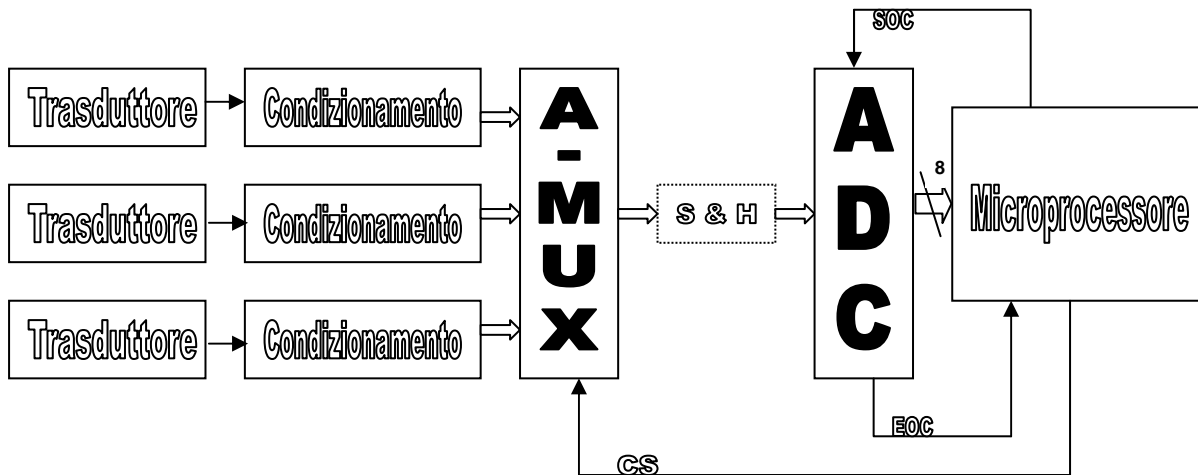
Dove **Q** è il valore minimo apprezzabile , V_{FS} è il valore di fondo scala, e nBit è il numero di bit utilizzati dal convertitore A/D.

Quindi se si utilizzeranno A/D converter con più bit di risoluzione, si avrà una conversione del dato analogico più precisa e fedele.

Interfacciamento con il PC

Una volta convertito il segnale analogico in digitale per mezzo del convertitore A/D si prevede la sua lettura tramite un sistema a microprocessore.

Di seguito viene riportato lo schema a blocchi di una catena di acquisizione dati:



Legenda:

CS : Channel Select (indica il canale selezionato per la conversione)

EOC: EndOfConversion(è un segnale generato dall'adc che indica al microprocessore che ha finito una conversione e che è pronto per la successiva)

SOC: Start Of Conversion (inizio conversione)

Per Interfacciare con un sistema a microprocessore il segnale convertito e permettere la sua acquisizione dal sistema stesso, è necessario scegliere la porta di comunicazione tra ADC e microprocessore.

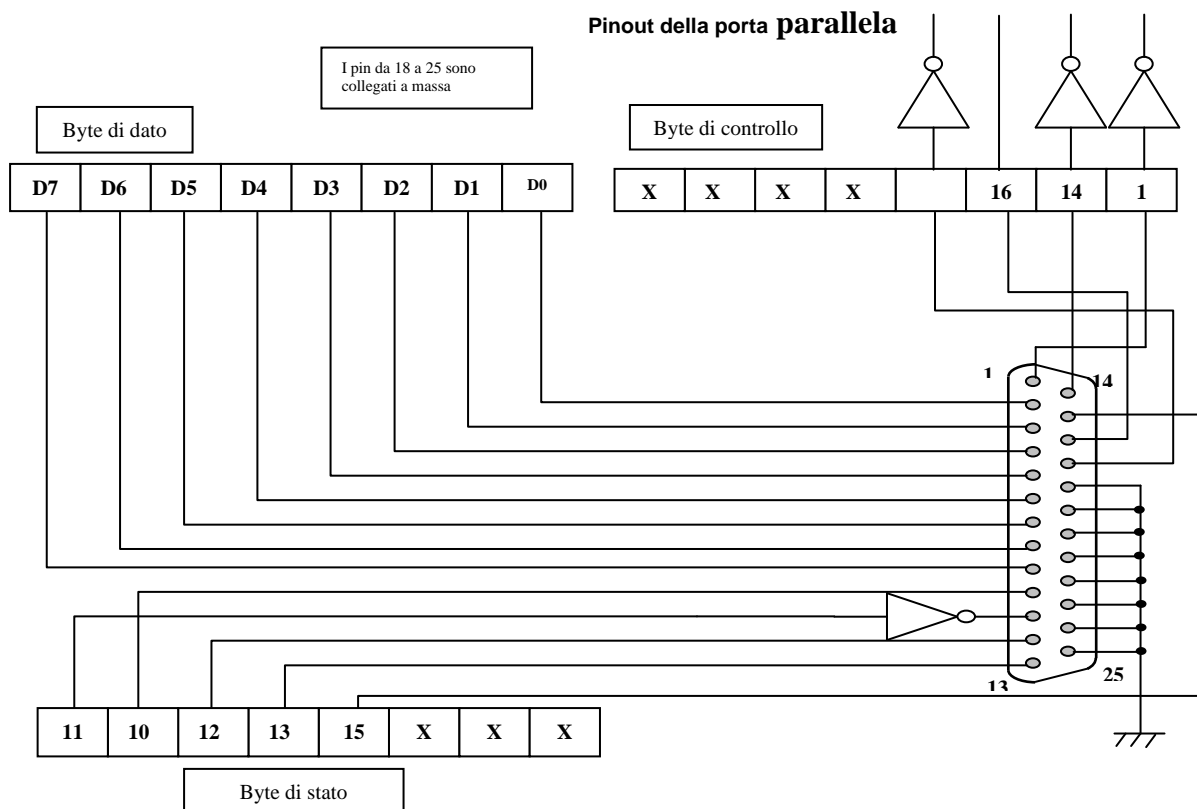
Visto che per comodità di utilizzo quasi tutti gli ADC hanno uscita in Parallelo, ho scelto, nel mio caso, di utilizzare la porta Parallela unidirezionale dei PC.

E' anche possibile utilizzare altre porte di I/O dei PC , quali la porta seriale o usb, previo adattamento del segnale con appositi integrati tipo: MAX232,FT232 detti anche traslatori di livello,o anche utilizzare gli slots di espansione dei pc.

LA PORTA PARALLELA DEL PC

L'interfaccia parallela del calcolatore, nella sua forma più utilizzata, è la periferica più semplice che si possa immaginare. Non ci riferiamo, qui, ai nuovi standard, quelli con nomi tipo EPP, ma al protocollo originale della parallela, che è supportato da tutti i PC, dall'8088 in poi. In pratica, sul connettore a 25 piedini si trovano direttamente i segnali corrispondenti ad alcuni bit di I/O, in logica TTL (con segnali a 0V e 5V). Con "direttamente" si intende che i bit che vengono scritti dal processore sulle porte di output appaiono sui piedini del connettore, e i bit della porta di ingresso vengono letti dal segnale di tensione sul connettore. Il connettore parallelo porta 12 bit di output e 5 bit di input, più 8 segnali di terra. L'interfaccia software si compone quindi di due porte di output ed una porta di input. Per rendere le cose un pò più complicate, alcuni bit subiscono una inversione tra quello che appare sul connettore e quello che viene visto dal processore. E' forse il caso di ricordare le caratteristiche dei segnali elettrici TTL utilizzati. TTL (Transistor-Transistor-Logic) è una famiglia logica molto utilizzata, almeno in passato, che presenta specifiche di ingresso/uscita non simmetriche: mentre un'uscita bassa (0V) può assorbire una corrente significativa, un'uscita alta (5V nominali) non è in grado di erogare più di un paio di mA di corrente, e risulta spesso a tensione molto più bassa di 5V (il minimo assicurato è 2.7 V). Allo stesso modo, per abbassare un ingresso occorre assorbire tanta corrente, mentre per alzarlo basta una resistenza di (1÷5)K Ω verso 5V. Le porte parallele più recenti forniscono prestazioni elettriche migliori, ma la cosa non è affatto garantita.

