

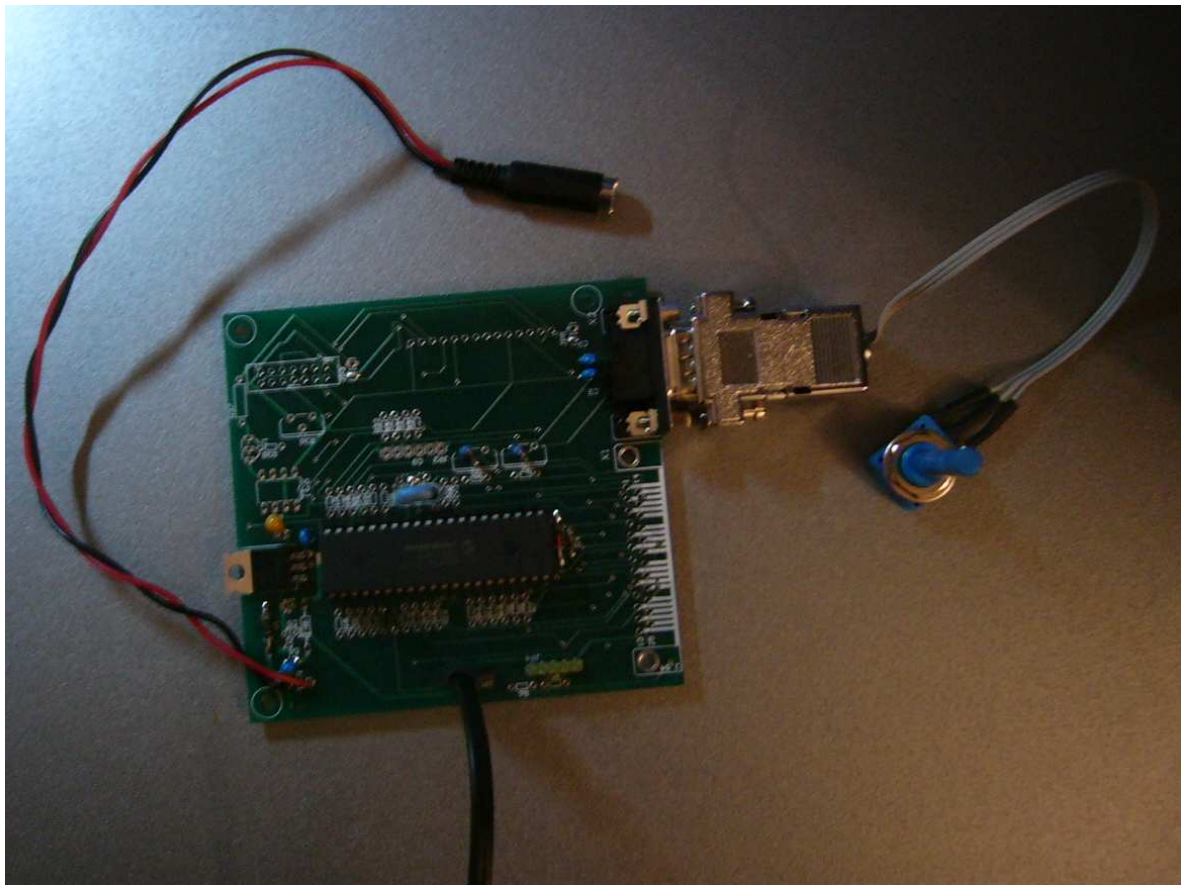
Relazione di sistemi elettronici Esami di Stato 2005

Alunno: Vincenzo Di Tanno

Classe 5^AetB

Coordinatore: Prof. Ettore Panella

Acquisizione di un segnale analogico mediante porta seriale RS232C



1. Introduzione

Il progetto che si descrive in questo lavoro parte dall'esigenza di acquisire, in forma numerica seriale, segnali analogici provenienti, ad esempio, da trasduttori.

In fig. 1 si mostra lo schema a grandi blocchi di un sistema di acquisizione.



Il segnale analogico proveniente ad esempio da un trasduttore di temperatura, viene opportunamente elaborato in modo da ottenere un segnale variabile nel campo di lavoro di input dell'ADC. Tipico range è: $V_{iADC} = 0 \div 5V$.

Il Convertitore Analogico Digitale ADC

Il Convertitore ha il compito di trasformare il dato analogico di ingresso in forma numerica a n bit. Tipicamente i convertitori ADC di uso commerciale operano con $n = 8$ bit, $n = 10$ bit, $n = 12$ bit, ecc.. I dati fondamentali forniti dal costruttore dell'ADC riguardano:

- 1) il numero di bit di conversione n (es. $n = 8$ bit)
- 2) il valore di fondoscala V_{FS} (es. $V_{FS} = 5V$)
- 3) il tempo di conversione T_c (es. $T_c = 50 \mu s$)
- 4) la risoluzione, o quanto $q = V_{FS}/2^n$

Il collegamento tra ADC e PC (Personal Computer) avviene utilizzando la porta di I/O del PC o costruendo un'apposita interfaccia. Nel nostro caso si utilizza come porta di I/O la seriale RS232C. Pertanto il circuito di conversione A/D deve prevedere un sistema di serializzazione dei dati nello standard RS232C.

Per risolvere il duplice aspetto legato alla conversione A/D e alle serializzazioni in formato RS232C si è usato il microcontrollore 16F874 della Microchip (www.microchip.com) opportunamente programmato.

Interfaccia seriale RS232C

Trasmissione seriale

Questo paragrafo descrive le caratteristiche della trasmissione seriale relativamente al livello fisico del modello ISO/OSI.

Nella trasmissione seriale il collegamento fra trasmettitore e il ricevitore si può realizzare con un minimo di due fili: il primo rappresenta la linea su cui viaggiano i bit, l'altro il filo di massa. Le trasmissioni seriali si dividono in:

- 1) sincrone;
- 2) asincrone.

Trasmissioni sincrone

Nelle trasmissioni sincrone il trasmettitore invia degli impulsi di clock contemporaneamente ai bit di informazione in modo da consentire al ricevitore la corretta lettura dei dati in arrivo ad intervalli regolari di tempo scanditi dal trasmettitore. Il collegamento, concettualmente, si realizza con 3 fili (clock, bit e massa) come in fig. 2.

In realtà si utilizzano due fili poiché i segnali di sincronismo si inviano sulla linea dati in coerenza col protocollo utilizzato. Se la trasmissione sincrona avviene tra un modem e l'interfaccia seriale di un computer, il clock può essere generato dall'interfaccia seriale o dal modem stesso. Se i dispositivi collegati sono due modem, il segnale di sincronismo è contenuto nella tensione analogica che il modem trasmettitore invia al ricevitore; quest'ultimo, attraverso l'operazione di demodulazione, estrae il segnale digitale che contiene particolari caratteri che consentono di sincronizzare il ricevitore al trasmettitore. I dati sono inviati in blocchi di decine o centinaia di caratteri. Ogni blocco è preceduto da caratteri di sincronismo e seguito da caratteri di controllo CRC (Codice Ciclico di Ridondanza), per la correttezza della trasmissione, e da un carattere che indica la fine del blocco trasmesso. In fig.3 si mostra il blocco di caratteri che transita in una trasmissione seriale sincrona tra due modem.



