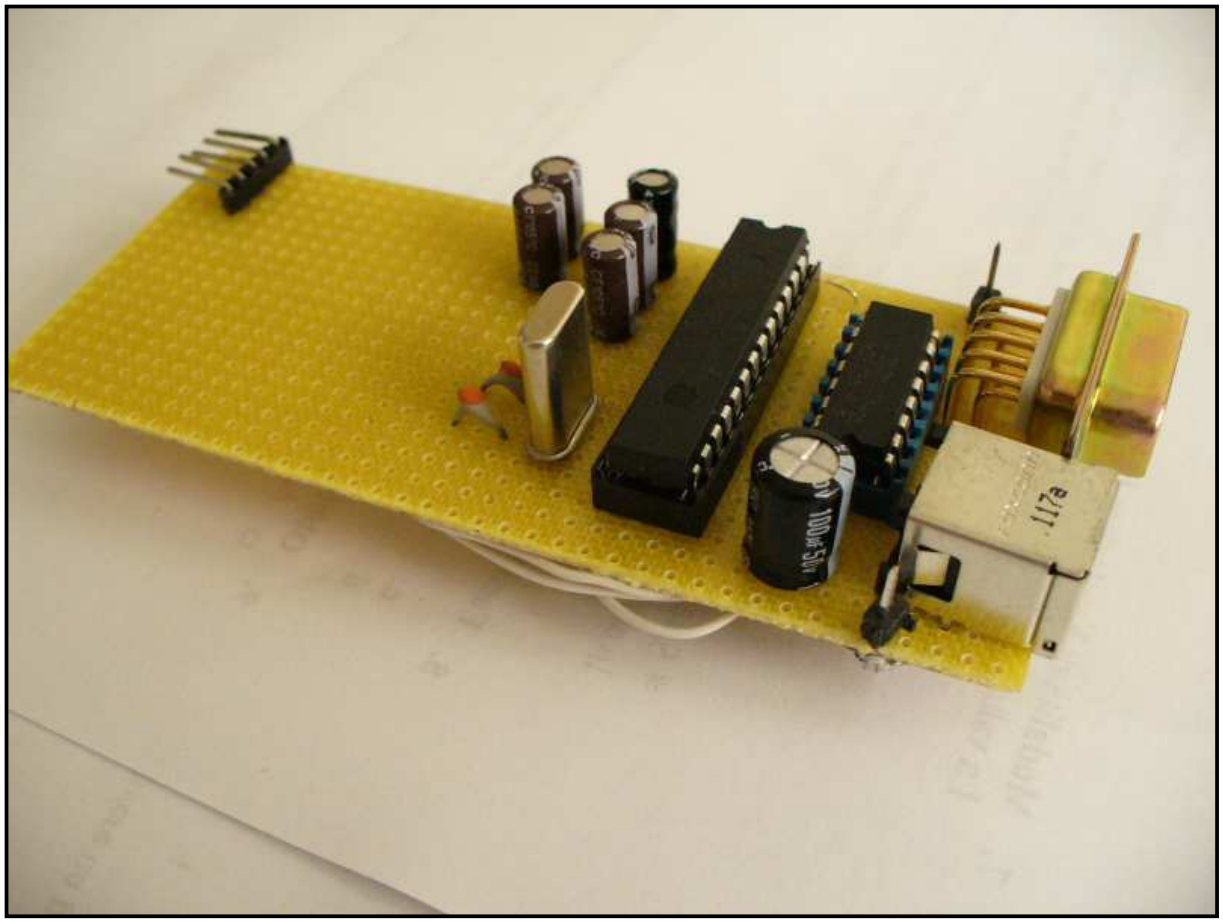
Sistema di allerta centralizzato con collegamento seriale

Relazione Tecnica

Castellani Riccardo
Chiodaroli Riccardo
Gatti Massimo
Paiola Daniel
Tramelli Roberto

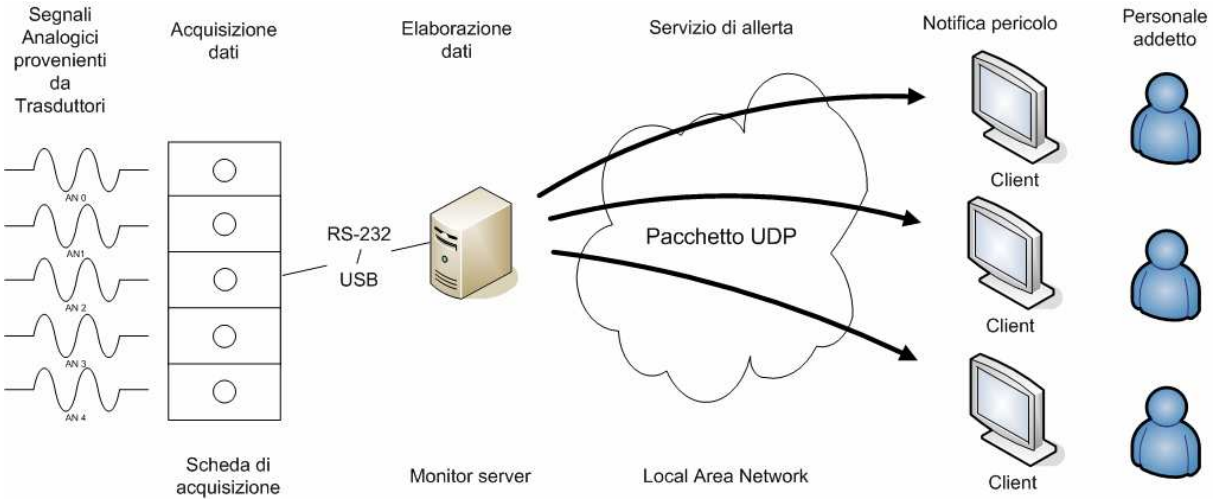
5 Informatica 1
A.S. 2005/2006
ISII Tecnico "G. Marconi" Piacenza



Descrizione del Progetto

Il progetto nasce dall'esigenza ormai comune di fornire soluzioni idonee a garantire la sicurezza dell' individuo, studente o lavoratore che sia, nel proprio ambiente di lavoro o di studio. In quest'ottica, l'esperienza si pone come obiettivo l'implementazione di un sistema completo per il controllo remoto dei locali, che permetta di analizzare parametri ambientali come temperatura, presenza di fumo, livello di ossigeno ed altri gas o più semplicemente la presenza di persone all'interno del locale, ed di avvisare il personale addetto in caso di pericolo.

Per ragioni di tempo, la soluzione da noi concepita si è limitata al controllo delle temperature di 5 locali, ma abbiamo comunque realizzato una scheda di acquisizione scalabile, in grado di supportare svariati tipi di segnali. Senza contare, poi, che l'applicativo software di nostra concezione è indipendente dal sistema hardware utilizzato e perciò adatto alle situazioni più diverse.



Lo schema qui sopra sintetizza efficacemente il funzionamento del sistema:

1. La **scheda di acquisizione** è un dispositivo dotato di cinque *trasduttori*, componenti che rilevano le temperature dei locali e che producono in uscita una tensione proporzionale alla temperatura stessa.

2. I **dati** così acquisiti vengono campionati e digitalizzati da un *microcontrollore* (un sistema a logica programmabile), che li invia ad un *monitor server*, un computer di controllo, cioè, configurato con il software necessario all'*interfacciamento* della scheda.

3. Il ***monitor server*** riceve i dati acquisiti dalla scheda e traccia un grafico con l'andamento delle temperature in ogni ambiente. In più, permette all'utente di impostare un *range* di temperature accettabili per quello specifico locale. In questo modo, qualora uno dei valori misura ecceda i limiti imposti, il server diramerà in rete un messaggio di avviso, utile per informare il personale addetto della situazione di pericolo.

4. Il ***client***, quindi, mostra al personale addetto alla sicurezza un prospetto riassuntivo delle temperature presenti nei diversi locali, ponendo particolare rilevanza su quelle fuori norma.

Parte Teorica

Il PIC

I **PIC** (*Programmable Interrupt Controller*) sono componenti integrati, prodotti dalla *Microchip Technology Inc.*, che appartengono alla categoria dei microcontroller, ovvero soluzioni che incorporano in un unico dispositivo tutti i circuiti necessari a realizzare un completo sistema digitale programmabile. Si presentano esternamente come dei normali circuiti integrati TTL o CMOS, ma internamente possiedono una struttura molto simile ad un sistema a microprocessore, ovvero:

- Una **CPU** (**C**entral **P**rocessing **U**nit ovvero unità centrale di elaborazione) il cui scopo è interpretare le istruzioni di programma contenute della EEPROM ed eseguirle.