Ogni porta parallela del sistema viene vista tramite tre indirizzi di I/O consecutivi: "base", "base+1", "base+2". Ad ogni porta parallela è anche associata un'interruzione. I valori di "base" più utilizzati ed i relativi numeri di interruzione sono 0378 (IRQ 7), 0278 (IRQ 2) e 03BC (IRQ 5). Il registro "base+1" è un registro di input, e viene usato per leggere i 5 segnali di ingresso del connettore, "base" e "base+2", invece, sono registri di output, e quando vengono letti restituiscono l'ultimo valore scritto (cioè lo stato attuale dei segnali al connettore). L'effettiva mappatura tra i bit nei registri e i piedini del connettore è spiegata meglio con una figura che a parole (fig.1).



PC Pin N°	I/O	Funzione	Indirizzo	Posizione	Descrizione
1	O	-STROBE	037A	-bit 0	Sincronizza il trasferimento
2	O	D0	0378	bit 0	Dato trasferimento (LSB)
3	O	D1	0378	bit 1	Dato da trasferire
4	O	D2	0378	bit 2	Dato da trasferire
5	O	D3	0378	bit 3	Dato da trasferire
6	O	D4	0378	bit 4	Dato da trasferire
7	O	D5	0378	bit 5	Dato da trasferire
8	O	D6	0378	bit 6	Dato da trasferire
9	O	D7	0378	bit 7	Dato da trasferire (MSB)
10	I	ACK	0379	bit 6	Conferma ricezione
11	I	-BUSY	0379	-bit 7	Stampante occupata
12	I	PE	0379	bit 5	Fine carta
13	I	SLCT	0379	bit 4	Stampante selezionata (ON_LINE)
14	O	-AUTOFD	037A	-bit 1	Avanzamento riga automatica
15	I	ERROR	0379	bit 3	Errore hardware
16	O	INIT	037A	bit 2	Inizializza la stampante
17	O	-SLCT IN	037A	-bit 3	Seleziona la stampante
18÷ 25		GND			

Possiamo suddividere i pin della parallela in 3 gruppi:

1. **BIT DI DATI**: Sono gli otto bit di dato (utilizzati nelle stampanti per inviare il carattere da stampare), sono allocati all'indirizzo **0378H** (relativamente alla porta LPT1). I pin relativi partono dal pin 2 fino al pin 9 come da tabella sottostante:

2	3	4	5	6	7	8	9
D	D	D	D	D	D	D	D
0	1	2	3	4	5	6	7

2. **BIT DI STATO**: Sono i bit di input della porta parallela. Il registro che si legge in totale offre 8 bit, ma i bit settabili sono 5. Tali bit sono i seguenti :

- Il pin *ACK* corrispondente al pin **10** della porta parallela
- Il pin *BUSY* corrispondente al pin **11** della porta parallela
- Il pin *PE* (Paper End) corrispondente al pin **12**
- Il pin *SLCT* (Select) corrispondente al pin **13**
- Il pin *ERROR* corrispondente al pin **15** della porta parallela

Quindi il registro dei bit di stato risulterà così configurato:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
PIN 11	PIN 10	PIN 12	PIN 13	PIN 15	Np*	Np*	Np*
<i>BUSY</i>	<i>ACK</i>	<i>PE</i>	<i>SLCT</i>	<i>ERROR</i>	/	/	/

*Np=Non Presente

3. **BIT DI CONTROLLO**: Sono dei bit utilizzati soprattutto nella gestione delle stampanti. Vengono utilizzati per inizializzare la stampante, per l'avanzamento riga automatico ed altro. Sono i pin:

- *STROBE* (**negato**) corrispondente al pin **1** della porta parallela
- *AUTOFD* (**negato**) corrispondente al pin **14** della porta parallela
- *INIT* corrispondente al pin **16** della porta parallela
- *SLCTIN* (**negato**) corrispondente al pin **17** della porta parallela

Il Registro dei bit di controllo sarà quindi così configurato:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Np	Np	Np	Np	PIN 17	PIN 16	PIN 14	PIN 1
/	/	/	/	<i>SLCTIN</i>	<i>INIT</i>	<i>AUTOFD</i>	<i>STROBE</i>

I restanti pin dal **18 al 25** sono tutti pin di massa.

Gestione di I/O con la porta parallela

Il problema nella gestione di I/O con la porta parallela si presenta durante l'acquisizione dei dati.

Visto che non è possibile acquisire un **byte** direttamente (IN PARALLELO) perché i bit di input sulla parallela sono solamente 5, per acquisire una parola quindi si deve procedere all'acquisizione per **nibble**.

L'acquisizione a nibble consiste nel acquisire per prima la parte LSB (i primi 4 bit meno significativi) e poi la parte MSB (i 4 bit più significativi).

Per fare questo, gli 8 bit che contengono il dato verranno divisi in due pacchetti da 4 bit (nibble). Ognuno di questi verrà selezionato ed inviato agli ingressi della porta parallela tramite il buffer tri-state **74LS244**, che svolge la funzione di "multiplexer". La selezione è effettuata dai pin **1** e **14** della porta parallela. Il nibble verrà così letto ed il software effettuerà la ricostruzione dei due nibble per formare il byte.

Nell'utilizzo in uscita della porta parallela non ci sono problemi di gestione visto che abbiamo:

- 8 bit della linea dati, che possono comodamente essere utilizzati per l'invio di un byte
- 4 bit di controllo, che di solito vengono utilizzati per la gestione delle periferiche collegate.

Nel caso si utilizzi Windows Xp l'accesso diretto ai "registri" della porta parallela non è possibile, quindi è necessario appoggiarsi ad una libreria che utilizza un API di Windows per leggere e impostare i valori sulla porta parallela.

La descrizione di tale libreria è spiegata nella relativa sezione ad essa dedicata (libreria JNPOUT32 e relativa dll).

Parte elettronica di Interfacciamento/Conversione

Descrizione Circuitale

Per effettuare l'acquisizione a nibble ho provveduto il circuito di un **74HCT244** (soluzione adottata in questo lavoro) che è un buffer tri-state formato da due gruppi da 4 buffer comandati separatamente. Tali gruppi sono selezionabili tramite i pin 1 e 19 (entrambi attivi bassi).

La gestione di tali pin avviene mediante parallela, attraverso il software da me scritto. Allego qui sotto gli schemi elettrici del circuito e i relativi sbrogli per la realizzazione del pcb.

Funzionamento del circuito

Il circuito possiede tre ingressi analogici contrassegnati dai connettori **JP1** , **JP4**, **JP5**. Il range di questi ingressi è **0÷5 V**. Tali ingressi sono collegati ad un Microcontroller **PIC16F818** che possiede un A/D Converter a 10 bit. Nel mio caso ho configurato l'A/D converter a 8 bit per facilitare l'acquisizione su porta parallela. Gli ingressi analogici sono collegati ad un AN-MUX (interno al PIC) che mi permette di selezionare un ingresso alla volta e farlo convertire dall' A/D converter del PIC. Per gestire la selezione dei canali dell' AN-MUX integrato nel PIC devo impostare i bit **CHS0** e **CHS1** appartenenti al registro **ADCON0** del PIC. Per poter fare questa selezione dall'esterno ho impostato due porte (**RA2 e RA4**) come ingressi, le quali a seconda della combinazione binaria applicata selezioneranno uno dei tre ingressi. Ho escluso via software la combinazione **11** che non avrebbe nessun riscontro. La gestione della selezione del canale può avvenire in due diverse modalità:

- Tramite gestione da PC (utilizzando l'apposito applicativo java)
- Tramite l'utilizzo del pulsante previsto per la selezione del canale di acquisizione

Il segnale analogico una volta convertito uscirà in parallelo sulle **PORTB** del PIC. Avremo quindi, partendo da **RBO (Pin 6)**, l'LSB e **RB7 (pin 13)** l'MSB.