Trasmissioni asincrone

Nel collegamento seriale asincrono non si trasmette il clock ma il ricevitore genera un clock locale della stessa frequenza del trasmettitore.

Affinché i due clock risultino in fase, occorre che il ricevitore sappia quando ha inizio la trasmissione di un carattere in modo da sincronizzare la lettura dei vari bit.

In pratica un carattere in trasmissione è preceduto da un bit di start e seguito da uno o più bit di stop.

Il bit di start è costituito dal livello logico 0 mentre il bit di stop dal livello logico 1.

In assenza di trasmissione si ha il livello logico 1; quando la trasmissione ha inizio, l'applicazione del bit di start genera un fronte che sincronizza il clock del ricevitore.

Successivamente sono inviati in sequenza, ad intervalli regolari di tempo, i bit del carattere da trasmettere nel codice ASCII a 7 o 8 bit seguito, eventualmente, da un bit di parità pari o dispari e da uno o due bit di stop.

Le modalità di trasmissione, ovviamente, devono essere note prima che questa sia attivata.

Un sistema che trasmette a 9600 bps (bit per secondo) con 8 bit per carattere senza alcun bit di parità ed un solo bit di stop può trasferire fino a 960 caratteri al secondo poiché un carattere è costituito da 10 bit: 1 bit di start, 8 bit di dato, 1 bit di stop.

In fig.4 si mostra l'analisi temporale della trasmissione del byte 94H supponendo di attribuire al valore negativo di tensione (denominato *mark*) il bit 1 e al valore positivo di tensione (denominato *space*) il bit 0.

Le associazioni scelte sono, pertanto, in logica negativa come stabilito dalle raccomandazioni V.1 e V.4 dell'ITU-T².

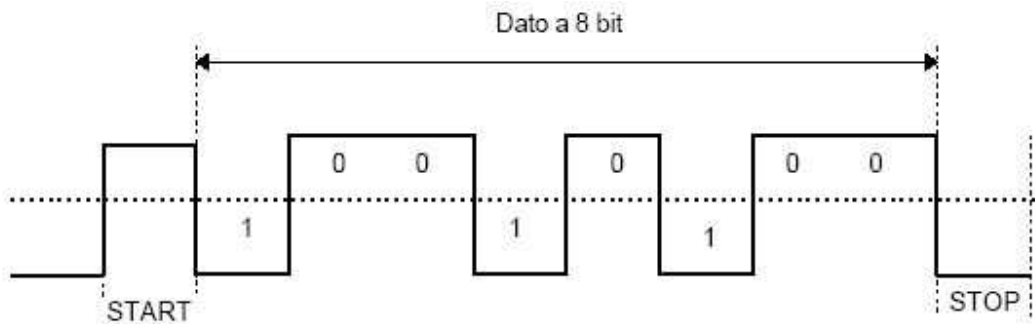


Fig. 4. - Temporizzazione della trasmissione del byte 94H senza bit di parità e con un solo bit di stop.

Il trasmettitore presenta nello stadio di uscita un circuito in grado di effettuare la conversione di un carattere dalla forma parallela a quella seriale in modo del tutto simile ad un registro a scorrimento con caricamento parallelo ed uscita seriale (PISO). Il ricevitore, invece, presenta nello stadio d'ingresso un circuito in grado di trasformare un carattere ricevuto da seriale a parallelo. Ciò si realizza con una soluzione simile ad un registro a scorrimento con caricamento seriale e uscita parallela (SIPO). Vi sono circuiti integrati in grado di comportarsi, all'occorrenza, sia da SIPO sia da PISO, presentano un clock locale che può essere selezionato sul valore richiesto. Essi prendono il nome di USART (Universal Synchronous Asynchronous Receiver Transmitter = ricevitore trasmettitore sincrono o asincrono universale) e sono utilizzati sia nei ricevitori che nei trasmettitori. Gli USART sono convenienti nelle trasmissioni bidirezionali dove i ruoli fra trasmettitore e ricevitore sono intercambiabili. Le velocità permesse nelle trasmissioni seriali asincrone hanno i seguenti valori espressi in bit al secondo: 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 14400, 19200, 57600, 115200. Gli USART più recenti consentono velocità maggiori.

Interfaccia seriale EIA RS232-C

L'interfaccia seriale americana EIA RS232-C (EIA= Electronic Industries Associates), corrispondente alla V.24/V.28 dell'ITU-T, è uno standard di collegamento seriale che può essere di tipo sincrono o asincrono tra un dispositivo di comunicazione DCE come, ad esempio, il modem (Data Communication Equipment) e un dispositivo terminale DTE, come, ad esempio, il computer (Data Terminal Equipment) con velocità di trasmissione inferiore o uguale a 19.2Kbps (questo limite è oramai superato). L'interfaccia è costituita da un insieme di 25 linee, non tutte indispensabili, che trasportano i bit di dati, segnali di controllo e la massa. Nel collegamento tra un computer ed un dispositivo periferico vengono adoperati dei connettori miniatura tipo D a 25 poli. Sul DTE (computer, ad

esempio) si trova la spina (connettore maschio) mentre sul DCE (modem) si trova la presa (connettore femmina). In alcuni DCE (ad esempio, il mouse seriale) manca la presa esterna poiché il cavo di collegamento entra direttamente nell'apparecchiatura. I tipici dispositivi periferici che si possono collegare ad un computer via RS232 sono il drive per dischetti, la stampante, il modem, il mouse ecc.

Caratteristiche meccaniche ed elettriche

In fig.9a si mostra il connettore a 25 poli per la RS-232C le cui caratteristiche meccaniche sono normalizzate secondo lo standard ISO 2110 della International Standard Organization. In molte applicazioni pratiche non si utilizzano tutte le linee ma solo una piccola parte di esse. In tal caso si fa uso di un connettore ridotto a 9 poli come quello in fig. 5.

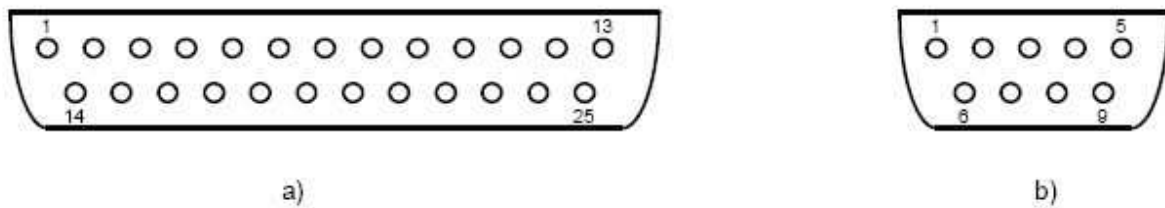


Fig. 5. - Connettore per la RS-232C: a) di tipo a 25 poli; b) di tipo a 9 poli.