- Una memoria **EEPROM** (**E**lectronically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory ovvero memoria programmabile a sola lettura cancellabile elettronicamente) in cui sono memorizzare in maniera permanente le istruzioni del programma da eseguire. Questa memoria può però essere cancellata per ripetere la *programmazione* del PIC.

- Una memoria **RAM** (**R**andom **A**ccess **M**emory ovvero memoria ad accesso casuale) è invece una memoria *volatile*, nel senso che mantiene le informazioni contenute al suo interno soltanto quando il componente è in tensione. Per questo viene impiegata dalla CPU per salvare le variabili utilizzate dal programma.

- Una serie di **LINEE DI I/O (Input / Output**, ingresso e uscita**)**, utili per ricevere segnali da sensori, pulsanti, etc… e per pilotare dispositivi ed elementi esterni al microcontrollore, come ad esempio un sistema di ventilazione o una campanella di allarme.

- Una serie di dispositivi ausiliari indispensabili al funzionamento stesso del PIC, quali generatori di clock, bus, registri, etc…

La presenza di tutti questi dispositivi in uno spazio estremamente contenuto, consente al progettista di avvalersi degli enormi vantaggi derivanti dall'uso di un sistema a microprocessore, anche in quei circuiti che fino a poco tempo fa erano destinati ad essere realizzati con circuiterie tradizionali.

I **PIC** sono disponibili in un'ampia gamma di modelli, per meglio adattarsi alle esigenze di progetto specifiche, differenziandosi per numero di linee di I/O, per prestazioni, funzioni aggiuntive, etc… Si parte dai modelli più piccoli identificati dalla sigla **PIC12C5xx** dotati di soli 8 pin, fino ad arrivare ai modelli più grandi con sigla **PIC17Cxx** dotati di 40 pin. La foto qui sotto presenta 4 microcontrollori della serie PIC16F e rispettivamente, dall'alto verso il basso: PIC16F84, PIC16F870, PIC16F876 e PIC16F877. Come si può facilmente notare, le dimensioni e il numero di pin aumentano rapidamente nel passaggio dai modelli più elementari (16F84) a quelli più sofisticati (16F877).

Una descrizione dettagliata dei singoli modelli è disponibile presso il sito web della Microchip raggiungibile via Internet, che consigliamo senz'altro di esplorare per la grande quantità di informazioni tecniche, software di supporto, esempi di applicazioni e aggiornamenti disponibili.

PIC 16F870

Come abbiamo già accennato, esistono innumerevoli tipi di PIC, ciascuno con le proprie caratteristiche specifiche. Dopo alcune analisi preliminari, abbiamo però optato per il modello 16F870, per una serie di motivi: prima di tutto, si trattava di un integrato a "soli" 28 pin, contro i modelli a 40 di altri modelli più sofisticati, quindi per la presenza di 5 ingressi analogici, collegati ad un ADC 10 bit e in ultimo (ma non certo per rilevanza) per le uscite RS232C.

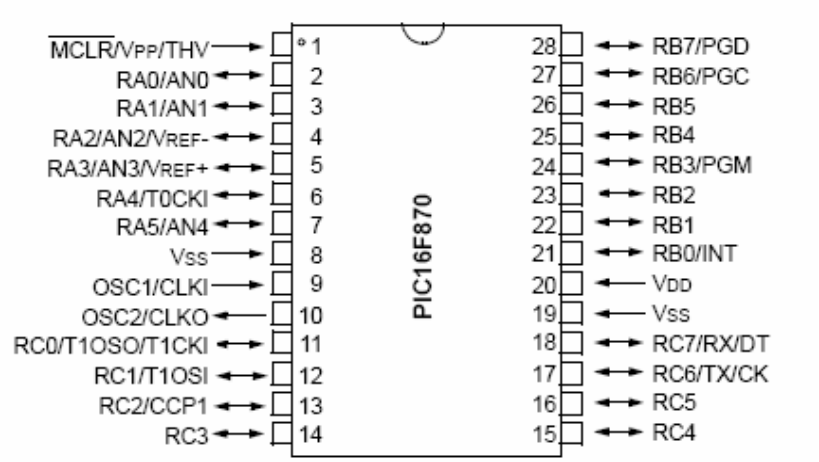
Caratteristiche salienti

Key Features PICmicro™ Mid-Range MCU Family Reference Manual (DS33023)	PIC16F870	PIC16F871
Operating Frequency	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	2K	2K
Data Memory (bytes)	128	128
EEPROM Data Memory	64	64
Interrupts	10	11
I/O Ports	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3
Capture/Compare/PWM modules	1	1
Serial Communications	USART	USART
Parallel Communications	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels
Instruction Set	35 Instructions	35 Instructions

La tabella qui sopra è un estratto del *datasheet* ufficiale del PIC16870 e 71, ed illustra le caratteristiche più rilevanti dei due dispositivi. Nel nostro caso, interessa la seconda colonna: innanzitutto, il microcontrollore in questione è in grado di funzionare ad una frequenza massima di 20 MHz, il che significa, in linea di massima, che la CPU interna esegue una istruzione ogni 50 ns; sono poi elencate le specifiche per la memoria EEPROM e RAM. La prima ha una dimensione di 64

Kilobyte, più che sufficiente per il *firmware* necessario all'acquisizione e all'invio dei dati. La seconda, invece, può contenere fino a 2000 parole (*word*) di 14 bit, quindi 3500 byte. Si passa quindi ai vari componenti aggiuntivi, dal numero di linee di IO, 3 nel caso del PIC16F870 e 5 con il PIC16F871, la tipologie delle interfacce di comunicazione (USART, seriale, o PSP, parallela), al numero di linee analogiche che varia dalle 5 nel PIC16F870 alle 8 sul PIC16F871. Infine il set di istruzioni, ossia il "dizionario", l'insieme di comandi accettati è 35 in entrambi i PIC. Si tratta invero di un set piuttosto limitato, che permette però un apprendimento più veloce e che non sminuisce le funzionalità del PIC. Tra l'altro, trattando di una CPU RISC, la poca varietà delle istruzioni è una caratteristica fondamentale. Queste CPU, infatti, privilegiano la velocità di esecuzione delle singole istruzioni, riducendo quindi il numero di cicli di clock necessari al loro completamento.

Pinout



Riportiamo qui la distinta completa dei pin del PIC 16F870. Vogliamo però focalizzare l'attenzione su alcuni particolari: innanzitutto vanno osservati le prime due voci in elenco, il pin 9 e il 10, che fungono da ingressi per il temporizzatore (clock) indispensabile al funzionamento del clock (se ne vedrà meglio l'uso in seguito).

(*segue dopo la tabella*)

TABLE 1-1: PIC16F870 PINOUT DESCRIPTION

Pin Name	DIP Pin#	SOIC Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI	9	9	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKO	10	10	O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, the OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP/THV	1	1	I/P	ST	Master Clear (Reset) input or programming voltage input or High Voltage Test mode control. This pin is an active low RESET to the device.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2	I/O	TTL	RA0 can also be analog input 0.
RA1/AN1	3	3	I/O	TTL	RA1 can also be analog input 1.
RA2/AN2/VREF-	4	4	I/O	TTL	RA2 can also be analog input 2 or negative analog reference voltage.
RA3/AN3/VREF+	5	5	I/O	TTL	RA3 can also be analog input 3 or positive analog reference voltage.
RA4/T0CKI	6	6	I/O	ST/OD	RA4 can also be the clock input to the Timer0 module. Output is open drain type.
RA5/AN4	7	7	I/O	TTL	RA5 can also be analog input 4.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	21	21	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	22	22	I/O	TTL	
RB2	23	23	I/O	TTL	
RB3/PGM	24	24	I/O	TTL/ST(1)	RB3 can also be the low voltage programming input.
RB4	25	25	I/O	TTL	Interrupt-on-change pin.
RB5	26	26	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	27	27	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	28	28	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
					PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	11	11	I/O	ST	RC0 can also be the Timer1 oscillator output or Timer1 clock input.
RC1/T1OSI	12	12	I/O	ST	RC1 can also be the Timer1 oscillator input.
RC2/CCP1	13	13	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/ PWM1 output.
RC3	14	14	I/O	ST	
RC4	15	15	I/O	ST	
RC5	16	16	I/O	ST	
RC6/TX/CK	17	17	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	18	18	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
VSS	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = input/output P = power
OD = Open Drain — = Not used TTL = TTL input ST = Schmitt Trigger input

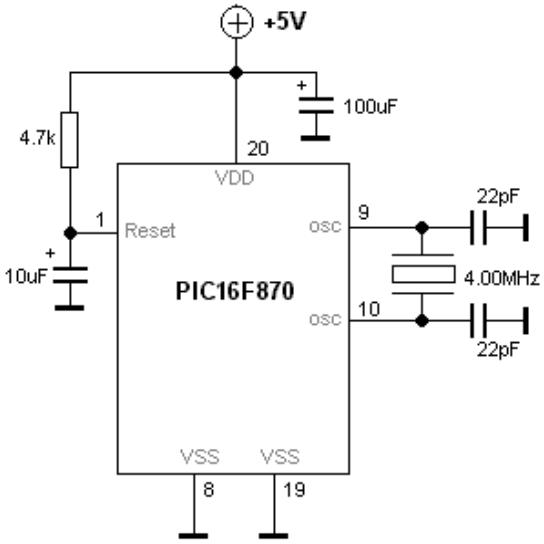
Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt or LVP mode.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

Veniamo ora al pin 1. Questa connessione è il fulcro della logica di programmazione del PIC, poiché riceve la tensione necessaria a cancellare la EEPROM e permette di memorizzare il numero eseguibile inviato al programmatore.