Ho previsto anche di visualizzare il corrispondente valore della conversione attraverso 8 led. Per evitare di sovraccaricare le porte (tutte le **PORTB**) del PIC ho inserito un **74LS240** (buffer tri-State) che mi permette di pilotare i led con lo "0". Così facendo le porte del PIC non saranno sovraccaricate, poiché la corrente necessaria per l'accensione del led è fornita dall'alimentazione.

L'integrato **74LS244** dispone di 8 ingressi e 8 uscite, suddivise in due gruppi da 4. E' possibile selezionare quale ingresso portare in uscita attraverso i pin 1 e 19. Questi pin sono collegati ai pin 1 e 14 della porta parallela che gestirà attraverso il software java la selezione sequenziale di questi due canali (UNO ALLA VOLTA).

La parte che seleziona il canale manualmente è composta da un contatore a 2 bit realizzato con due FLIP FLOP J-K in configurazione toggle. Ai capi del pulsante è stato previsto un circuito anti-rimbalzo composto dalla resistenza R10, il condensatore C4 e la porta not a trigger di Schmitt **U5A**. Per far sì che ad ogni accensione il contatore riparta sempre dalla combinazione 00, è stato inserito il circuito di auto-reset composto dalla resistenza R11 e dal condensatore C5.

Per visualizzare gli ingressi selezionati ho aggiunto altri due led (LED9 e LED10) che visualizzeranno la combinazione binaria corrispondente.

Per la selezione dei canali (manuale o automatica) ho utilizzato 2 deviatori a levetta SW1 ed SW2.

Nota

Si consiglia di cortocircuitare a massa gli ingressi analogici non utilizzati del PIC

Convertitore A/D con PIC16F818

Come spiegato prima , la parte di conversione A/D è stata gestita attraverso l'uso di un microcontrollore (**PIC16F818**).

Il microcontrollore possiede un A/D converter a 10 bit che verrà invece utilizzato a 8 bit settando l'apposito registro (**ADCON1**) giustificato a destra. In questa maniera avremo nel registro **ADRESH** il risultato della conversione a 8 bit.

Una volta convertito il dato analogico il risultato sarà mandato in uscita su **PORTB**.

Per la selezione del canale da convertire bisognerà settare i registri **CHS0:CHS1:CHS2**.

Tali canali verranno settati internamente (tramite codice scritto in c) facendo attenzione agli stati di **PORTA** ed in particolare **RA2 ed RA4**.

Le porte **RA2** e **RA4** sono collegate sia alla selezione automatica (tramite parallela) che alla selezione manuale (tramite il pulsante di selezione manuale). Per selezionare la modalità (automatica o manuale) basta spostare i deviatori verso le apposite indicazioni.

La configurazione dell'A/D converter viene gestita attraverso i due registri **ADCON0 e ADCON1**.

In particolare :

- Il registro **ADCON0** è impostato nel seguente modo:

- **ADCON0=0B00000001 ;**

Che indica le seguenti impostazioni

0	0	0	0	0	0	0	1
Imposta il clock di conversione (ADCS1:ADCS0)	Seleziona il canale da convertire, per cambiare canale andremo ad agire su questi bit				GO/ \overline{DONE} indica lo stato della conversione, per far partire la conversione agiremo su questo bit	NON IMPLEMENTATO	Indica se la conversione è in corso o se non lo è

- Il registro **ADCON1** è impostato nel seguente modo:

- **ADCON1=0B00000100 ;**

Che indica le seguenti impostazioni

0	0	0	0	0	1	0	0
Imposta il formato dell'uscita: se 1 (giustificato a destra) le 6 cifre più significative del registro ADRESH sono tutti zeri. Se 0, giustificato a sinistra tutti e 6 i bit di ADRESL sono letti come zeri.	Serve per decidere se il clock del microC deve essere diviso durante l'utilizzo dell'ADC	Non implementato	Non implementato	Osservando la tabella riportata nel datasheet, con questi bit è possibile decidere quali porte saranno gli ingressi analogici e quali gli eventuali Vref (tensioni di riferimento) nel caso non si voglia utilizzare come riferimento la tensione di alimentazione			

```

/*****
/* ADC - LPT - PIC16F818
/* (C) D3v1L/[3v0xXx] - Davide Dal Fra 12/2005
/*****/

#include<pic16f81x.h>
#include"Delay.c"
short value=0;
void testLED()
{
    PORTB=0B00000001; //Accende il led 0
    DelayMs(125); //Crea un ritardo di 125 Millisecondi
    PORTB=0B00000010; //Accende il led 1
    DelayMs(125);
    PORTB=0B00000100; //Accende il led 2
    DelayMs(125);
    PORTB=0B00001000; //Accende il led 3
    DelayMs(125);
    PORTB=0B00010000; //Accende il led 4
    DelayMs(125);
    PORTB=0B00100000; //Accende il led 5
    DelayMs(125);
    PORTB=0B01000000; //Accende il led 6
    DelayMs(125);
    PORTB=0B10000000; //Accende il led 7
    DelayMs(125);
    PORTB=0B00000000; //Accende il led 8
}
main(void)
{
    PORTA=0; // setta tutte le porte del registro PORTA a 0
    TRISA=0xFF; //setta tutte le porte del registro PORTA come ingressi
    PORTB=0; //setta tutte le porte del registro PORTB a 0
    TRISB=0; //setta tutte le porte del registro PORTB come uscite
    ADCON0=0B00000001; //imposta il registro adcon0, come descritto sopra
    ADCON1=0B00000100; //imposta il registro adcon1, come descritto sopra

    testLed();//fa un test su tutti i led, accendendoli uno per volta
    while(1)
    {
        if(PORTA==0b00000000) //se viene selezionato il primo ingresso
        {
            ADCON0=0b00000001;//imposto il canale di conv. nel Registro
adcon0(bit CHS1:CHS2:CHS3)
        }
        if(PORTA==0b00000100) //se viene selezionato il secondo ingresso
        {
            ADCON0=0b00001001; ;//imposto il canale di conv. nel
Registro adcon0(bit CHS1:CHS2:CHS3)
        }
        if(PORTA==0b00010000) //se viene selezionato il terzo ingresso
        {
            ADCON0=0b00011001; ;//imposto il canale di conv. nel
Registro adcon0(bit CHS1:CHS2:CHS3)
        }
        GODONE=1; //Avvio la conversione
        DelayMs(100);
        while(GODONE) //Attendo la fine della conversione
            continue;
        value=ADRESH; //metto nella variabile value il risultato della conversione
        PORTB=value; //e mando in uscita su portb il risultato della conversione
    }
}

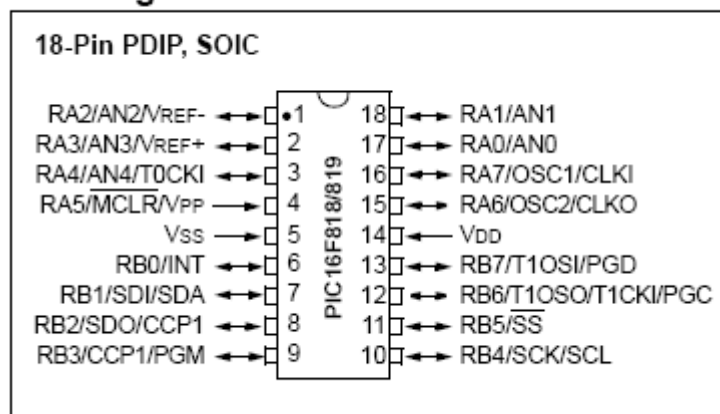
```

Datasheet dei componenti

Allego solo le pagine più significative.