Il significato di tali linee sarà descritto nelle pagine successive. Qualunque sia la linea (dati, clock o controlli), il circuito elettrico equivalente di tale interfaccia tra il trasmettitore e il ricevitore è quello indicato in fig.6.

Fig. 10. - Circuito equivalente tra trasmettitore e ricevitore nello standard RS232-C.

VT = f.e.m. del trasmettitore a circuito aperto;
 RT = Resistenza interna del trasmettitore;
 CT = Capacità equivalente del trasmettitore;
 CL = Capacità equivalente del ricevitore;
 RL = Resistenza d'ingresso del ricevitore;
 VL = f.c.e.m. del ricevitore a circuito aperto;
 Vi = d.d.p. all'interfaccia.

In tale standard si definisce *mark* la tensione V_i di valore inferiore a -3V e si definisce *space* quella superiore a +3V. Durante la trasmissione si associa il livello logico 1 a mark e 0 a space. Si osserva subito che i livelli logici sono bipolari e in logica negativa (tensione positiva=0; tensione negativa=1). Tipicamente i valori di tensione assunti sono $\pm 12V$. La resistenza di carico del ricevitore R_L deve essere compresa tra $3K\Omega$ e $7K\Omega$, la capacità C_L in parallelo al carico deve essere inferiore a $2.5nF$ e la f.c.e.m. V_L non deve superare i 2V.

La f.e.m. V_T del driver del trasmettitore non deve superare i 25V, R_T e C_T non sono specificati ma devono essere tali da evitare una corrente di corto circuito superiore a 0.5A e da consentire una V_i compresa tra 5 e 15V.

Poiché la capacità per unità di lunghezza di un cavo è di circa 200pF/metro si evince che la massima distanza tra i dispositivi collegati in tale standard non deve superare i 12-15 metri. In tabella 1 si descrive la piedinatura del connettore a 9 e 25 poli, il nome e la descrizione delle linee della RS-232C.

Tab. 1

PIN (9)	PIN (25)	NOME V.24 ITU	NOME RS-232	DESCRIZIONE
	1	C101	FG	Frame ground = Massa di protezione
3	2	C103	TxD	Trasmitted data = Dati in trasmissione
2	3	C104	RxD	Received data = Dati in ricezione
7	4	C105	RTS	Request to send = Richiesta di trasmissione
8	5	C106	CTS	Clear to send = Pronto a trasmettere
6	6	C107	DSR	Data set ready = DCE pronto
5	7	C102	GND	Ground = Massa dei segnali
1	8	C109	DCD	Data carrier detector = Portante in ricezione presente
	9			Riservato per apparecchi di collaudo
	10			Riservato per apparecchi di collaudo
	11	C126	CK	Scelta frequenza in trasmissione
	12	C122	SCF	Segnale di ricezione presente sul canale ausiliario
	13	C121	SCB	Pronto per la trasmissione sul canale ausiliario
	14	C118	SBA	Dati in trasmissione del canale ausiliario
	15	C114	TC	Transmit clock = Clock di trasmissione dal modem
	16	C119	SBB	Dati in ricezione del canale ausiliario
	17	C115	RC	Received clock = Clock di ricezione
	18			Non connesso
	19	C120	SCA	Richiesta di trasmissione del canale ausiliario
4	20	C108	DTR	Data terminal ready = DTE pronto
	21	C110	CG	Rivelatore della qualità del segnale
9	22	C125	RI	Ring indicator = Chiamata in arrivo
	23	C111	CI	Selezione velocità di trasmissione da DTE
	24	C113	DA	Clock di trasmissione da DTE
	25			Non connesso

Microcontrollore PIC 16F874

Si descrivono di seguito, le caratteristiche generali del microcontrollore de in particolare gli aspetti legati alla conversione A/D e alla trasmissione seriale. Per un approfondimento delle problematiche legate al software e all'hardware dell'integrato si rimanda al manuale d'uso.

In **fig. 7** si mostra la piedinatura dell'integrato.

Il 16F874 è un microcontrollore con memoria di programma di tipo *flash* che risulta particolarmente indicato per la messa a punto e la prova di programmi che fanno uso di dispositivi periferici non implementati nel 16C84. Sono integrati infatti nel *chip* i seguenti moduli periferici:

- 1 convertitore Analogico digitale a 10 bit con 5 canali di ingresso
- 3 timer
- 2 moduli *Capture, Compare, PWM*
- porte seriali sincrone ed asincrone

Si presuppone che il lettore abbia già acquisito una buona conoscenza delle caratteristiche tecniche e dei principi di programmazione del microcontrollore PIC16C84 ed in particolare dell'ambiente MPLAB.

Caratteristiche di base del PIC16F874

Il PIC16F874 è un dispositivo a 40 pin che lavora con frequenza massima di *clock* pari a 20 MHz e dispone di un set di 35 istruzioni. Ha 4 Kword di memoria FLASH di programma, 256 byte di memoria EEPROM per i dati a 368 byte di RAM (per i dati). Dispone di un set di tre porte di *I/O*: **Port A**, con cinque linee di *I/O*, **Port B** e **Port C** con 8 linee di *I/O*.

Della stessa serie esistono i microcontrollori 16F873 (con minore estensione di memoria dati e programma) aventi la stessa memoria del 16F874 ma differenziandosi per il fatto che il 16F874 è dotato di 8 canali analogici e 5 porte di *I/O*; ed infine il 16F877 con la stesa memoria del 16F876 ma con 8 canali analogici e 5 porte di *I/O*.

Nella figura 7 è rappresentato il PIC16F874 e nella tabella 8.1 sono descritti i principali segnali del dispositivo. Si tenga presente che molti dei pin svolgono più di una funzione.

Nella tabella 8.2 è riportata la mappa di memoria per i *file register* con i relativi indirizzi, suddivisi per banchi. Sono presenti quattro banchi selezionabili con i bit RP1 (bit6) e RP0 (bit5) del registro STATUS.

Sono mostrate inoltre le aree di memoria RAM per i dati (registri di uso generale).

TABLE 8.1 : PIC16F874 AND PIC16F877 PINOUT DESCRIPTION

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS ⁽⁴⁾	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
MCLR/VPP	1	2	18	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
RA0/AN0	2	3	19	I/O	TTL	<p>PORTA is a bi-directional I/O port.</p> <p>RA0 can also be analog input0.</p> <p>RA1 can also be analog input1.</p> <p>RA2 can also be analog input2 or negative analog reference voltage.</p> <p>RA3 can also be analog input3 or positive analog reference voltage.</p> <p>RA4 can also be the clock input to the Timer0 timer/counter. Output is open drain type.</p> <p>RA5 can also be analog input4 or the slave select for the synchronous serial port.</p>
RA1/AN1	3	4	20	I/O	TTL	
RA2/AN2/VREF-	4	5	21	I/O	TTL	
RA3/AN3/VREF+	5	6	22	I/O	TTL	
RA4/T0CKI	6	7	23	I/O	ST	
RA5/SS/AN4	7	8	24	I/O	TTL	
RB0/INT	33	36	8	I/O	TTL/ST ⁽¹⁾	<p>PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.</p> <p>RB0 can also be the external interrupt pin.</p> <p>RB3 can also be the low voltage programming input.</p> <p>Interrupt-on-change pin.</p> <p>Interrupt-on-change pin.</p> <p>Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.</p> <p>Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.</p>
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	
RB4	37	41	14	I/O	TTL	
RB5	38	42	15	I/O	TTL	
RB6/PGC	39	43	16	I/O	TTL/ST ⁽²⁾	
RB7/PGD	40	44	17	I/O	TTL/ST ⁽²⁾	