Troviamo poi tre porte bidirezionali, marcate A, B e C che hanno svariate funzionalità in base al contesto:

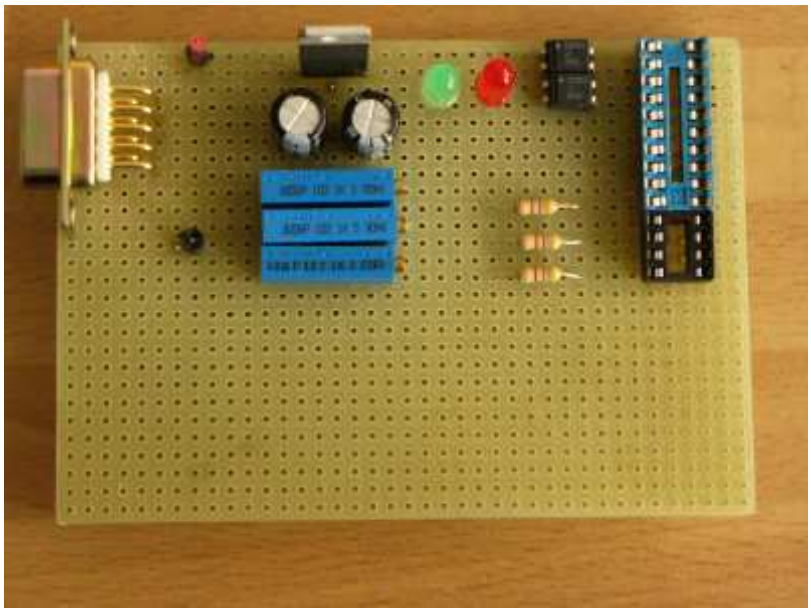
- Interscambio di dati, sia in uscita sia in ingresso con altri componenti circuitali.
- PORTA può anche essere utilizzata come input analogico (ma occorre prima impostare correttamente alcuni registri).
- PORTB predispone funzionalità di interrupt necessari al debug online (sotto tensione) del dispositivo.
- PORTC fornisce infine il pinout necessario alla comunicazione seriale.

Schema base di funzionamento

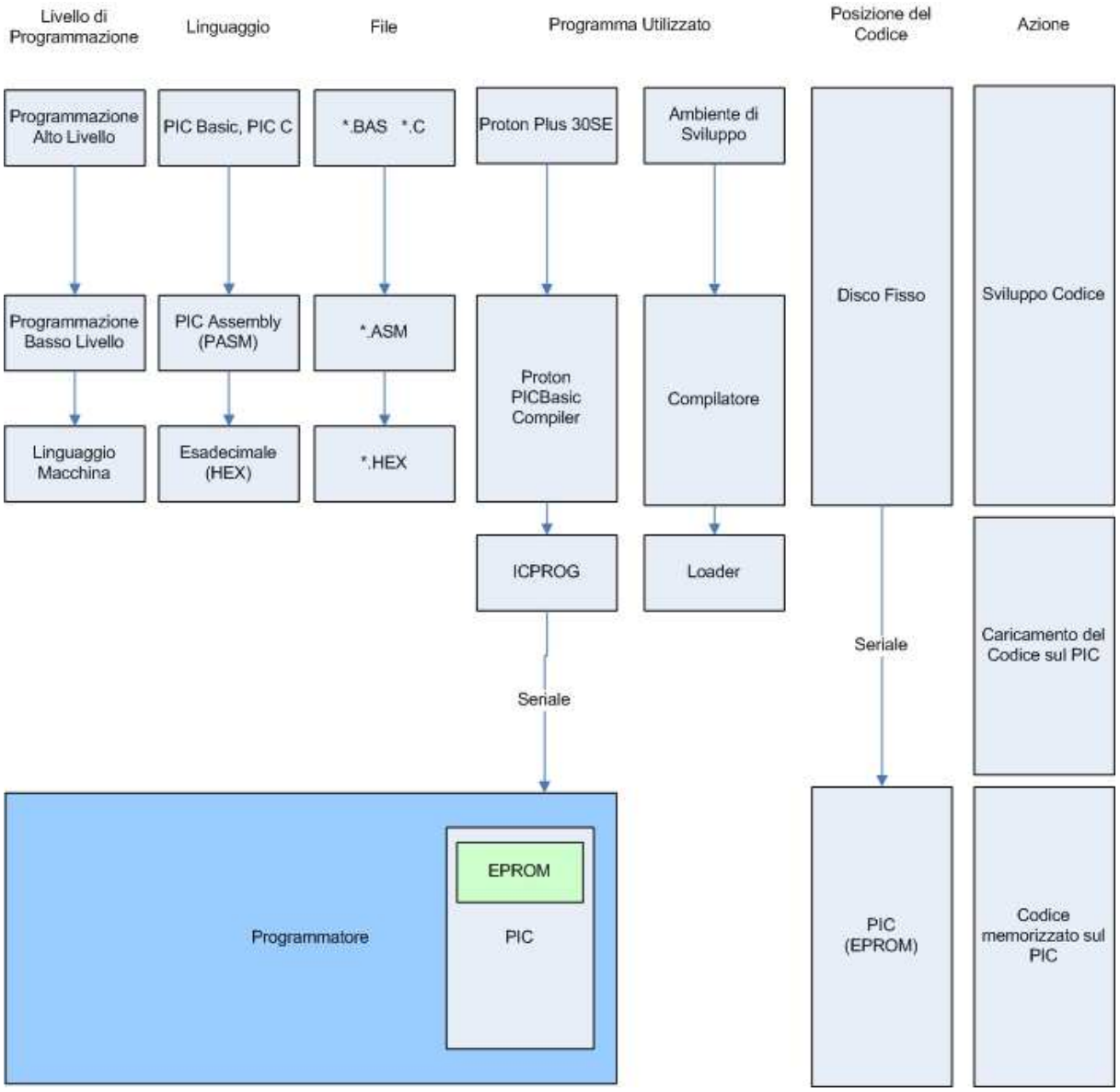


Lo schema qui illustrato rappresenta la *configurazione minima* necessaria al funzionamento del PIC16870. Ricordando i pin 9 e 10 appena illustrati, questi vengono collegati ad un sistema oscillante al quarzo, con relativi condensatori (i quali, per la ridottissima capacità, sono costruiti con il tantalio). L'ingresso di RESET è invece posto in uno stato "intermedio", come non collegato, vista la sua funzione di MASTER CLEAR. Il condensatore da 100 µF è utile, infine, per stabilizzare la tensione di alimentazione.