PIC 16F818

Pin Diagram



-Registro ADCON0

REGISTER 11-1: ADCON0: A/D CONTROL REGISTER 0 (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits

If ADCS2 = 0:

00 = $F_{osc}/2$

01 = $F_{osc}/8$

10 = $F_{osc}/32$

11 = FRC (clock derived from the internal A/D module RC oscillator)

If ADCS2 = 1:

00 = $F_{osc}/4$

01 = $F_{osc}/16$

10 = $F_{osc}/64$

11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = Channel 0 (RA0/AN0)

001 = Channel 1 (RA1/AN1)

010 = Channel 2 (RA2/AN2)

011 = Channel 3 (RA3/AN3)

100 = Channel 4 (RA4/AN4)

bit 2 **GO/DONE:** A/D Conversion Status bit

If ADON = 1:

1 = A/D conversion in progress (setting this bit starts the A/D conversion)

0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit

1 = A/D converter module is operating

0 = A/D converter module is shut-off and consumes no operating current

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

-n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

-Registro ADCON1

REGISTER 11-2: ADCON1: A/D CONTROL REGISTER 1 (ADDRESS 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

- bit 7 **ADFM:** A/D Result Format Select bit
 1 = Right justified, 6 Most Significant bits of ADRESH are read as '0'
 0 = Left justified, 6 Least Significant bits of ADRESL are read as '0'
- bit 6 **ADCS2:** A/D Clock Divide by 2 Select bit
 1 = A/D clock source is divided by 2 when system clock is used
 0 = Disabled
- bit 5-4 **Unimplemented:** Read as '0'
- bit 3-0 **PCFG<3:0>:** A/D Port Configuration Control bits

PCFG	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	AVDD	AVSS	5/0
0001	A	VREF+	A	A	A	AN3	AVSS	4/1
0010	A	A	A	A	A	AVDD	AVSS	5/0
0011	A	VREF+	A	A	A	AN3	AVSS	4/1
0100	D	A	D	A	A	AVDD	AVSS	3/0
0101	D	VREF+	D	A	A	AN3	AVSS	2/1
011x	D	D	D	D	D	AVDD	AVSS	0/0
1000	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1001	A	A	A	A	A	AVDD	AVSS	5/0
1010	A	VREF+	A	A	A	AN3	AVSS	4/1
1011	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1100	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	A	AVDD	AVSS	1/0
1111	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input

D = Digital I/O

C/R = Number of analog input channels/Number of A/D voltage references

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

-n = Value at POR

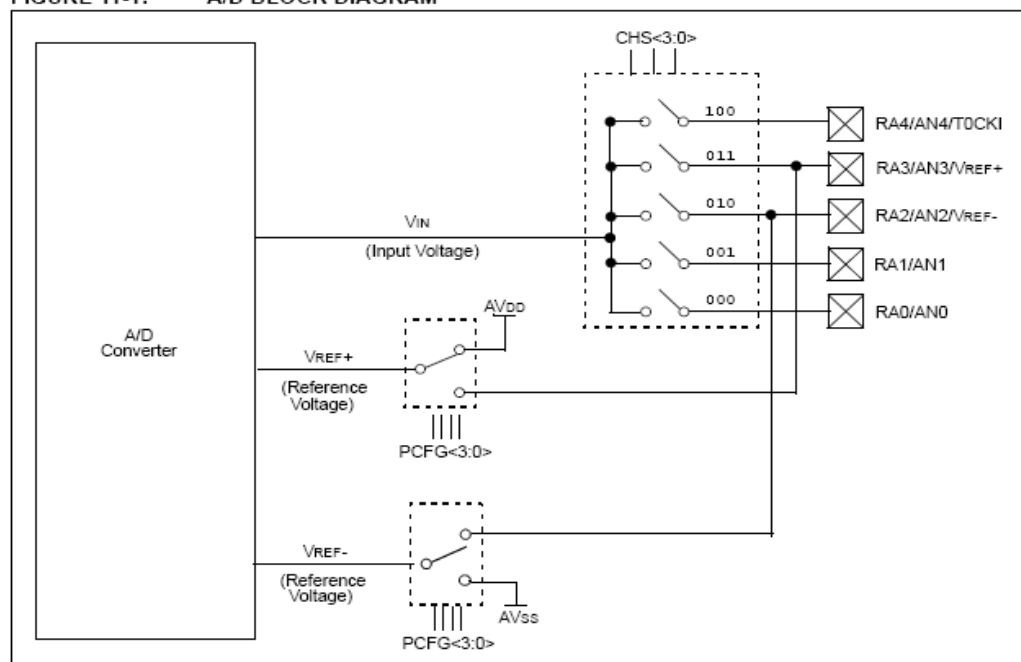
'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown

-Schema interno dell' A/D converter con AN-MUX

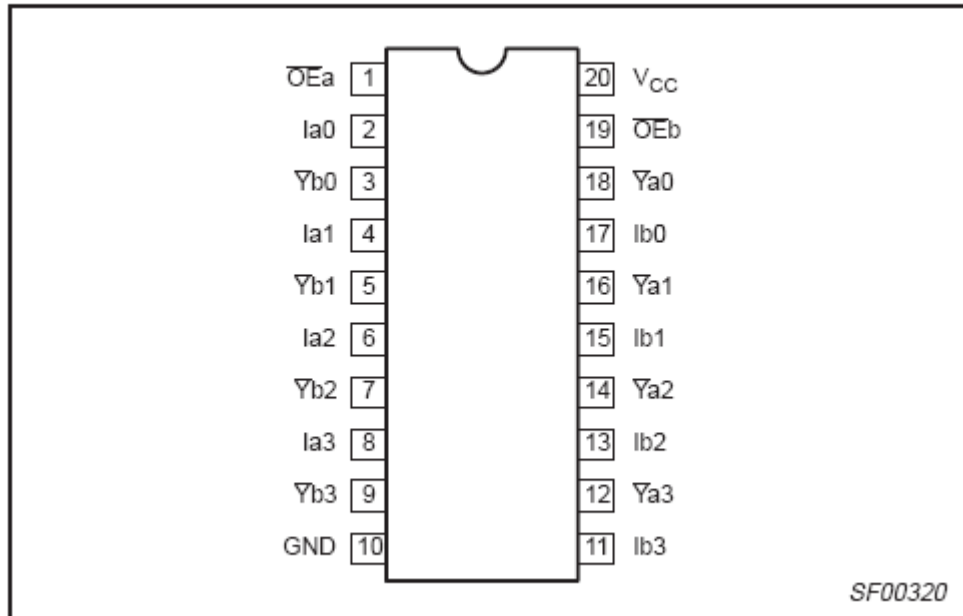
FIGURE 11-1: A/D BLOCK DIAGRAM



74LS240

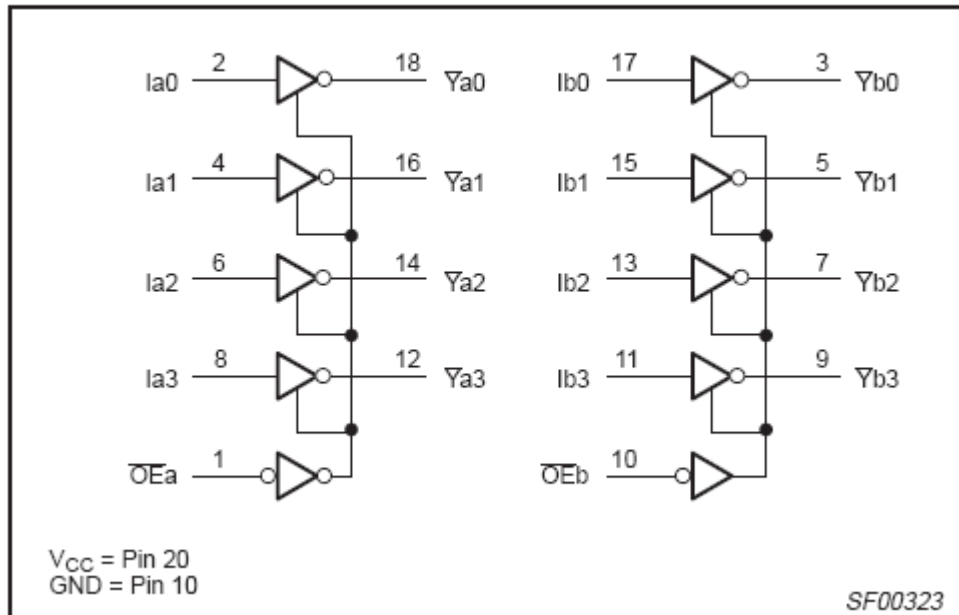
Piedinatura

PIN CONFIGURATION



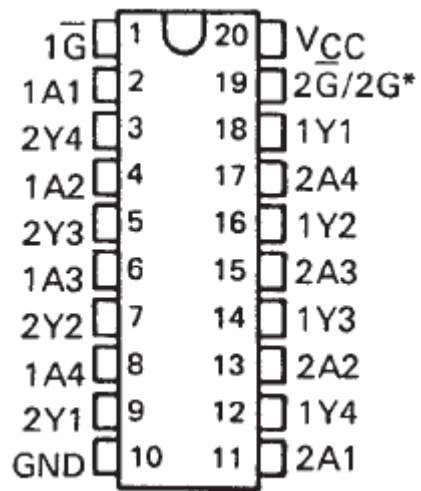
Schema interno

LOGIC DIAGRAM

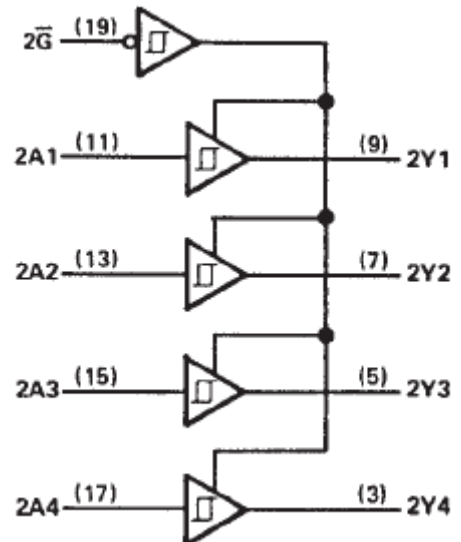
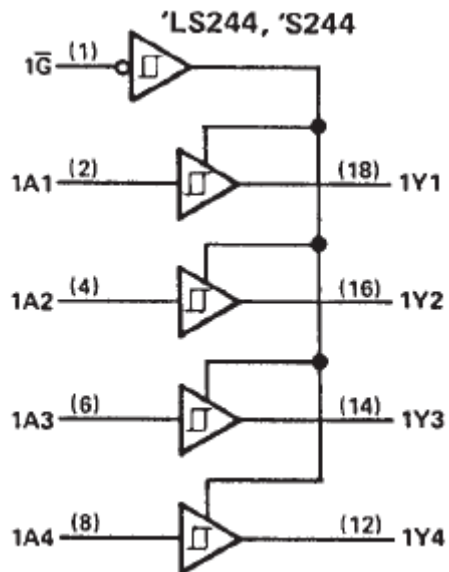


74LS244

Piedinatura

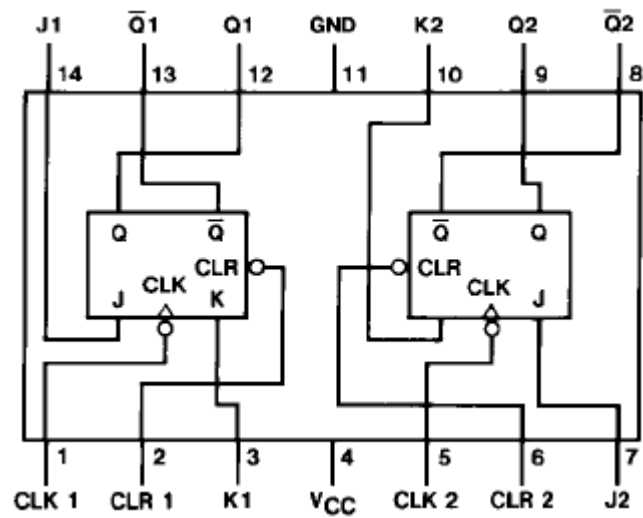


Schema interno



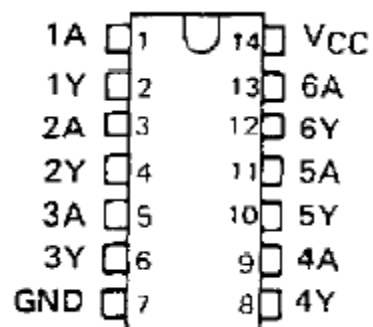
74LS73

- Schema Interno e Piedinatura

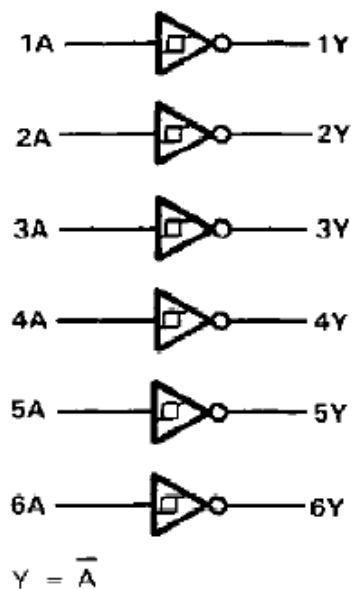


74LS14

- Piedinatura

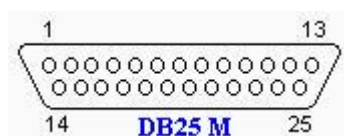


- Schema Interno

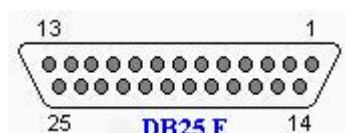


CONNETTORE DB25

- Versione M



- Versione F



La libreria JNPOUT32 e relativa DLL

Per permettere al computer di leggere i dati dalla parallela è necessario utilizzare un “interfaccia software” che permette la comunicazione tra Applicativo e parte Hardware (la porta parallela). Non è più possibile leggere direttamente il valore della porta come avviene in ambiente DOS.

Quindi per permettere all'applicativo Java di leggere e impostare valori sulla porta parallela è necessario utilizzare il package JNPOUT32 e una DLL da copiare nella cartella system32 del nostro pc.

Questa libreria si collega alla DLL che provvederà a leggere o impostare i valori della porta parallela.

Per quanto riguarda la parte di output non ho riscontrato nessun problema.

Ho eseguito correttamente i test di output agli indirizzi **0378h** e **037Ah**.

Per quanto riguarda l'acquisizione invece ho riscontrato problemi.

Il problema riscontrato è il seguente ed è descritto con un esempio:

Byte da Acquisire:

0 1 0 1 0 0 1 0

Vista la obbligatorietà di eseguire l'acquisizione solamente a nibble, il primo nibble meno significativo è 0010. Durante l'acquisizione in binario ho riscontrato che al posto di ottenere il nibble 0010 ho ottenuto solamente 10.

Per acquisire il dato in input è necessario utilizzare il metodo **.input()** che restituisce un valore di tipo **short**. Suppongo che il problema della perdita dei due zeri sia nella riconversione da short a binario, utilizzando il metodo della classe **Integer.toBinaryString**. Per fare la ricomposizione del byte attraverso i due nibble è necessario “recuperare” i due 0 persi.

Per ovviare a questo problema e per rendere il codice dell'interfaccia GUI più semplice ed interpretabile ho deciso di scrivere una classe che includa tutti i metodi di acquisizione, output e soprattutto di integrazione degli 0 nella conversione in forma binaria.

Ho quindi realizzato la classe ACQInterface, basata sul package JNPOUT32.

Modifiche alla libreria: La classe ACQInterface

La classe ACQInterface è basata, per quanto riguarda la gestione di I/O, sulla libreria JNPOUT32.