TABLE 8.1 : PIC16F874 AND PIC16F877 PINOUT DESCRIPTION (CONTINUA)

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	PORTC is a bi-directional I/O port. RC0 can also be the Timer1 oscillator output or a Timer1 clock input. RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output. RC2 can also be the Capture1 input/Compare1 output/PWM1 output. RC3 can also be the synchronous serial clock input/output for both SPI and I ² C modes. RC4 can also be the SPI Data In (SPI mode) or data I/O (I ² C mode). RC5 can also be the SPI Data Out (SPI mode). RC6 can also be the USART Asynchronous Transmit or Synchronous Clock. RC7 can also be the USART Asynchronous Receive or Synchronous Data.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	
RC2/CCP1	17	19	36	I/O	ST	
RC3/SCK/SCL	18	20	37	I/O	ST	
RC4/SDI/SDA	23	25	42	I/O	ST	
RC5/SDO	24	26	43	I/O	ST	
RC6/TX/CK	25	27	44	I/O	ST	
RC7/RX/DT	26	29	1	I/O	ST	
RD0/PSP0	19	21	38	I/O	ST/TTL ⁽³⁾	PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD1/PSP1	20	22	39	I/O	ST/TTL ⁽³⁾	
RD2/PSP2	21	23	40	I/O	ST/TTL ⁽³⁾	
RD3/PSP3	22	24	41	I/O	ST/TTL ⁽³⁾	
RD4/PSP4	27	30	2	I/O	ST/TTL ⁽³⁾	
RD5/PSP5	28	31	3	I/O	ST/TTL ⁽³⁾	
RD6/PSP6	29	32	4	I/O	ST/TTL ⁽³⁾	
RD7/PSP7	30	33	5	I/O	ST/TTL ⁽³⁾	

PORTE BIDIREZIONALI

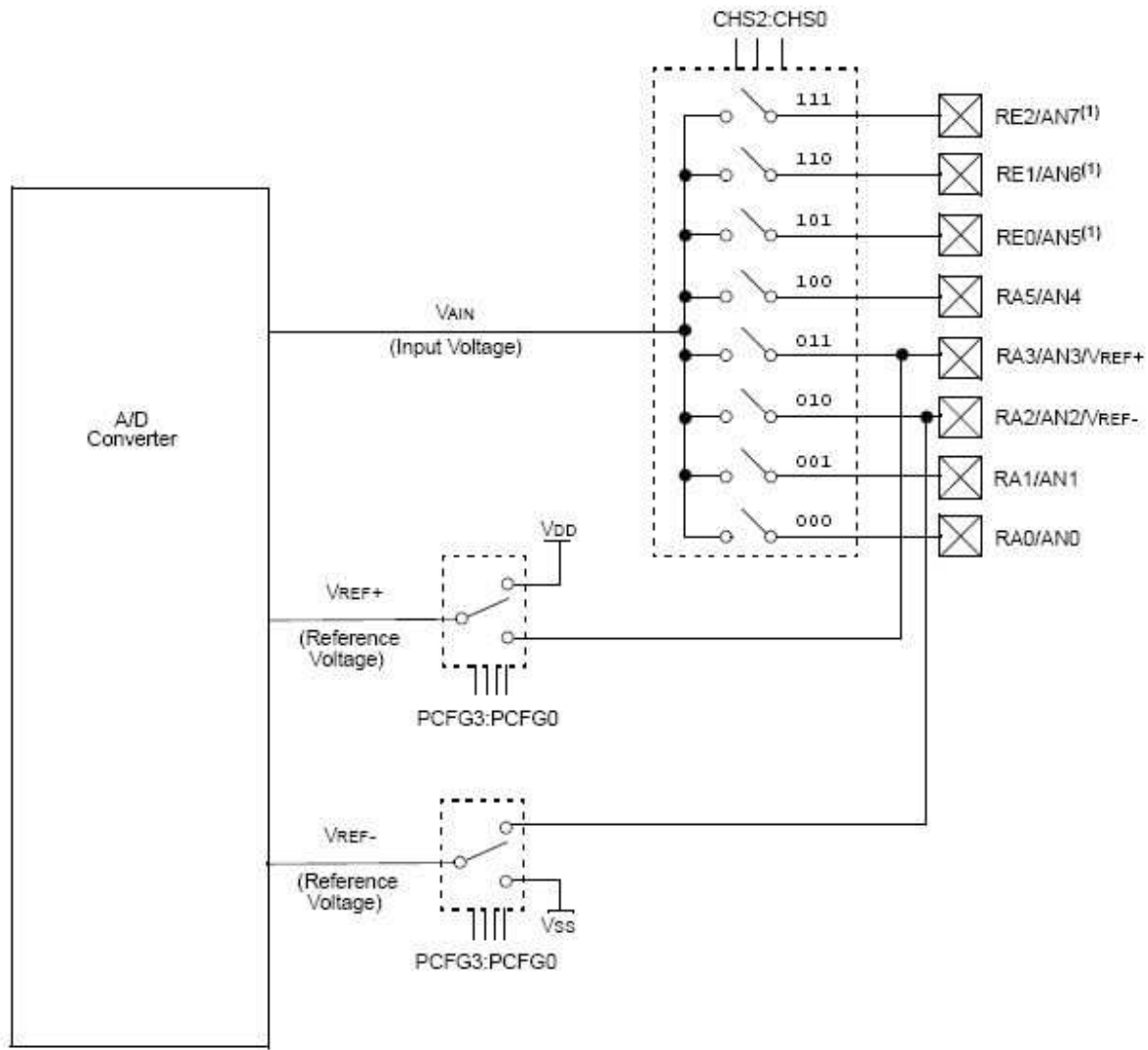
RE0/ $\overline{\text{RD}}$ /AN5	8	9	25	I/O	ST/TTL ⁽³⁾	PORTE is a bi-directional I/O port. RE0 can also be read control for the parallel slave port, or analog input5. RE1 can also be write control for the parallel slave port, or analog input6. RE2 can also be select control for the parallel slave port, or analog input7.
RE1/ $\overline{\text{WR}}$ /AN6	9	10	26	I/O	ST/TTL ⁽³⁾	
RE2/ $\overline{\text{CS}}$ /AN7	10	11	27	I/O	ST/TTL ⁽³⁾	
V _{ss}	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
V _{DD}	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28,40	12,13,33,34		—	These pins are not internally connected. These pins should be left unconnected.

Moduli periferici del PIC16F874

Nella presente area verranno presi in esame i moduli periferici del microcontrollore PIC16F874. In particolare verrà esaminato il funzionamento del convertitore A/D, dei tre TIMER e dei due moduli CCP (*Capture/Compare/PWM*).