Il Programmatore



Schema logico



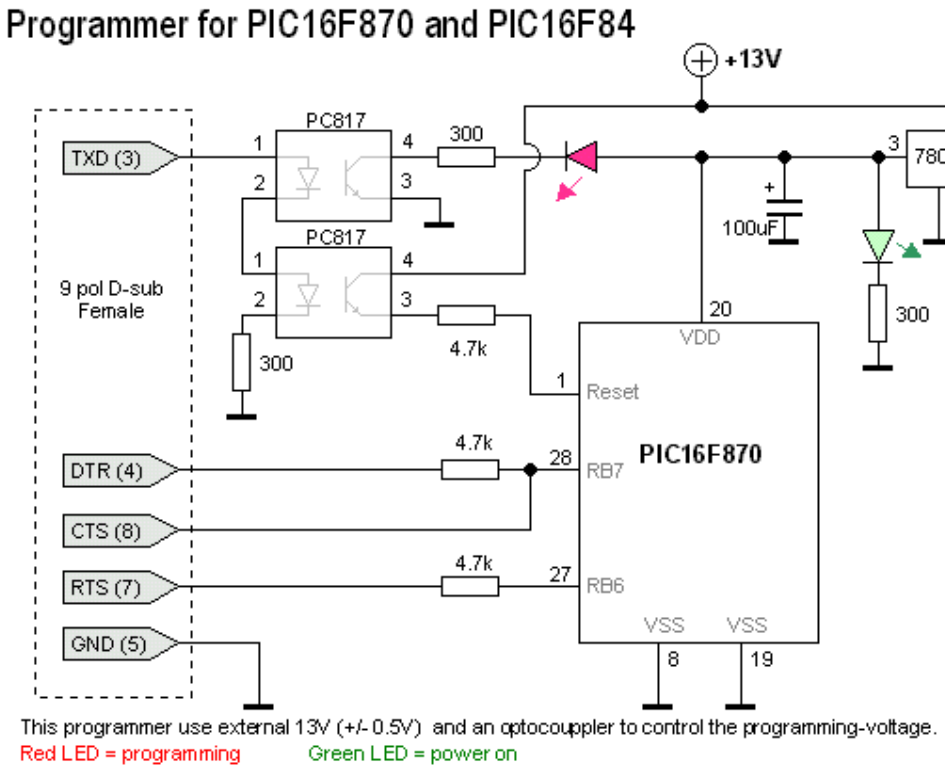
Lo schema qui sopra illustra a livello concettuale il funzionamento di un programmatore per microcontrollori. La struttura si sviluppa lungo due assi, uno verticale, che rappresenta il *livello di programmazione*, ed uno orizzontale che considera i diversi aspetti e le diverse fasi del processo.

Scorreremo il grafico da sinistra verso destra e dall'alto verso il basso. Partiamo quindi dal primo punto, la scrittura del codice: di norma, occorrerebbe sviluppare il firmware richiesto in modo specifico a seconda del PIC utilizzato, conoscendone appieno i registri e le caratteristiche. Con un linguaggio come PICBasic o C, invece, il livello di programmazione viene "spostato" dal livello "fisico" al quello "logico" o funzionale, per cui non è più indispensabile tenere in considerazione il microcontrollore scelto ed è invece possibile concentrarsi sulle funzionalità da sviluppare. A questo strato, o *layer*, la progettazione avviene su PC, per mezzo di *IDE* (*Integrated Development Enviroment*, ambienti di sviluppo integrati) come *Proton*. Questi applicativi producono file di tipo diverso a seconda del linguaggio scelto (*bas, c, …*) salvandoli sul disco fisso del computer.

Scendendo di un livello, si arriva ad uno strato intermedio tra l'hardware, il PIC, e il PC. Qui il codice considerato non è più di alto livello, ma diviene specifico per mezzo di un'operazione nota, la *compilazione* del sorgente, che viene effettuata da appositi *tool*, in grado di trasformare un set di istruzioni di alto livello in blocchi di codice di basso livello, specifichi a seconda del modello di PIC scelto. Eventuali problemi di incompatibilità (memoria insufficiente, funzionalità non disponbili, etc…) emergono in questo stadio e vanno risolti prima di proseguire.

Quando anche questa fase è stata superata, occorre memorizzare il codice macchina sulla memoria EEPROM del PIC. Prima di procedere, però, è indispensabile convertire il formato *assembler* in uno adatto alla memoria del microcontrollore, cioè il formato *esadecimale*. È ora la volta del *loader*, un software in grado di inviare correttamente, tramite un'interfaccia comune (come RS232, PSP o USB) i dati al PIC, pilotando un apposito dispositivo hardware in grado di accettare il codice compilato e di memorizzarlo correttamente nella EEPROM.

Schema Circuitale



Come si può facilmente notare, il blocco circuitale di sinistra rappresenta il connettore DB-9 seriale, cui verranno trasmessi i dati. I pin 27 e 28 rappresentano rispettivamente il *clock serial* e il *serial data input* del PIC in fase di programmazione, e sono perciò connessi: alla linea RTS (*Request-To-Send*) della seriale il primo, e al DTR (*Data Terminal Ready*) e CTS (*Clear-To-Send*) il secondo. In quest'ultima circostanza, il pin DTR è usato per ricevere i dati veri e propri, mentre il CTS è necessario per indicare al controller seriale del computer di iniziare la trasmissione.

La parta alta dello schema, invece, si attiva quando si ha un potenziale alto sul pin 3 del connettore: il primo *disaccoppiatore ottico* chiude il collegamento a massa del LED rosso (indicando così lo stato "programmazione in corso"), ed abilitando a catena il secondo PC817. Questi chiude invece il collegamento tra i potenziale di alimentazione +13V e il pin 1 del PIC, usato per fornire la tensione di programmazione.

Precisiamo infine che il dispositivo 7805 è uno stabilizzatore di tensione, il quale, in combinazione con i due condensatori (richiesti dal suo schema di utilizzo tipico), fornisce in uscita

+5V esatti, necessari ad alimentare il PIC e ad attivare i diodi LED (il LED verde, in particolare, si illumina ogni volta che viene fornita tensione al circuito).

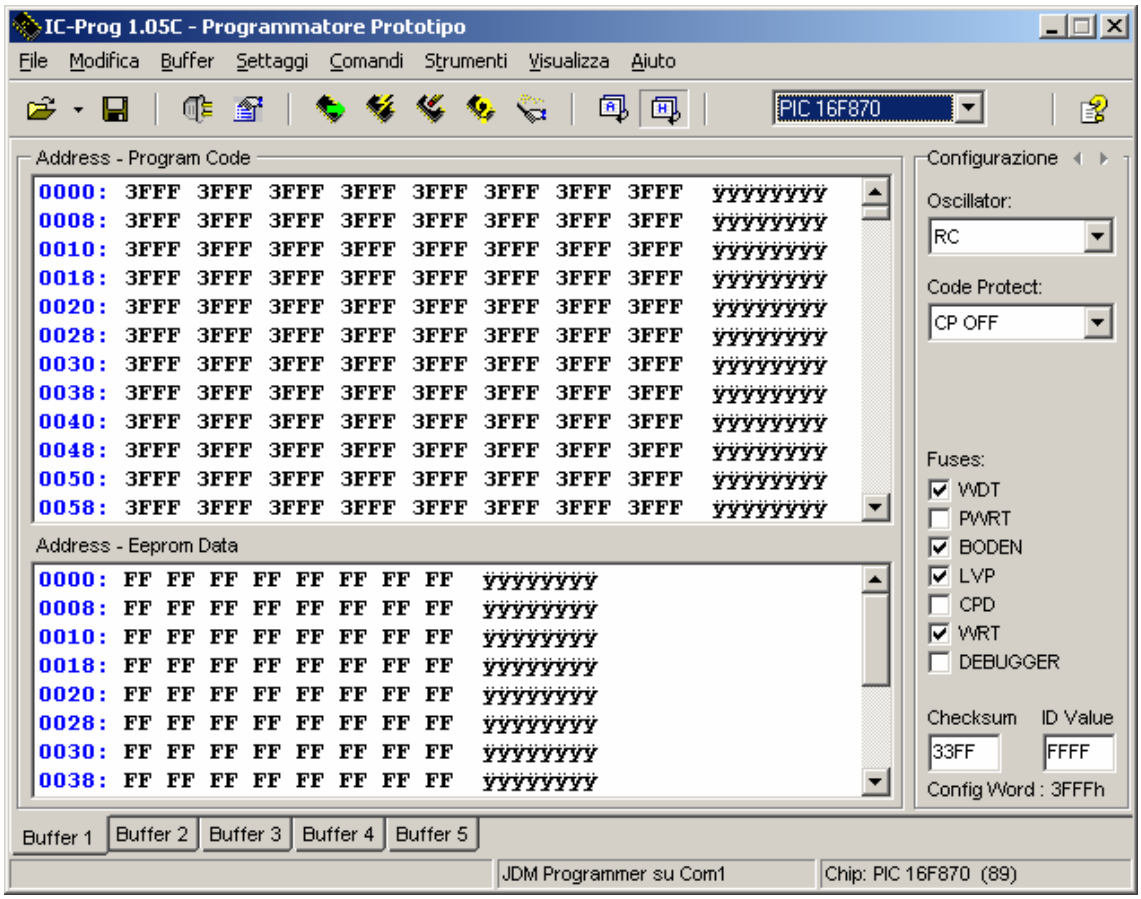
Anche in questa circostanza, le difficoltà sono state non poche. Innanzitutto è stato ostico reperire tutti i componenti necessari, dai *disaccoppiatori ottici (PC817)* allo *stabilizzatore di tensione (7805)* e al connettore della *porta seriale (DB-9)*. In quest'ultimo caso, poi, l'allievo Gatti ha dovuto adattare un cavo esistente, rimuovendo i collegamenti già presenti ed effettuando una operazione di saldatura per i nuovi.