Ho scritto i seguenti metodi nella classe:

- **void blankLS244Select()** ha il compito di portare a 0 i due pin dedicati alla selezione del primo o del secondo nibble in uscita all'integrato **LS244**. I pin interessati a questo comando sono:
 - Pin 1 LPT → Pin 1 74LS244 ($\overline{1OE}$)
 - Pin 14 LPT → Pin 19 74LS244 ($\overline{2OE}$)

- **void blankSelectionChannel()** ha il compito di portare a 0 i bit per la selezione del canale da convertire ovvero porterà a 0 le porte **RA4** e **RA2** del PIC.
- Pin 2 LPT → Pin 1 PIC16F818
- Pin 3 LPT → Pin 3 PIC16F818
- **void setStrobe(boolean StrobeValue):** ha il compito di settare il pin di Strobe
- **void setAutoFD(boolean AutoFDValue):** ha il compito di settare il pin di AutoFD
- **void setDataOutput(short pin,short value):** ha il compito di impostare un valore compreso tra 0 e 255 all'indirizzo 0378h sulla parallela (gli 8 bit di dati).
- **void setInputChannel(int channel):** questa funzione ha il compito di selezionare il canale analogico da convertire. Fornito come parametro, il canale (compreso tra 1 e 3) provvederà automaticamente ad inviare i corrispondenti bit di selezione sulle porte **RA2** e **RA4** del PIC.
- **String KeepData(int channel):** ha il compito di acquisire i due nibble dalla parallela e di restituire come uscita un byte ricomposto di quegli zeri persi nella conversione da short a binario. Per fare questa reintegrazione degli zeri la funzione **KeepData** si basa su un'altra funzione di nome stringToInteger che restituirà la stringa di bit ricomposta più altre due stringe (LSB E MSB) di cui parlerò dopo.
- **String stringToInteger(String msb,String lsb):** questa funzione ha come parametri le stringhe convertite con la funzione **toBinaryString**, integra in automatico gli zeri mancanti e come output ha una stringa composta da 24 caratteri con un separatore (il ' ; ') ogni 8 caratteri, così avremo 3 gruppi da 8 bit. Il primo gruppo sarà l'MSB ricomposto , il secondo Gruppo sarà l'LSB composto e il 3 gruppo sarà l'intero Byte Composto.
- **int getIntValue(String sDataBit):** questa funzione ha il compito di convertire il byte in forma binaria (ovvero quello ricomposto e formato dagli 8 bit) nel corrispondente numero intero
- **float getVoltValue(int channel):** ha il compito di restituire il corrispondente valore in tensione misurato. Per fare ciò è necessario impostare la costante **toVolt** nel seguente modo:

$$toVolt = \frac{Vfs}{Bit}$$

Dove Vfs sarà la tensione di fondoscala , Bit invece rappresenta il numero di bit con cui lavora l'ADC.

Allego di seguito il sorgente della libreria **ACQInterface.java**

```
package acqp;
import jnpout32.*;
import java.math.*;

public class ACQInterface {

    private pPort LPTPort=new pPort();
    private int AddressDataOut;
    private int AddressDataCTRL;
    private int AddressStatusIn;
    private final float toVolt=((float)((float)5/(float)255));
    public ACQInterface(int ADO,int ADCT,int ASI)
    {
        AddressDataOut=ADO;
        AddressDataCTRL=ADCT;
        AddressStatusIn=ASI;
    }
    public void blankLS244Select()
    {
        LPTPort.setAddress(AddressDataCTRL);
        LPTPort.setPin((short)2,(short)1);
        LPTPort.setPin((short)3,(short)1);
    }
    public void blankSelectionChannel()
    {
        LPTPort.setAddress(AddressDataOut);
        LPTPort.setAllDataBits((short)0);
    }
    public void setStrobe(boolean StrobeValue)
    {
        LPTPort.setAddress(AddressDataCTRL);
        if(StrobeValue)
        {
            LPTPort.setPin((short)2,(short)1);
        }
        if(!StrobeValue)
        {
            LPTPort.setPin((short)2,(short)0);
        }
    }
    public void setAutoFD(boolean AutoFDValue)
    {
        LPTPort.setAddress(AddressDataCTRL);
        if(AutoFDValue)
        {
            LPTPort.setPin((short)3,(short)1);
        }
        if(!AutoFDValue)
        {
            LPTPort.setPin((short)3,(short)0);
        }
    }
    public void setDataOutput(short pin,short value)
    {
        LPTPort.setAddress(AddressDataOut);
        LPTPort.setPin(pin,value);
    }
    private void setInputChannel(int channel)
```



```

{
    switch(channel)
    {
        case 1:
            setDataOutput((short)2,(short)0);
            setDataOutput((short)3,(short)0);
            break;
        case 2:
            setDataOutput((short)2,(short)1);
            setDataOutput((short)3,(short)0);
            break;
        case 3:
            setDataOutput((short)2,(short)0);
            setDataOutput((short)3,(short)1);
            break;
        default:
            System.out.println("Canale non supportato");
    }
}
}
public String KeepData(int channel)
{
    setInputChannel(channel);
    String dataBit;
    setStrobe(true);
    setAutoFD(false);
    LPTPort.setAddress(0x379);
    short lsb=LPTPort.input();
    //entra MSB
    setStrobe(false);
    setAutoFD(true);
    LPTPort.setAddress(0x379);
    short msb= LPTPort.input();
    //converto in stringa e passo alla funzione per vedere la stringa
bit intera
    String sLSB=Integer.toBinaryString(lsb);
    String sMSB=Integer.toBinaryString(msb);
    String toVisualize[]=stringToInteger(sMSB,sLSB).split(";");
    dataBit=toVisualize[2];
    return dataBit;
}
public String stringToInteger(String msb,String lsb)
{
    String MSB[]=new String[7];
    String LSB[]=new String[7];
    String DataBit[]=new String[8];
    switch(msb.length())
    {
        case 1:
            for(int a=0;a<7;a++)
            {
                MSB[a]="0";
            }
            break;
        case 4:
            for(int b=0;b<3;b++)
            {
                MSB[b]="0";
            }
            for(int c=3;c<7;c++)
            {
                MSB[c]=String.valueOf(msb.charAt(c-3));
            }
            break;
        case 5:
            MSB[0]="0";

```

```

        MSB[1]="0";
        for(int d=2;d<7;d++)
        {
            MSB[d]=String.valueOf(msb.charAt(d-2));
        }
        break;
    case 6:
        MSB[0]="0";
        for(int e=1;e<7;e++)
        {
            MSB[e]=String.valueOf(msb.charAt(e-1));
        }
        break;
    default:
        for(int er=0;er<7;er++)
        {
            MSB[er]=String.valueOf(msb.charAt(er));
        }
        break;
    }
    switch(lsb.length())
    {
        case 1:
            for(int f=0;f<7;f++)
            {
                LSB[f]="0";
            }
            break;
        case 4:
            for(int g=0;g<3;g++)
            {
                LSB[g]="0";
            }
            for(int h=3;h<7;h++)
            {
                LSB[h]=String.valueOf(lsb.charAt(h-3));
            }
            break;
        case 5:
            LSB[0]="0";
            LSB[1]="0";
            for(int i=2;i<7;i++)
            {
                LSB[i]=String.valueOf(lsb.charAt(i-2));
            }
            break;
        case 6:
            LSB[0]="0";
            for(int l=1;l<7;l++)
            {
                LSB[l]=String.valueOf(lsb.charAt(l-1));
            }
            break;
        default:
            for(int el=0;el<7;el++)
            {
                LSB[el]=String.valueOf(lsb.charAt(el));
            }
            break;
    }
}