In seguito sarà fornito il listato del programma che ha premesso di programmare la PIC nel linguaggio C, con la relativa spiegazione

Il convertitore A/D del PIC16F874



Il PIC16F874 ha un modulo convertitore Analogico digitale a 10 bit con 5 canali di ingresso ciascuno di essi selezionabile via *software*. Il convertitore, ad approssimazioni successive, è dotato internamente di *track-hold*.

I registri specifici associati con il modulo convertitore sono ADCON0 (all'indirizzo 1Fh) e ADCON1 (all'indirizzo 9Fh).

Il risultato della conversione (10 bit) viene posto nei registri a 8 bit ADRESH e ADRESL. Prima di avviare la conversione si debbono impostare tutti i parametri necessari dei registri ADCON0 e ADCON1 nel modo seguente:

1. Con il registro ADCON1 (vedere la seguente tabella) si seleziona l'abbinamento dei canali analogici con gli ingressi digitali (RA0 ÷ RA5), la sorgente della tensione di riferimento e la modalità di salvataggio del risultato della conversione nei registri ADRESH e ADRESL.

REGISTRO 11-2 : REGISTRO ADCON1 (INDIRIZZO 9Fh)

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

2. Con il registro ADCON0 (vedere la seguente tabella) si seleziona il clock di conversione (bit 7 e bit 6), il canale analogico per la conversione (bit 5 ÷ bit 3) e si pone in ON il convertitore A/D (bit 0).

REGISTRO 11-1 : REGISTRO ADCON0 (INDIRIZZO: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

3. Si attende che trascorra il Tempo di Acquisizione (tempo impiegato dal condensatore di holding per caricarsi al valore del segnale di ingresso). Come valore si sceglie 20 µs.

4. Si avvia la conversione ponendo ad uno il bit 2 (GO/DONE) di ADCON0.

5. Si controlla il termine della conversione verificando il bit 2 di ADCON0 (GO/DONE); quando diviene zero la conversione è terminata.

6. Si legge il dato convertito nei registri ADRESH (gli 8 bit più significativi) e ADRESL (i 2 bit meno significativi).

Ovviamente con l'allineamento a sinistra come risultato della conversione troveremo i 10 bit verso sinistra, trovando gli zeri solo nell'ADRESL.

Programmazione PIC16F874

Di seguito vengono elencati e commentati i file principali del firmware, commenteremo inizialmente il file header AD_cov.h proseguendo con il file in C, AD_conv.c.

```
#include <16F874.h>
//#device ICD=TRUE //per Debug
#device adc=10
//#fuses NOWDT,XT, NOPUT, NOPROTECT, LVP, NOCPD, NOWRT, BROWNOUT, DEBUG //per
Debug
#fuses NOWDT,XT, NOPUT, NOPROTECT, LVP, NOCPD, NOWRT, BROWNOUT
#use delay(clock=4000000) //Frequenza quarzo 4MHz
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,errors)
//dichiaro di usare la UART interna con un Baud Rate di 9600, 8bit, nessuna parità
#use fast_io(B) //con fast decido da prima come utilizzarli, quindi come
#use fast_io(C) //input o output, a-b-c-d-e sono i port della pic
#use fast_io(D)
#use fast_io(E)
#priority RDA, AD //priorità nella gestione dell'interrupt Rx > AD

//DEFINIZIONI

#define SINCRO '$' //carattere sincronismo in ricezione (da TX - PC)
#define NULL 0
#define CR 0x0D //carriage return
#define RX_BUFFER_SIZE 3
#define RX PIN_C7
#define TX PIN_C6

// ASSEGNAMENTO VARIABILI

static unsigned char cAD_Low; //valore basso conversione AD
static unsigned char cAD_High; // " alto "
static unsigned char cCount; //conteggio generico
static unsigned char cRxBuffer[RX_BUFFER_SIZE]; //buffer seriale
static unsigned char cRxprt; //puntatore
static unsigned char cRiferimenti; //registro generico per riferimenti (a bit)
//bit 0 = 1 done AD
//bit 1 = 1 done Rx
//.....
#define sADdone = cRiferimenti.0
#define sSdone = cRiferimenti.1
static float fResult; //risultato conversione floating (virgola mobile)
static unsigned long lResult; // " " in decimale
```

Di seguito è riportato l'AD_conv.c

```
/*Progetto didattico di conversione AD con MCU.
```

Si è abbandonato il classico ADC0804 ad 8bit, un pò per il basso numero di bit e per evidenziare come è preferibile utilizzare un microcontrollore con periferiche a bordo per la costruzione di progetti più compatti e performanti.

La pic scelta è una 16f874 sovradimensionata per lo scopo con un'alto numero di I/O, una Uart, I2C, SPI, PWM a bordo e 5 canali AD con possibilità varie di utilizzare gli stessi AD come ingressi per la tensione di riferimento (vedi 16f874.h).

In questo progetto la Vref è quella d'alimentazione, quindi si avrà una conversione di 10 bit (1024 step) spalmati da 0 a 5V con risoluzione di 10mV in visualizzazione (interna di 5 mV).

La pic inizia una conversione e la rende disponibile su seriale se riceve la seguente stringa:

```
< $ canale CR >
```

dove \$ = carattere sincronismo

canale <0 1> = canale di conversione utilizzato, se > 1 manda in uscita il seguente errore ""Canali di conversione max 2 ... Err!!"

CR = carattere di fine stringa (Carriage Return).

Il micro risponderà in uscita con delle stringhe riportanti la conversione in decimale e in Volts con il canale interrogato.

Es.

```
A/D conversione Decimale = 1022 (ch1)
```

```
A/D conversione Volts= Vcc 4.99 (ch1)
```

Si è scelto di non utilizzare l'affascinante assembler che avrebbe reso il codice più veloce nell'esecuzione e occupato meno flash ma più articolato e di difficile comprensione per chi non lo mastica, in favore del C, per la sua portabilità (riutilizzo del codice con poche modifiche cambiando micro) e per la sua leggibilità con programmi complessi.

Con il C o comunque con programmi ad alto livello si evita di conoscere a fondo la struttura interna del micro (anzi puoi anche non conoscerla) ma è bene che si legga in ogni caso il data sheet del micro per sapere sempre al meglio quello che fai e per ottenere il massimo, inserendo quando serve anche delle linee in assembler. Quindi un cocktail di C e assembler, ma solo se necessario.