Dovendo poi attendere la consegna degli integrati e non potendo provare il circuito appena realizzato, è stato deciso di trasferirlo su una *millefori* mediante la tecnica della *wrappatura*. Essendo novizi alla tecnica, è stato necessario un po' di tempo per apprenderne il meccanismo, ma alla fine il risultato ottenuto è stato soddisfacente, specialmente in virtù della migliore organizzazione dei componenti sulla scheda e dei collegamenti fra questi.

Il Loader (ICPROG)

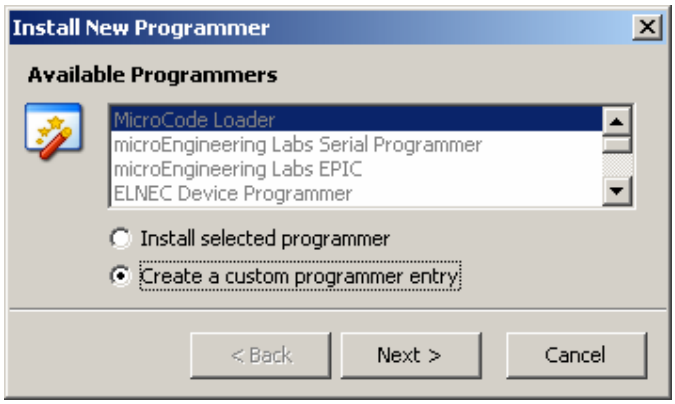
Abbiamo parlato del funzionamento del programmatore dal punto di vista elettrico. Vediamo ora quanto concerne la parte software. È necessario infatti un applicativo in grado di ricevere il file esadecimale compilato ed inviarlo al programmatore su una specifica porta seriale, garantendo fra l'altro anche la correttezza dell'operazione (*verifica* del codice). In termini tecnici, questa utility prende il nome di *loader*.

L'IDE di ProtonDS include già alcuni loader, ma uno dei più utilizzati ed efficienti è certamente ICPROG. ICPROG è dotato di numerose funzioni: programmazione, verifica e cancellazione totale di dispositivi programmabili (l'elenco dei componenti supportati è notevole) su più linee contemporaneamente, supporto alla protezione del codice e a numerose altre impostazioni che lo rendono compatibile con una vasta gamma di circuiti di programmazione.

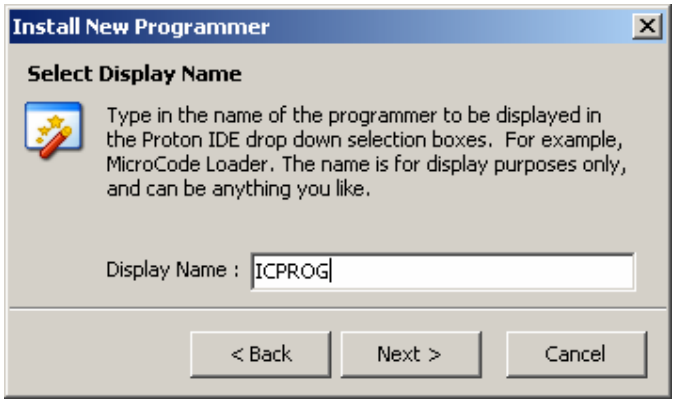


Installazione di ICPROG su ProtonDS

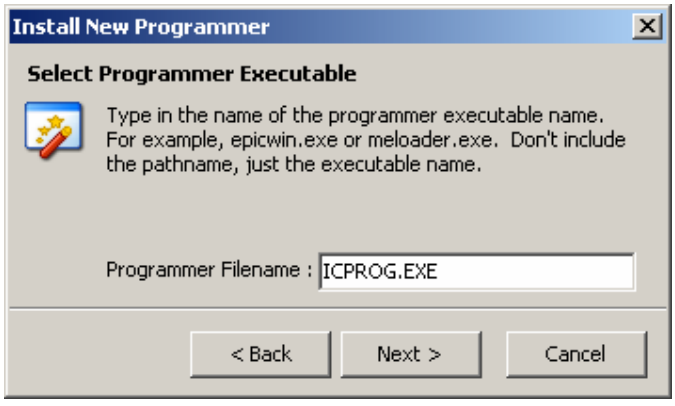
Per prima cosa è necessario selezionare *Install New Programmer* dal menu di ProtonDS.



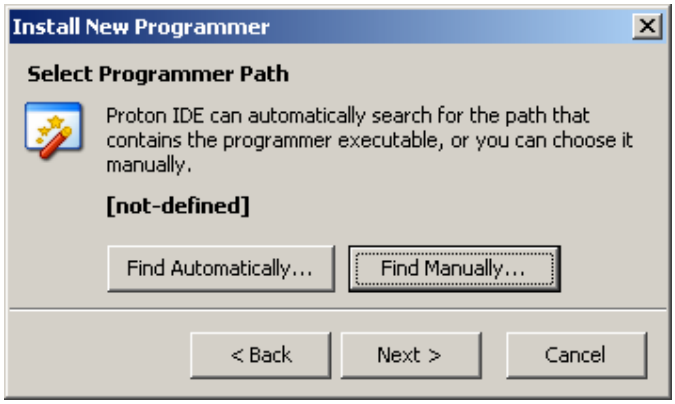
Verrà avviata una procedura guidata. Selezionare innanzitutto la seconda opzione, *Create a custom program entry*.



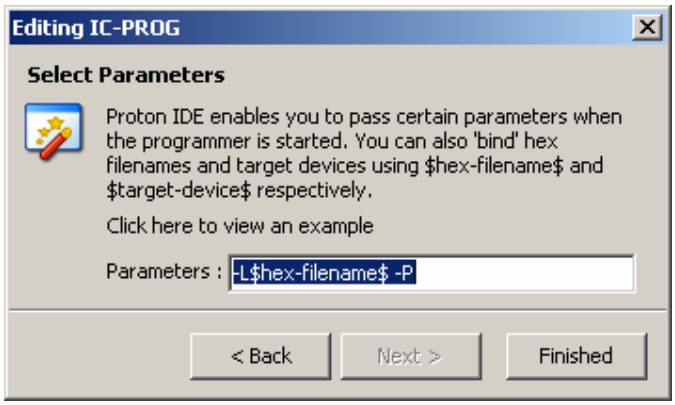
Specificare ora il nome da mostrare per il nuovo loader. Sarebbe opportuno scrivere sempre *ICPROG*.



Viene adesso richiesto di immettere il nome del file eseguibile del loader. Specificare esattamente *ICPROG.EXE*.



È necessario impostare anche il percorso su disco dell'eseguibile del loader. Consigliamo di premere *Find Manually* ed indicarlo manualmente.



In ultimo, replicare la stessa stringa mostrata in figura nell'ultimo passaggio. Rappresenta infatti la *stringa* da passare a riga di comando a ICPROG ed è fissa.