```

```

        //compongo la stringa bit esatta(mascherando i bit non utilizzati
        for(int f=0;f<8;f++)
        {
            if(f<4)
            {
                DataBit[f]=MSB[f];
            }
            if(f>=4 && f<8)
            {
                DataBit[f]=LSB[f-4];
            }
        }

        return
        MSB[0]+MSB[1]+MSB[2]+MSB[3]+MSB[4]+MSB[5]+MSB[6]+"; "+LSB[0]+LSB[1]+LSB[2]+L
        SB[3]+LSB[4]+LSB[5]+LSB[6]+"; "+DataBit[0]+DataBit[1]+DataBit[2]+DataBit[3]+
        DataBit[4]+DataBit[5]+DataBit[6]+DataBit[7];
    }
    public int getIntValue(String sDataBit)
    {
        int sum=0;
        int bitValue=0;

        String[] DataBit=new String[8];
        for(int b=0;b<8;b++)
        {
            DataBit[b]=String.valueOf(sDataBit.charAt(b));
        }
        for(int a=0;a<8;a++ )
        {
            bitValue=Integer.valueOf(DataBit[a]).intValue();
            sum+=bitValue*Math.pow((double)(2),((double)7-a));
        }
        return sum;
    }

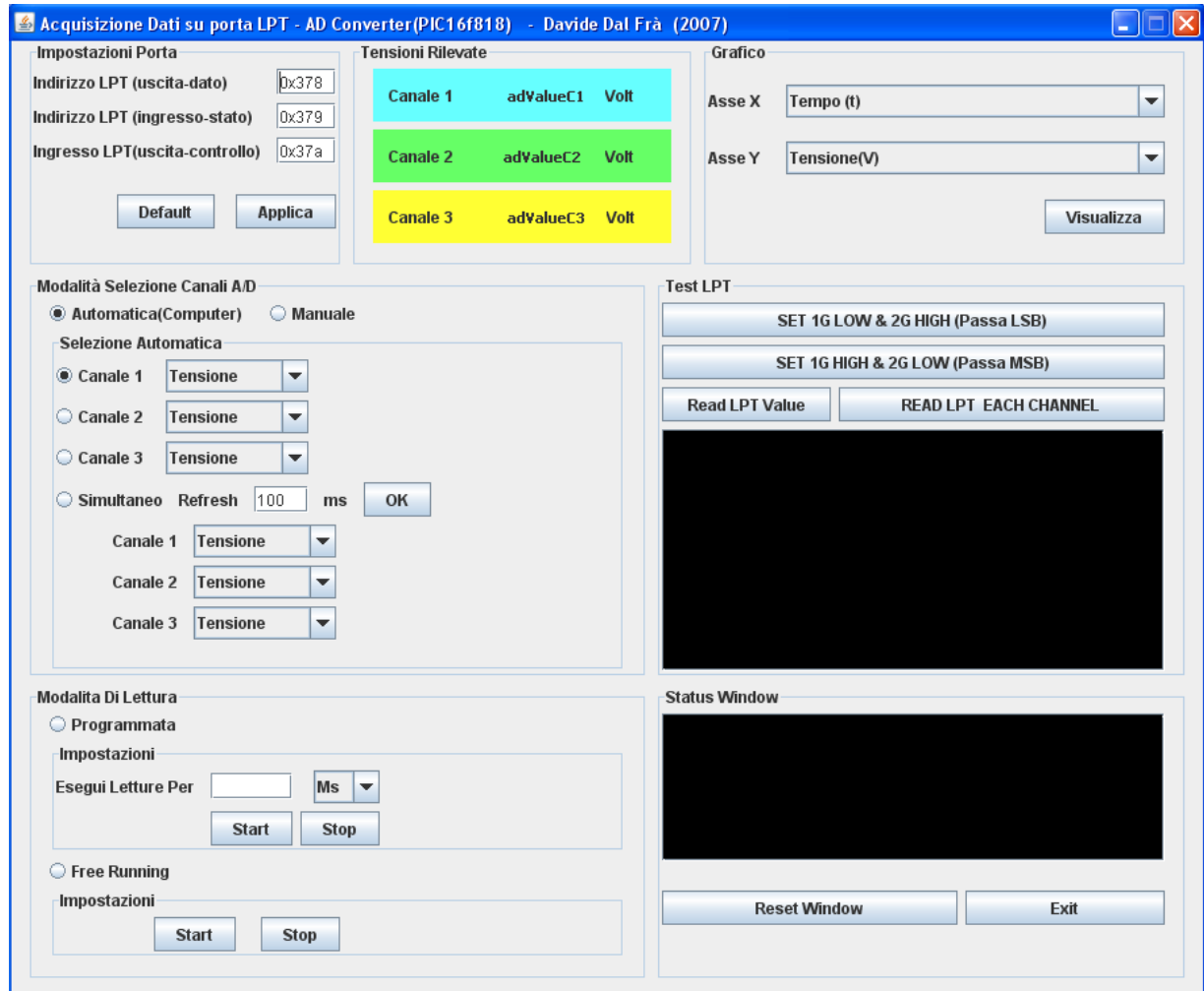
    public float getVoltValue(int channel)
    {
        int fDataBit=0;
        fDataBit=getIntValue(KeepData(channel));

        return (Float.valueOf(fDataBit).floatValue()*toVolt);
    }
}

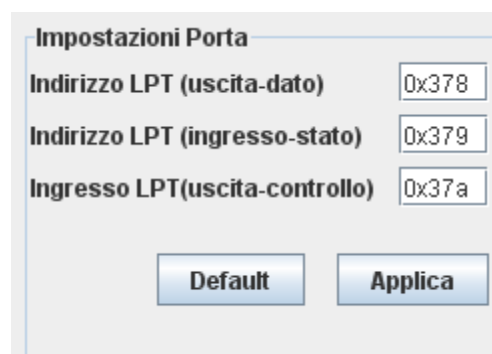
```

Applicativo Java

Il seguente applicativo serve per acquisire il byte, attraverso la ricomposizione dei due nibble, e visualizzare il dato analogico convertito. Per semplificare la spiegazione riporto sotto l'immagine principale del programma.



Nel riquadro "impostazioni porta" è possibile impostare gli indirizzi della porta parallela nel caso in cui volessimo utilizzare una porta parallela secondaria installata su un altro slot.



Nel frame “Modalità selezione canale A/D” è possibile impostare la modalità di selezione dei canali da convertire. Sono selezionabili le seguenti Modalità:

- Automatica (gestita da computer) la quale si suddivide in altre 4 modalità:
 - Canale 1 → effettua le letture sul Canale 1
 - Canale 2 → effettua le letture sul Canale 2
 - Canale 3 → effettua le letture sul Canale 3
 - Simultaneo → effettua le letture su tutti i canali sequenzialmente con un Timeout tra una misurazione e un'altra impostabile nell'apposita textBox

Modalità Selezione Canali A/D

☒ Automatica(Computer) ☐ Manuale

Selezione Automatica

☒ Canale 1 Tensione ▼

☐ Canale 2 Tensione ▼

☐ Canale 3 Tensione ▼

☐ Simultaneo Refresh 100 ms OK

Canale 1 Tensione ▼

Canale 2 Tensione ▼

Canale 3 Tensione ▼

- Nel frame “Modalità di lettura” è possibile selezionare le due modalità:
 - **Programmata** esegue letture per “x” Ms o Sec
 - **Free Running** esegue letture fino a che l'utente non preme stop.

Modalità Di Lettura

☒ Programmata

Impostazioni

Esegui Letture Per Ms ▼

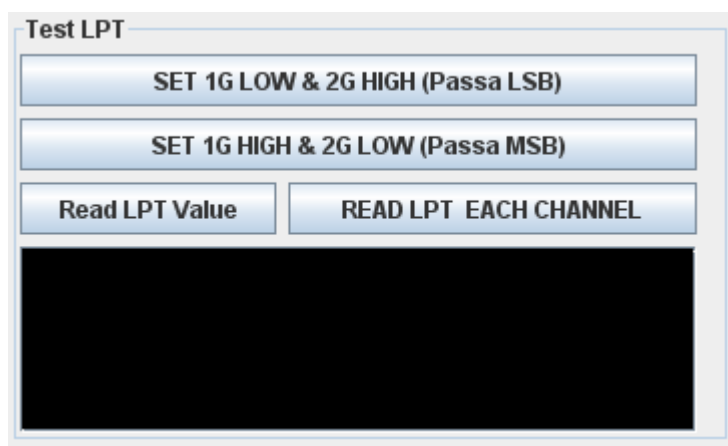
Start Stop

☐ Free Running

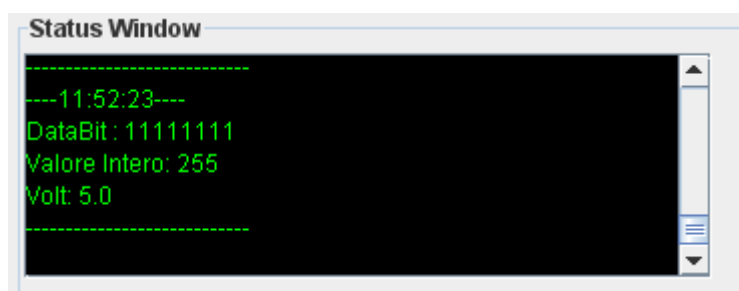
Impostazioni

Start Stop

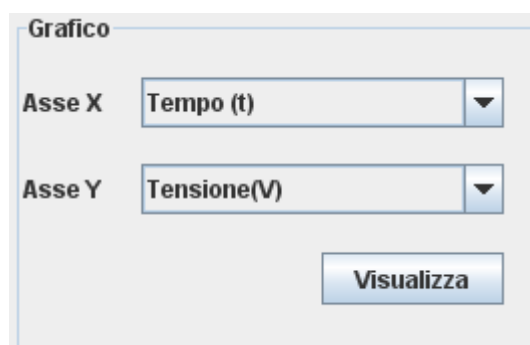
- Il frame dedicato al test della porta LPT è stato implementato durante la fase di test del programma per verificare i nibble acquisiti e il valore short del nibble. Potrebbe essere utilizzata nel caso in cui ci siano problemi di acquisizione facendo il debug passo a passo.