Le tensioni da convertire si applicano sul connettore DB9 presente sulla scheda

Pin1 = canale 0 (ch0) visualizzato come ch1

Pin3 = canale 1 (ch1) visualizzato come ch2

Disponibili sul Pin2-4 GND e sul Pin9 +5Vcc

DI TANNO VINCENZO VETB

```
*/
```

```
#include "C:\Programmi\PICC\Enzo_AD\AD_conv.h"
```

```
#int_RDA
```

```
void RDA_isr() //segmento accessibile quando avviene un'interrupt in RX
```

```
{
```

```
    char cReceive; //dichiaro la variabile cReceive di tipo char
```

```
    if(rs232_errors & 0x04) //framing error, cancello errore, contatore e rif.
```

```
    {
```

```

    cReceive = getchar(); //scarico quindi il registro di ricezione
    sSdone = 0;
    cRxprt = &cRxBuffer[0]; //reinializzo puntatore
}
else
{
    cReceive = getchar(); //non ci sono errori e proseguo, metto in cReceive il dato
                        //in Rx
    if((cReceive == SINCRO) && (cRxprt == &cRxBuffer[0])) //controllo se è SINCRO
    {
        *cRxprt = cReceive; //ricevo carattere e metto in buffer (puntato da cRxprt)
        cRxprt ++; //passo a puntare la locazione successiva
        goto exit_int_rda; //esco dalla routine di gestione dell'interrupt in Rx
    }
    if(cRxBuffer[0] == SINCRO)
    {
        *cRxprt = cReceive; //ricevo carattere e metto in buffer
        cRxprt ++; //passo a puntare la locazione successiva
    }
    if((cRxBuffer[0] == SINCRO) && (cRxBuffer[2] == CR)) //se la stringa ricevuta inizia
                        // con SINCRO e termina con CR dovrebbe essere corretta
    {
        cRxprt = &cRxBuffer[0]; //reinializzo puntatore
        cRxBuffer[0] = NULL; //pulisco parte del buffer per sicurezza nella prox lettura
        cRxBuffer[2] = NULL;
        sSdone = TRUE; //stringa ricevuta correttamente ($ canale CR)
    }
    if(cRxprt > &cRxBuffer[2]) //overflow e dati non corretti
    {
        cRxprt = &cRxBuffer[0]; //reinializzo puntatore e buffer
        cRxBuffer[0] = NULL;
    }
}
exit_int_rda: ; //uscita gestione interrupt Rx
}

#int_AD
void AD_isr() //segmento accessibile con interrupt A/D
{
    fResult = read_adc(); //interrupt AD e leggo dato a 10 bit
    lResult = ((long)fResult); //in decimal
    fResult = fResult / 1024 * 5.00; //in Volts per scala 0-5V (dato di tipo float, in
                        //virgola mobile)
    sADdone = TRUE; //conversione finita, dato disponibile
}

void main()
{
    setup_adc_ports(AN0_AN1_AN3); //inizializzazione registri, modalita funzionamento
    setup_adc(ADC_CLOCK_INTERNAL);
    setup_psp(PSP_DISABLED);
}

```

```

setup_spi(FALSE);
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256); //TMR0 abilit. con precaler 256 ma non usato
setup_timer_1(T1_DISABLED); //Timer 1 disabilitato
setup_timer_2(T2_DISABLED,0,1); //Timer2 disabilitato
output_float(RX); //metto il pin Rx in alta impedenza
cRxprt = & cRxBuffer[0]; //inializzo cRxprt puntando alla prima
//locazione del buffer cRxBuffer[0]

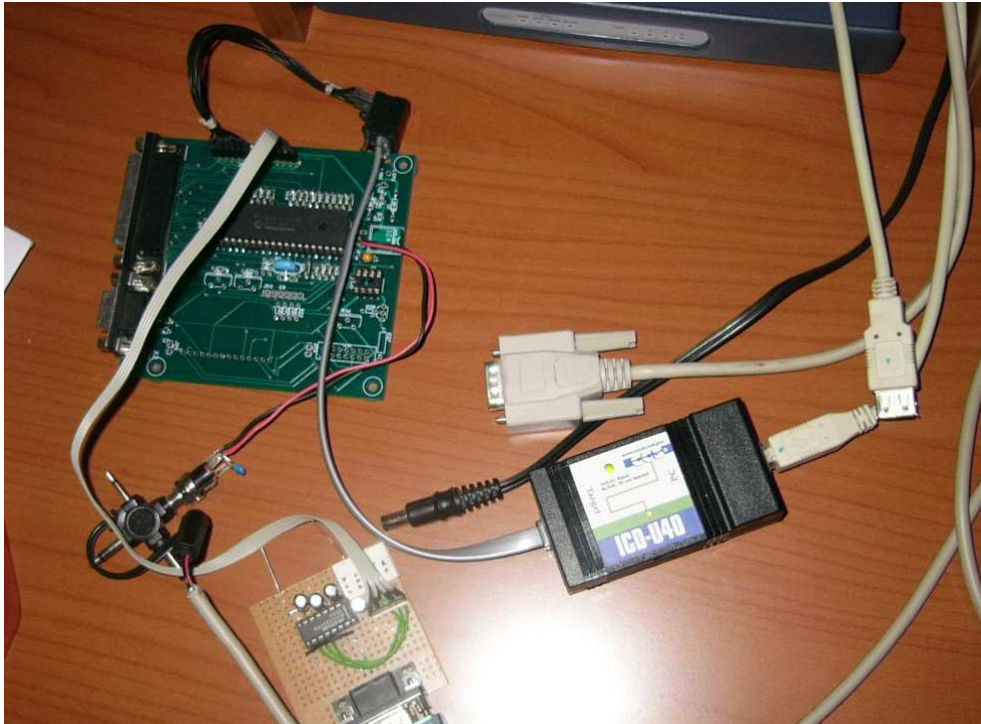
cRiferimenti = NULL;
cCount = NULL;
enable_interrupts(INT_RDA); //abilito l'interrupt in Rx e setto in INTCON
enable_interrupts(GLOBAL); //(registro interrupt) il bit d'interrupt GLOBAL

while(TRUE) // loop continuo
{
if(sSdone) //entro nel ciclo se ho ricevuto uno start
{
//per la conversione, quindi sSdone = 1
if(cRxBuffer[1] > '1')
{
printf("Canali di conversione max 2 ... Err!!!"); //se il canale di conversione > 2
delay_ms(200); //pausa di 200 ms
sSdone = FALSE;
continue;
}
disable_interrupts(INT_RDA); //disabilito interruzione in Rx, avvio
set_adc_channel(cRxBuffer[1]); //procedura per conversione, in attesa
enable_interrupts(INT_AD); //dell'interrupt di fine conversione
read_adc(ADC_START_ONLY);
sSdone = FALSE;
}
if(sADdone) //entro nel ciclo se la conversione è avvenuta
{
//e mando in Tx il dato in decimale e in Volts
printf("A/D conversione Decimale = %lu (ch%1x) \n\r", IResult, cRxBuffer[1]+ '1');
printf("A/D conversione Volts= Vcc %1.2f (ch%1x) \n\r", fResult, cRxBuffer[1]+ '1');
sADdone = FALSE;
delay_ms(100); //pausa di 100 ms
enable_interrupts(INT_RDA); //riabilito interruzione in Rx e riparto per
//un nuovo ciclo.
}
}
}
}

```

Ciò che ha consentito la programmazione del PIC16F874 è stato l'ICD.

Che cos'è l'ICD?