Impostazioni per ICPROG su sistemi Windows NT/2000/XP/2003

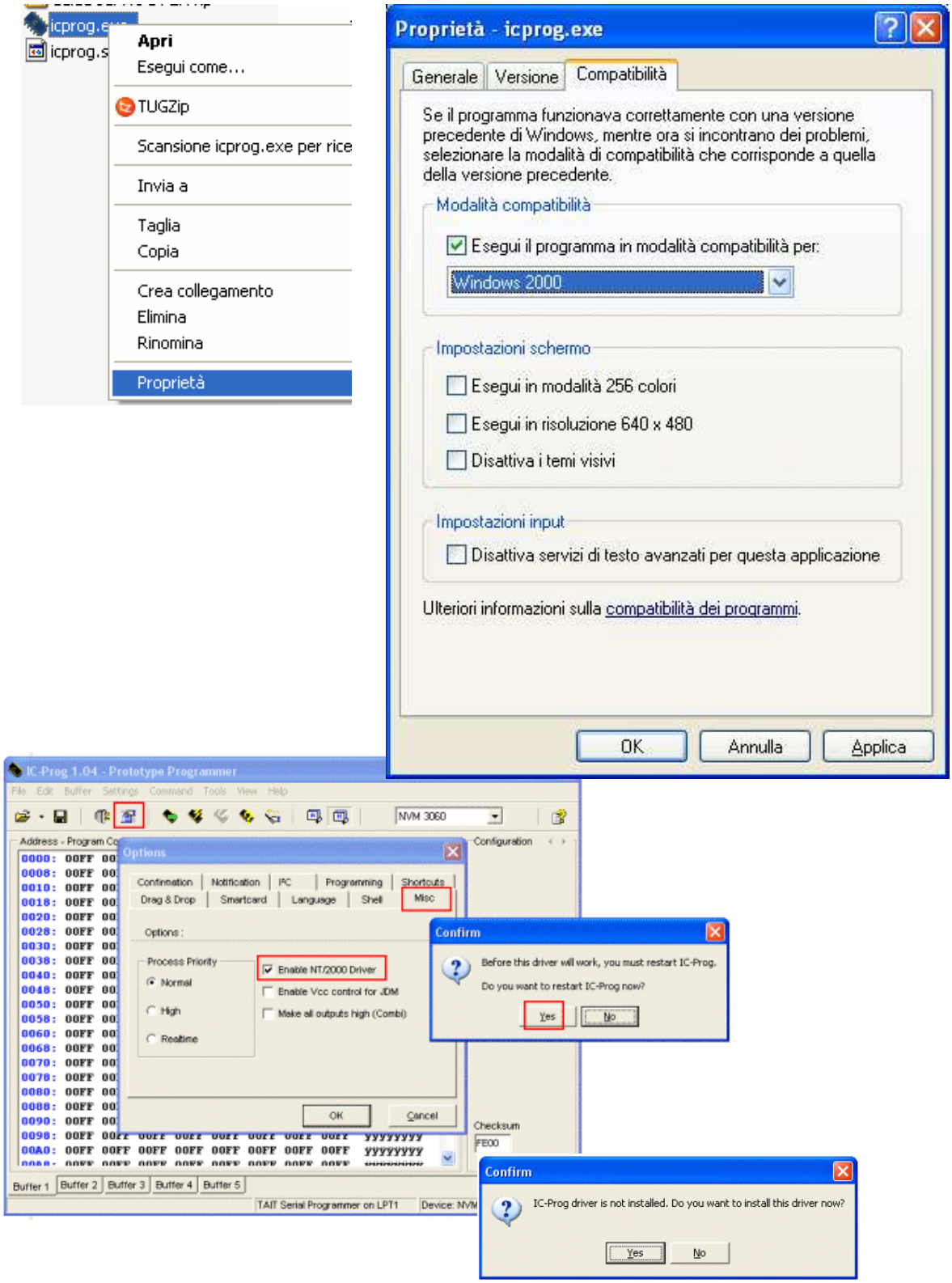
Attenzione: se sulla Macchina è installato un sistema Windows NT/2000/XP/2003, ossia un sistema operativo con kernel NT, in cui è attiva la protezione della memoria, è necessario seguire la seguente procedura per rendere ICPROG compatibile con il sistema operativo (si dispone di Windows NT/2000, saltare al passo 3).

1. Aprire una finestra di Esplora Risorse, raggiungere la cartella di ICPROG e aprirne la finestra *Proprietà* (click destro oppure ALT + INVIO).

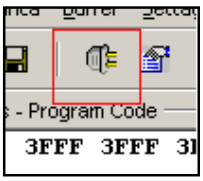
2. Dalla scheda *Compatibilità*, selezionare *Windows 2000* nella casella a discesa. Confermare e chiudere.

3. Avviare ICPROG. Verrà mostrato un messaggio di errore. Ignorarlo ed accedere alla finestra *Settaggi / Settings* (vedi figura). Selezionare la scheda *Misc*, e mettere un segno di spunta sulla casella *Enable NT/2000 Driver*, quindi premere *OK* e riavviare ICPROG come richiesto.

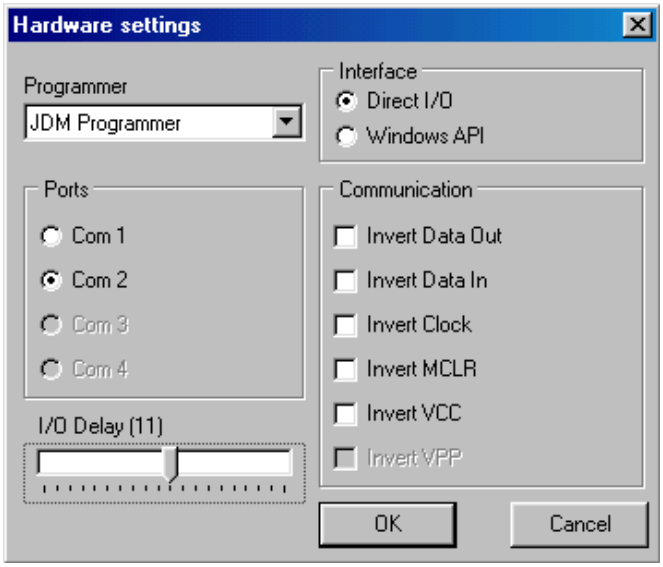
4. Al riavvio, rispondere affermativamente alla richiesta in installare il driver.


Apri
Esegui come...
TUGZip
Scansione icprog.exe per rice
Invia a
Taglia
Copia
Crea collegamento
Elimina
Rinomina
Proprietà
Proprietà - icprog.exe
Generale
Versione
Compatibilità
Se il programma funzionava correttamente con una versione precedente di Windows, mentre ora si incontrano dei problemi, selezionare la modalità di compatibilità che corrisponde a quella della versione precedente.
Modalità compatibilità
Esegui il programma in modalità compatibilità per:
Windows 2000
Impostazioni schermo
Esegui in modalità 256 colori
Esegui in risoluzione 640 x 480
Disattiva i temi visivi
Impostazioni input
Disattiva servizi di testo avanzati per questa applicazione
Ulteriori informazioni sulla compatibilità dei programmi.
OK
Annulla
Applica
IC-Prog 1.04 - Prototype Programmer
Options
Confirmation
Notification
I²C
Programming
Shortcuts
Drag & Drop
Smartcard
Language
Shell
Misc
Process Priority
Normal
High
Realtime
Enable NT/2000 Driver
Enable Vcc control for JDM
Make all outputs high (Combi)
OK
Cancel
Confirm
Before this driver will work, you must restart IC-Prog.
Do you want to restart IC-Prog now?
Yes
No
Confirm
IC-Prog driver is not installed. Do you want to install this driver now?
Yes
No

ICPROG:configurazione iniziale



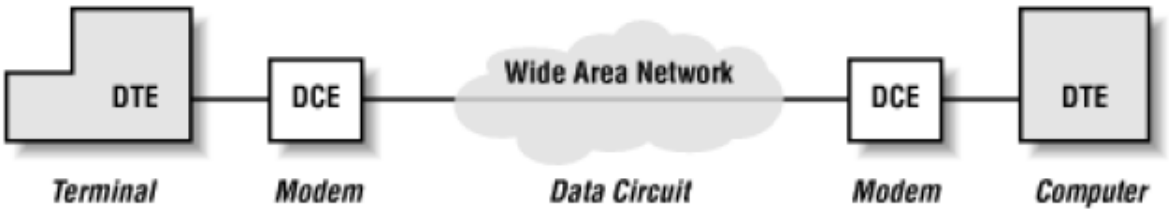
A questo punto ICPROG è stato installato correttamente. Per renderlo compatibile con il programmatore da noi realizzato, però, occorre che le impostazioni del tipo di programmatore siano esatte. Dalla finestra principale di ICPROG, selezionare il pulsante con il simbolo della "spina" (come mostrato in figura). Apparirà una finestra. Impostarne i valori in modo tale che appaia identica alla figura qui sotto. Si noti solo la porta COM da utilizzare, che varia a seconda di dove è connesso il circuito. Anche il numero di porte disponibile potrebbe variare.



L'interfaccia Seriale RS 232

Cos'é e a cosa serve l'RS232

Lo standard RS232 definisce una serie di specifiche per la trasmissione seriale di dati tra due dispositivi denominati **DTE** (Data Terminal Equipment) e **DCE** (Data Communication Equipment). Come si può vagamente intuire dal nome, il Data Communication Equipment è un dispositivo che si occupa di gestire una comunicazione dati mentre il Data Terminal Equipment è un dispositivo che si occupa di generare o ricevere dati. In pratica l'RS232 è stata creata per connettere tra loro un terminale dati (nel nostro caso un computer) con un modem per la trasmissione a distanza dei dati generati. Per avere una connessione tra due computer è quindi necessario disporre di quattro dispositivi: un computer (DTE) collegato al suo modem (DCE) ed un altro modem (DCE) collegato al suo computer (DTE). In questo modo qualsiasi dato generato dal primo computer e trasmesso tramite RS232 al relativo modem verrà trasmesso da questo al modem remoto che a sua volta provvederà ad inviarlo al suo computer tramite RS232. Lo stesso vale per il percorso a ritroso.