- Nella tabella status window vengono visualizzati i dati di ogni misurazione, come nell'immagine sottostante:



- Inoltre ho previsto anche la visualizzazione del grafico relativo alla misurazione avvenuta. Nella visualizzazione del grafico sono ammissibili al massimo 800 misurazioni visualizzabili nel grafico.



Sensori Applicati

Per fare una semplice dimostrazione di come avvengono le misure, sia in singolo canale che in multicanale, ho deciso di collegare a due dei tre ingressi disponibili due sensori, uno di temperatura ed uno di gas.

Sensore di Temperatura

Il sensore di temperatura utilizzato è l'AD590 il quale ha un range di temperatura compreso tra -55°C e +150°C.

La risposta del sensore è in corrente, secondo la legge:

$$I_{(t)} = I_0 + kT$$

Dove a °C:

$$I_0 = 273\mu A$$

e k:

$$k = \frac{1\mu A}{^{\circ}C}$$

Ho previsto di utilizzare il sensore per misurare temperature ambientali di una casa, le rientrano nel range 0 - 31 °C.

Quindi potremo riassumere il tutto nello schema sottostante:

$$\begin{array}{l} I_{(t)} = 273\mu A \longrightarrow 0^{\circ}C \longrightarrow 0Volt \\ I_{(t)} = 304\mu A \longrightarrow 31^{\circ}C \longrightarrow 5Volt \end{array}$$

Per poter convertire il corrispondente valore in corrente in tensione, compreso nel range 0-5V ho introdotto un convertitore corrente tensione utilizzando una resistenza. Allego lo schema elettrico e PCB del sensore di temperatura.

Spiegazione del circuito

La corrente generata dal sensore AD590 viene fatta scorrere nella resistenza R7 per convertire la corrente in tensione. Il segnale in tensione viene collegato all'ingresso non invertente dell'operazionale U3 che è configurato inseguitore. Il secondo Operazionale U4 è configurato in modalità differenziale per permettere di effettuare la taratura del circuito, annullando l'eventuale offset presente. Il partitore composto dal trimmer R1 e la resistenza R2 serve per regolare la temperatura effettiva durante la taratura del circuito. L'ultimo operazionale anch'esso in configurazione di inseguitore. Durante la fase di taratura e prova del circuito si è notato che la tensione oscillava in modo anomalo (sul tester a selezione automatica continuavano a lampeggiare le spie AC e DC). Si è capito così che era necessario applicare un filtro passa basso del primo ordine in uscita per permettere la taratura e poi l'effettivo utilizzo del sensore.

La frequenza di taglio del filtro è: $f_t = \frac{1}{2\pi RC}$ dove R=100K Ω e C=15 nF

Per effettuare la taratura mi è stata fornita gentilmente dal prof. Gaetano Grieco la tabella relativa alle tensioni presenti nei vari TP (test Point).

Allego qui sotto la tabella necessaria per la taratura del circuito:

La taratura è semplice basterà infatti regolare attraverso il trimmer la tensione in P1, e vedere le corrispondenze in P3 e in uscita.

Temp.	P1	P3	Out
0 °C	2,738	2,750	0,010
1 °C	2,738	2,760	0,020
2 °C	2,737	2,765	0,160
3 °C	2,737	2,780	0,240
4 °C	2,737	2,790	0,410
5 °C	2,737	2,800	0,630
6 °C	2,737	2,810	0,800
7 °C	2,736	2,820	0,950
8 °C	2,736	2,830	1,100
9 °C	2,736	2,840	1,280
10 °C	2,736	2,850	1,440
11 °C	2,736	2,860	1,600
12 °C	2,735	2,870	1,760
13 °C	2,735	2,880	1,980
14 °C	2,734	2,890	2,160
15 °C	2,734	2,900	2,360
16 °C	2,734	2,910	2,500

Temp.	P1	P3	Out
17 °C	2,734	2,92	2,66
18 °C	2,733	2,93	2,86
19 °C	2,733	2,94	3,07
20 °C	2,733	2,95	3,23
21 °C	2,732	2,96	3,35
22 °C	2,732	2,97	3,55
23 °C	2,731	2,98	3,71
24 °C	2,731	2,984	3,86
25 °C	2,73	2,99	4,04
26 °C	2,73	3	4,19
27 °C	2,73	3,01	4,36
28 °C	2,729	3,02	4,5
29 °C	2,729	3,03	4,68
30 °C	2,728	3,04	4,82
31 °C	2,728	3,05	4,96

Sensore di GAS

Il secondo sensore è un sensore di gas, siglato TGS813 della Figarò. Tale sensore risponde con una tensione compresa tra 0,350 V (nel caso di aria pura) e 5,350V nel caso di presenza di gas (la massima rilevabile è di 10000 ppm).

Il trasduttore è in grado di rilevare la maggior parte di gas combustibili quali il propano, metano, butano, idrogeno, etc.

Andremo quindi a configurare la parte di condizionamento di questo circuito in modo da avere un uscita compresa tra 0 e 5 V. La parte di condizionamento sarà composta da un amplificatore operazionale (INA111) in configurazione differenziale. Al morsetto invertente è stato collegato un partitore in grado di fornire una tensione compresa tra 0 e 0,680V, al morsetto non invertente invece è stata applicata la tensione in uscita dal sensore. Si utilizzerà il partitore collegato al morsetto invertente per annullare l'offset di 0,350V presente nel sensore in caso di aria pura. Il regolatore 7805 fornisce l'alimentazione di 5V al riscaldatore del sensore.

L'INA111 possiede oltre agli ingressi invertente, non invertente e all'alimentazione, due pin (1 e 8) con i quali è possibile regolare il guadagno dell'uscita secondo la formula:

$$G = 1 + \frac{50 \times 10^3 \Omega}{R_G}$$

Da cui:

$$R_G = \frac{1}{G - 1}$$

dove **G** rappresenta il guadagno, ed **R_G** rappresenta la resistenza applicata tra i pin 1 e 8.

Nel nostro caso è necessario un guadagno uguale a 1, per cui la resistenza **R_G** risulterà infinita. Per fare ciò quindi si lasciano i pin 1 e 8 non connessi.

Allego gli schemi relativi

Bibliografia

Datasheet:

- Microcontroller PIC 16F818 fonte **Microchip**
- Integrati Logici, National Semiconductor, Fairchild, St Micro
- Pin Out Connettori LPT fonte Internet
- Sensore di temperatura AD590, fonte Analog Devices
- Sensore di Gas, fonte Figarò Engineering (www.figarosensor.com)

Librerie Java

- Package JNPOUT32 e relativa dll sono stati scaricati dal sito www.logix4u.net (DLL)
<http://www.hytherion.com/beattidp/comput/pport.htm> (Java Pack)
- Librerie e Package utilizzati per effettuare i grafici (net.moioli) sono stati scaricati dal sito www.moioli.net

Sensori

- Il circuito di condizionamento del sensore di gas è basato sul modello proposto nel libro “Corso di Sistemi Volume 3” , A.De Santis, M.Cacciaglia, C.Saggese. (Pag 71)
- Le tabelle di taratura per il sensore AD590 sono state fornite dal prof. G.Grieco ,provenienti dall' Enia Spazi