L'ICD (In Circuit Debugger) consente una facile programmazione e lettura, e opzionalmente il debug di programmi per alcuni micro-controllori. L'ICD non lavora con tutti i chip. Il chip deve avere l'ICSP per la programmazione cosicché l'ICD possa effettuare il debug.

L'ICD è collegato ai microcontrollori via MCLR (reset), B7, B6 e B3 (non necessario). Usando B6 e B7, l'ICD può scaricare programmi e comunicare con un modulo di debug nel microcontrollore. Collegare B3 è opzionale, in quanto è usato nel monitor di debug nel PCW (compilatore). Se B3 è collegato, non è usato nel programma essendo già in debug. Se la porta B è usata in un programma, mantenere 3 piedini inutilizzati.

L'unica limitazione rispetto ad un emulatore hardware sono quelle di utilizzare parte delle risorse interne (anche se poche) come qualche locazione di RAM dell'ultimo banco, e far uso di un solo breakpoints (punto di interruzione del firmware durante il debug).

L'ICD-U40 (...) si interfaccia al PC via USB.

Programma in Visual Basic 6

Di seguito viene riportato e commentato il listato del programma realizzato in Visual Basic.

Option Explicit

Dim h_com As Boolean 'dichiaro una variabile di tipo booleana per sapere se è andata
'a buon fine l'apertura della Com.

Dim stringa_in As String 'dichiaro una variabile di tipo string per appoggiare i dati letti dalla seriale

Private Sub cmd_start_Click()

Dim pos1 As Integer, pos2 As Integer 'variabili utilizzate per filtrare la posizione del dato letto
'dalla seriale e visualizzare la parte voluta (in abbinamento all' istruzione Instr)

On Error GoTo err_ser

Select Case modi_funz.ListIndex 'testo il modo scelto dalla combolist e mi comporto di
conseguenza

'Acquisizione continua - Case 0

'Acquisizione singola - Case1

'Acquisisci e memorizza val. decimali - Case2

'Acquisisci e memorizza in Volts - Case3

Case 0

If h_com Then 'se è aperta la Com, la chiudo, metto in off il led ecc.

MSComm1.PortOpen = False

Timer1.Enabled = False 'per sicurezza

cmd_start.Caption = "Start_lettura A/D"

h_com = False

com_list.Enabled = True

modi_funz.Enabled = True

Label1.Caption = "....disconnesso !"

log.Text = "Log..."

lamp.FillStyle = 1

Exit Sub

Else 'se è close apro la Com con i vari settaggi ed interrogo la pic
'chiedendo il valore convertito sul canale 1 o 2

set_ser

Label1.Caption = "....connesso !"

cmd_start.Caption = "Stop_lettura A/D":

com_list.Enabled = False

modi_funz.Enabled = False

Do While cmd_start.Caption = "Stop_lettura A/D": 'ciclo continuo sino a stoppare il tutto

If canale_1.Value Then

invia_tx ("0" & vbCr) 'richiedo conversione sul canale 1 (\$ = sincronismo 0 = 1°

canale)

Else

invia_tx ("1" & vbCr) 'richiedo conversione sul canale 2 (\$ = sincronismo 0 = 2°

canale)

End If

Call pausa(400) 'pausa di 400 ms tra una lettura e l'altra

ricevi_rx 'ricevo il dato dalla com

If stringa_in = "" Then 'se la stringa in ingresso dalla seriale è nulla, allora non ricevo

nulla

log.Text = "Nessun dato in ricezione !!"

If Not h_com Then log.Text = "Log..."

Else

```

        log.Text = stringa_in
    End If
    stringa_in = ""
    DoEvents          'cedo il passo ad altre applicazioni ed eventi
Loop
Exit Sub
End If
Case 1, 2, 3        'qui mi comporto come sopra, solo che l'acquisizione non è
                    continua e posso
                    'anche memorizzare i valori letti associati al tempo
    cmd_start.Enabled = False
    com_list.Enabled = False
    modi_funz.Enabled = False
    set_ser
    Label1.Caption = "...connesso !"
    If canale_1.Value Then
        invia_tx ("$0" & vbCr)
    Else
        invia_tx ("$1" & vbCr)
    End If
    Call pausa(400)
    ricevi_rx
    If stringa_in = "" Then
        log.Text = "Nessun dato in ricezione !!"

    Else
        log.Text = stringa_in
        If modi_funz.ListIndex = 2 Then 'qui adesso riempio le listbox con le conversioni e il
tempo associato
            pos1 = InStr(stringa_in, "Decimale = ")
            pos2 = InStr(stringa_in, "(ch)")
            list_mem.AddItem (Mid$(stringa_in, pos1 + 11, pos2 - (pos1 + 11))):
List_now.AddItem Now
        End If
        If modi_funz.ListIndex = 3 Then
            pos1 = InStr(stringa_in, "Vcc ")
            list_mem.AddItem Right("0.00" & (Mid$(stringa_in, pos1 + 4, 4)), 4):
List_now.AddItem Now
        End If
    End If
    If h_com Then MSComm1.PortOpen = False: h_com = False
    If Not stringa_in = "" And Not modi_funz.ListIndex = 1 Then log.Text = "OK valore
acquisito..."
    stringa_in = ""
    Label1.Caption = "...disconnesso !"
    cmd_start.Enabled = True
    com_list.Enabled = True
    modi_funz.Enabled = True
End Select
Exit Sub
err_ser:          'qui arrivo se c'è un errore sulla seriale, durante il settaggio o
l'apertura
    If h_com Then

```

```

MSComm1.PortOpen = False
h_com = False
End If
com_list.Enabled = True
modi_funz.Enabled = True
cmd_start.Enabled = True
Label1.Caption = "...disconnesso !"
lamp.FillStyle = 1 'metto in off il led
Screen.MousePointer = vbNormal
MsgBox "Errore seriale su " & com_list.Text & " !!", vbCritical, "Errore"
End Sub

```

```

Private Sub set_ser() 'routine di settaggio seriale e apertura com
With MSComm1
.Settings = "9600,n,8,2" 'baudrate 9600, nessuna parità, 8 bit di dato, 2 bit di stop
.Handshaking = comNone
.RThreshold = 1 'attiva evento di acquisizione dopo un solo byte ricevuto, serve solo per
attivare il lampeggio del led
.CommPort = com_list.ListIndex + 1
.PortOpen = True
End With
h_com = True
End Sub

```

```

Private Sub Form_Load() 'parte attivata all'apertura del programma, mi carico i dati impostati
precedentemente dal file didattica.ini
Dim tmp As String
On Error Resume Next
modi_funz.ListIndex = 0
com_list.ListIndex = 0
Open Trim(App.Path) + "\didattica1.ini" For Input As #1 'apertura file
Line Input #1, tmp 'lettura prima riga e settaggio della porta
selezionata
com_list.ListIndex = Cint(tmp)
Line Input #1, tmp 'lettura seconda riga e settaggio modo di
funzionamento
modi_funz.ListIndex = Cint(tmp)
Close #1 'chiudo file
End Sub