Per usare la RS232 per collegare tra loro due computer vicini senza interporre tra loro alcun modem dobbiamo simulare in qualche modo le connessioni intermedie realizzando un cavo **NULL MODEM** o cavo invertente, ovvero un cavo in grado di far scambiare direttamente tra loro i segnali dai due DTE come se tra loro ci fossero effettivamente i DCE.

Per connettere il PC al nostro circuito simuleremo invece direttamente un DCE facendo credere al PC di essere collegato ad un modem. Prima di fare questo diamo uno sguardo in dettaglio al principio di funzionamento di una comunicazione seriale.

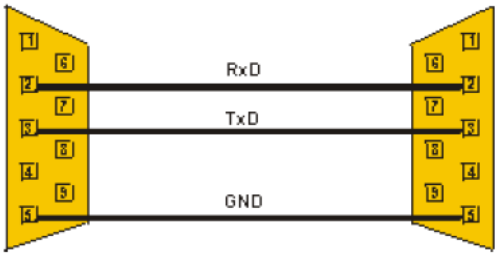
La comunicazione seriale asincrona

Per consentire la trasmissione di dati tra il PC ed il modem, lo standard RS232 definisce una serie di specifiche elettriche e meccaniche. Una di queste riguarda il tipo di comunicazione seriale che si vuole implementare che può essere sincrona o asincrona.

Nel nostro caso analizzeremo solo la comunicazione seriale asincrona ignorando completamente quella sincrona in quanto più complessa e non disponibile sui normali PC.

Una comunicazione seriale consiste in genere nella trasmissione e ricezione di dati da un punto ad un altro usando una sola linea elettrica. In pratica se desideriamo trasmettere un intero byte dobbiamo prendere ogni singolo bit in esso contenuto ed inviarlo in sequenza sulla stessa linea elettrica, un pò come avviene per la trasmissione in codice morse. La differenza sostanziale sta nel fatto che a generare e ricevere dati non c'é il telegrafista ma un computer per cui le velocità di trasmissione raggiungibili sono molto superiori.

Porta Seriale: pin utilizzati

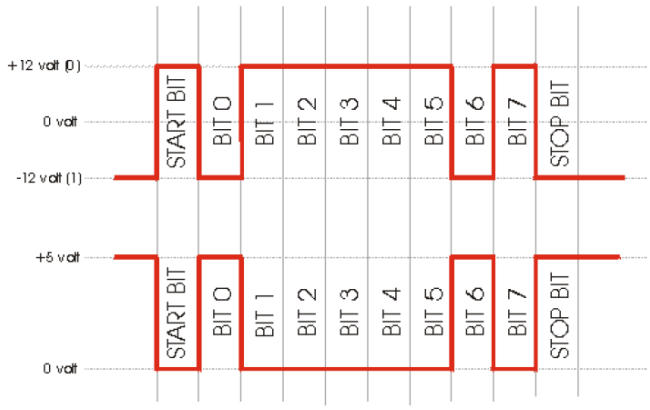


Nel nostro specifico caso, fra le tante possibili configurazione della porta seriale, abbiamo scelto la più semplice, ossia un collegamento diretto senza controllo di flusso. Pertanto, i pin coinvolti erano soltanto tre: TXD,RXD, GND.

La linea **Transmit Data** (TXD) presente sul pin 3 del connettore DB9 maschio è connessa alla rispettiva gemella, così pure come **Receive Data** (RXD) presente sul pin 2. Le masse (GND) presenti sul pin 5 di entrambe gli utilizzatori sono connesse tra loro. Si ha quindi una configurazione non incrociata.

Adattamento di Livello TTL – RS232

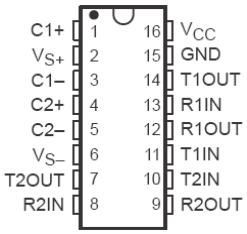
L'impiego dello standard seriale, però, non è esente da problemi. Uno dei più ostici è l'adattamento di livello con i dispositivi TTL. RS232, infatti, prevede un range di tensioni ± 12 V, contro gli 0 – 5 V TTL. Ciò significa che collegando direttamente il PIC alla porta seriale e quindi a PC, questi non sarà in grado di riconoscere i bit in arrivo, compromettendo così la trasmissione. Fortunatamente, esiste un soluzione semplice ed efficace al caso: l'integrato **MAX232** prodotto dalla **Maxim**.



Questo dispositivo è dotato di due ingressi e due uscite per ciascuna porta che è in grado di pilotare (nella figura qui sotto sono due). Ingressi ed uscite sono accoppiati, nel senso che ad un ingresso TTL è associata un'uscita RS232, e simmetricamente ad un ingresso RS232 è associata un'uscita TTL. Funzionalmente, il comportamento può essere così riassunto: supponiamo di aver in ingresso al MAX232 un segnale TTL. Questi inizia con un "1" logico, che viene trasformato in uno

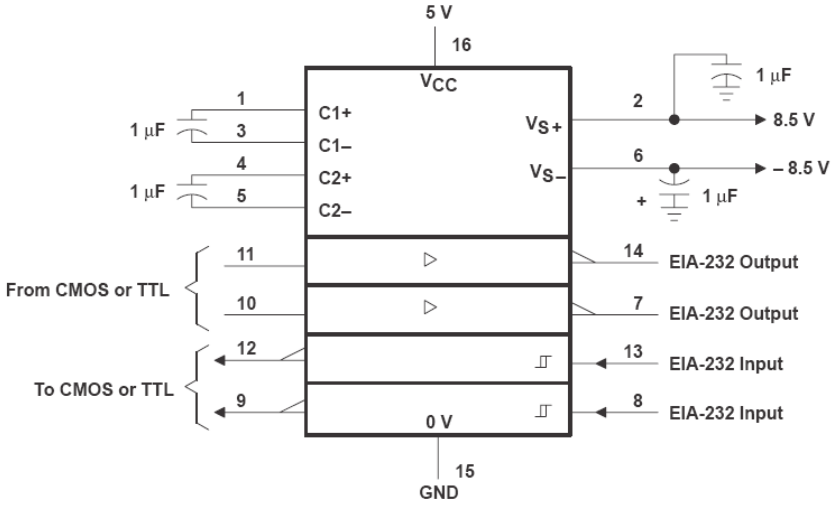
zero seriale, con tensione – 12 V. A questo punto si alternano gli altri bit, dal sincronismo ai dati, la cui tensione è "invertita" rispetto al valore TTL. Ad uno zero TTL corrispondono + 12 V RS232 e viceversa. Lo schema a lato può fornirne un'idea.

PinOut



Come già accennato, MAX232 è fornito di coppie di due ingressi – uscite TTL – RS232, tante quante le porte che è in grado di gestire. Il modello qui riportato ne pilota due, quindi i pin utili alla trasmissione sono 8. Escludendo l'alimentazione e la massa, avanzano altre 6 pin, il cui scopo è permettere il collegamento esterno di alcuni condensatori necessari al funzionamento e non inclusi nel package (si veda qui sotto per ulteriori spiegazioni). Esistono circa 15 modelli di MAX232 ognuno con caratteristiche diverse: per esempio alcuni modelli (più costosi) includono internamente i 4 condensatori da 1µf. La versione da noi usata è la N, che necessita di 4 condensatori da 1µf.