```

```

Private Sub Form_Unload(Cancel As Integer) 'parte attivata alla chiusura del programma e
operazione inversa in form load, in pratica
'qui salvo i dati nel file didattica.ini
If MSComm1.PortOpen = True Then MSComm1.PortOpen = False
Open Trim(App.Path) + "\didattica1.ini" For Output As 1 'apertura file
Print #1, com_list.ListIndex 'scrittura prima riga (porta seriale selezionata)
Print #1, modi_funz.ListIndex 'scrittura seconda riga (modo funzionamento
selezionato)
Close #1 'chiudo file ed esco dal programma
Unload Me
End
End Sub

```

```
Private Sub list_mem_Click()  
List_now.ListIndex = list_mem.ListIndex  
End Sub
```

```
Private Sub list_mem_DbClick()  
list_mem.Clear: List_now.Clear  
End Sub
```

```
Private Sub List_now_Click()  
list_mem.ListIndex = List_now.ListIndex  
End Sub
```

```
Private Sub List_now_DbClick()  
list_mem.Clear: List_now.Clear  
End Sub
```

```
Private Sub log_Change()  
log.SelStart = Len(log.Text)  
End Sub
```

```
Private Sub log_DbClick()  
log.Text = "Log..."  
End Sub
```

```
Private Sub log_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)  
If h_com Then Screen.MousePointer = vbHourglass  
End Sub
```

```
Private Sub MSComm1_OnComm() 'gestione eventi prodotti da MSComm  
Select Case MSComm1.CommEvent  
Case comEvReceive 'unico evento utilizzato  
If Timer1.Enabled = False Then  
lamp.FillStyle = 0 'accendo led con durata lampeggio associato al timer1  
Timer1.Enabled = True  
End If  
End Select  
End Sub
```

```
Private Sub ricevi_rx()  
If h_com Then stringa_in = MSComm1.Input  
End Sub
```

```
Private Sub invia_tx(ByRef dato As String)  
If h_com Then MSComm1.Output = dato  
End Sub
```

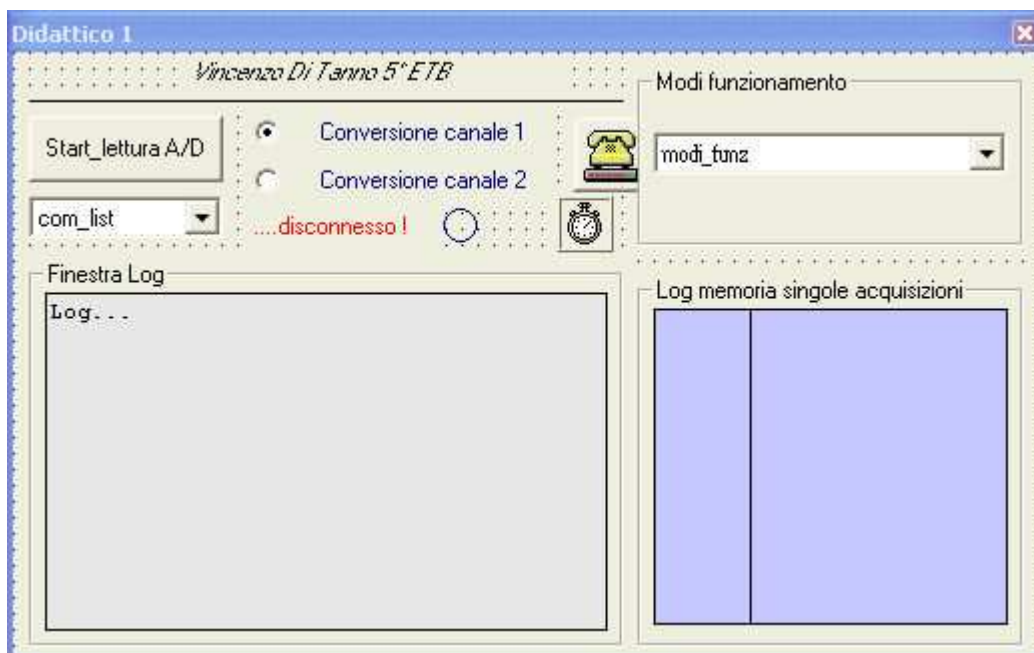
```
Private Sub Timer1_Timer() 'routine attivata per generare il ritardo del lampeggio led  
lamp.FillStyle = 1  
Timer1.Enabled = False  
End Sub
```

```

Private Sub pausa(ms As Long) 'routine di generazione pausa
Dim t As Single
t = Timer 'definisco un timer ed esco quando t = ms passato come argomento
Do While Timer < t + (ms / 1000) 'sino a che t < ms resto nel loop
DoEvents 'cedo il passo ad altre applicazioni ed eventi
Loop
End Sub

```

Di seguito è riportato il Form Load e una spiegazione dettagliata dei principali oggetti usati per il corretto funzionamento del programma.



Controllo MSComm (comunicazioni)

Il controllo **MSComm** (comunicazioni) fornisce all'applicazione funzioni per le comunicazioni seriali, consentendo la trasmissione e la ricezione di dati tramite una porta seriale.

Sintassi

MSComm

Osservazioni

Il controllo **MSComm** consente di gestire le comunicazioni nei due modi seguenti:

- La gestione delle comunicazioni tramite eventi è molto efficace per il controllo delle interazioni tra porte seriali. In molte situazioni può essere utile ricevere un segnale nel momento in cui viene generato un determinato evento, quale l'arrivo di un carattere o una modifica sulle linee CD (Carrier Detect) o RTS (Request To Send). In tali casi, l'evento OnComm del controllo **MSComm** consente di intercettare e gestire gli eventi di

comunicazione. L'evento OnComm consente inoltre di individuare e gestire gli errori di comunicazione. Per un elenco di tutti gli eventi e gli errori di comunicazione, vedere la proprietà **CommEvent**.

- Il polling di eventi ed errori può essere inoltre eseguito controllando il valore della proprietà **CommEvent** dopo l'esecuzione di ogni funzione critica del programma. Questo metodo è consigliato nel caso di applicazioni autonome e di dimensioni ridotte. Se ,ad esempio, si sta scrivendo un semplice programma di composizione di numeri telefonici, potrebbe non essere necessario generare un evento dopo la ricezione di ciascun carattere, in quanto il solo carattere atteso è il segnale OK di risposta del modem.

Ogni controllo **MSComm** utilizzato corrisponde a una porta seriale. Per accedere a più porte seriali con l'applicazione, sarà necessario utilizzare più controlli **MSComm** . L'indirizzo di porta e di interrupt può essere modificato tramite il Pannello di controllo di Windows.

Per il controllo **MSComm** sono disponibili diverse proprietà importanti. Di seguito sono riportate le proprietà fondamentali.

Proprietà	Descrizione
CommPort	Imposta e restituisce il numero della porta di comunicazioni.
Settings	Imposta e restituisce in forma di stringa la velocità, la parità, i bit di dati e i bit di stop.
PortOpen	Imposta e restituisce lo stato di una porta di comunicazioni, oltre ad attivare e disattivare la porta.
Input	Restituisce e rimuove caratteri dal buffer di ricezione.
Output	Scrive una stringa di caratteri nel buffer di trasmissione.