Schema Tipico di Utilizzo



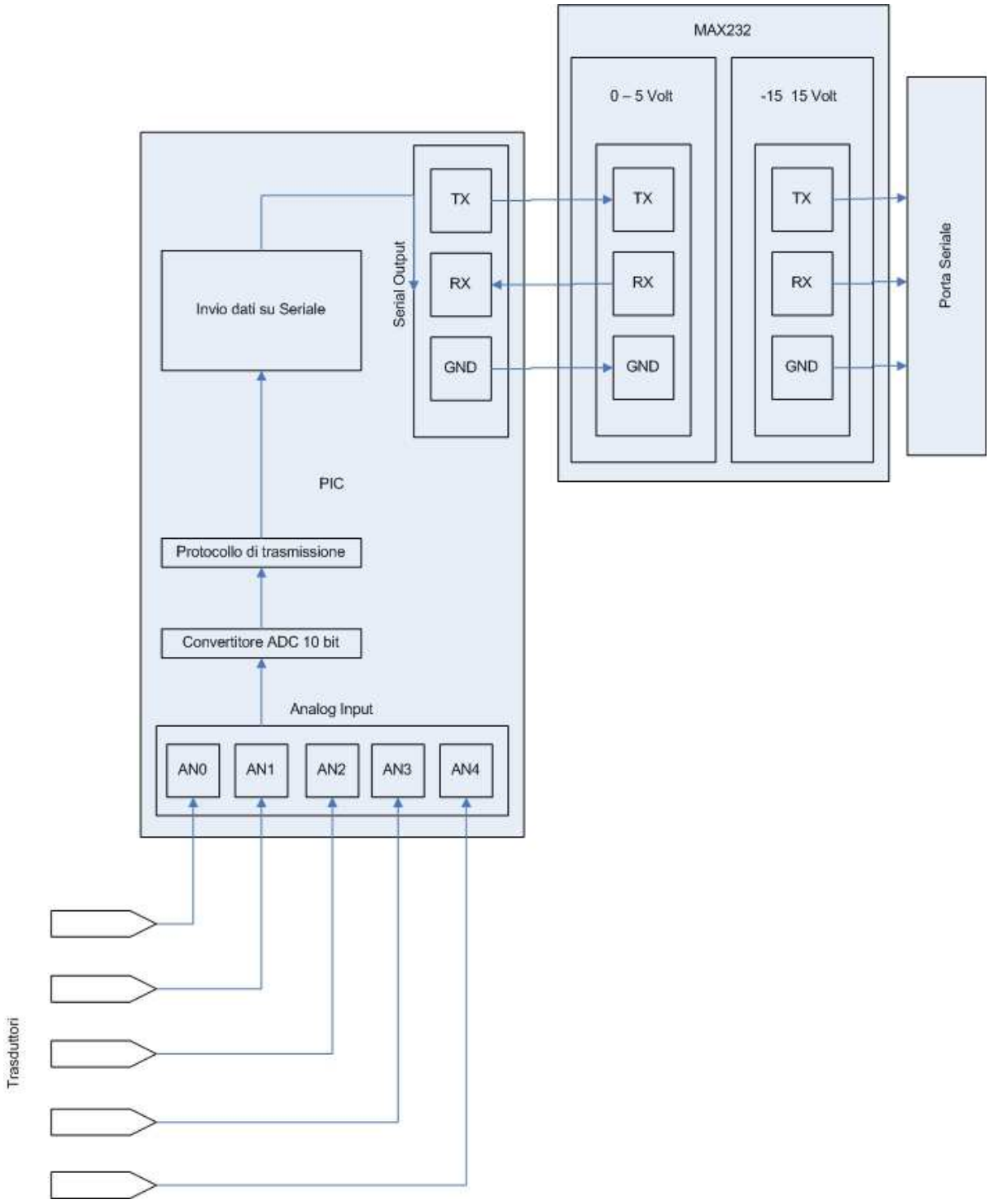
Questo semplice schema descrive la configurazione tipica del MAX232. È possibile notare i diversi ingressi e le diverse uscite, che agiscono nel processo di adattamento, ma in particolare altre tre coppie di pin, tra loro connesse tramite un condensatore da 1 µF. In questo modello, i condensatori sono esterni e a carico dell'utente, mentre in altre versioni sono addirittura inclusi nel package del dispositivo.

Scheda d'Acquisizione

Veniamo ora alla scheda di acquisizione vera e propria. In commercio esistono già soluzioni precostruite, in molti casi affiancate da un applicativo utile alla loro gestione. Nel caso, però, abbiamo voluto partire da zero, sia perché le soluzioni commerciale avevano un costo elevato, sia perché è sempre stata nostra intenzione non appoggiarci a progetti realizzati da terzi. Il progetto è

stata anche occasione di misurare le nostre abilità e di raffrontarci con le difficoltà tipiche della progettazione di sistemi partendo da zero.

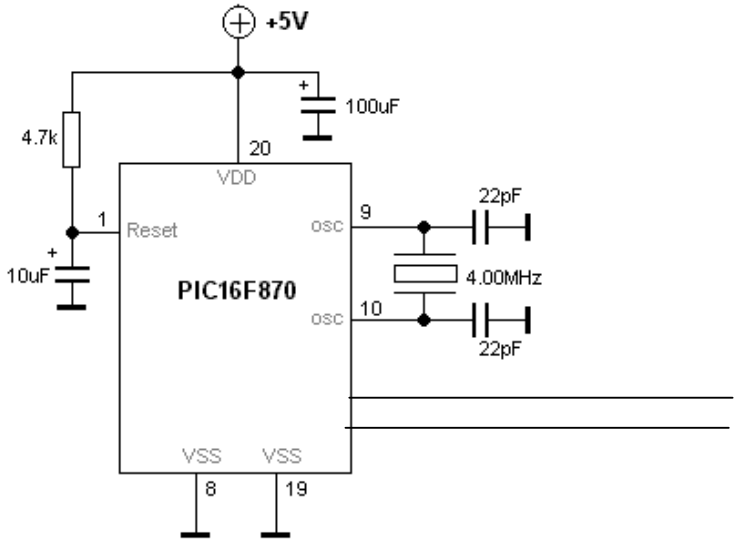
Schema Logico



A livello funzionale, il comportamento della scheda di acquisizione potrebbe essere così riassunto: abbiamo in ingresso alle cinque linee analogiche altrettanti trasduttori. Le tensioni generate da questi dispositivi vengono acquisite dal PIC, e quindi passate all'ADC (*Analog-To-*

Digital Converter, convertitore analogico – digitale), che ne fornisce una rappresentazione numerica a 10 bit, cioè su di una scala di 1024 (2^{10}) valori. A questo punto, il microcontrollore compone il messaggio da trasmettere su seriale e lo pone in uscita sui propri pin dedicati alla trasmissione. Questi valori di tensione non sono però conformi a quelli richiesti dallo standard RS232, e perciò è necessario un integrato di adattamento. A questo punto, le informazioni sono state correttamente inviate alla porta seriale del PC.

Schema Circuitale



Protocollo di comunicazione

La necessità di inviare al PC i valori digitalizzati dei 5 input analogici su seriale ha comportato l'esigenza di creare un semplice protocollo di comunicazione che si prestasse all'uso. Le scelte possibili erano in realtà due: o inviare singolarmente ogni coppia porta – valore sequenzialmente, o tutte le coppie contemporaneamente. È prevalsa quest'ultima ipotesi ed abbiamo così predisposto il seguente schema:

porta1:valore1; porta2:valore2; porta3:valore3; porta4:valore4; porta5:valore5

Dove *portaX* è un valore 0,1,2,3,4 (1-5 partendo da 0), ed identifica la porta dai cui il dato proviene. *valoreX* è invece compreso tra 0 e 1023 (10 bit, 2^{10} valori) e rappresenta la codifica in binario del segnale acquisito.

Esempio:

0:1024;1:512; 2:1024;3:0;4:1024

Significa:

Porta 0:	5	Volt
Porta 1:	2.5	Volt
Porta 2:	5	Volt
Porta 3:	0	Volt
Porta 4:	5	Volt

Il linguaggio PICBasic

Di norma l'unico modo per programmare un microcontrollore è scriverne il codice in linguaggio assembly, o per meglio dire un linguaggio a basso livello, con istruzioni molto simili, e poi usare un compilatore per generare il file esadecimale da inviare al dispositivo. La programmazione a basso livello è però molto complessa, poiché richiede di curare ogni fase ed ogni singolo particolare del ciclo di vita di un "applicativo", sebbene consenta ad uno sviluppatore molto esperto di migliorarne talvolta le prestazioni. Ciononostante, sia per limiti di tempo sia per chiarezza di apprendimento sono stati inventati alcuni linguaggi di livello superiori, in grado si sopperire alla scarsa immediatezza e praticità dell'assembly. Uno di questi è *PICBasic*, la cui sintassi si ispira largamente a *Basic*. Sebbene si caratterizzato da un dogma di programmazione piuttosto datato, la semplicità e la chiarezza del Basic hanno reso PICBasic uno dei linguaggi (se non IL linguaggio) più utilizzato nell'ambito della programmazione di microcontrollori. Oltre a questo, è necessario riportare come a tutti gli effetti questo linguaggio abbia di fatto raggiunto un ottimo livello di standardizzazione, consentendo di riutilizzare il medesimo listato di codice anche su PIC diversi. La stessa cosa non era concepibile in assembly, dove ogni PIC richiede un diverso codice.

DEVICE

Imposta il dispositivo target, ossia il modello di microcontrollore utilizzato. È fondamentale perché la compilazione in codice assembly è differente a seconda del modello ed in linea di massima eseguibili con DEVICE diversi non sono compatibili tra di loro.

DECLARE

Utilizzata per dichiarare variabili e costanti e per impostare parametri interni al PIC. In modo particolare, è indispensabile per configurare opzioni come la precisione dei convertitori o il tipo di oscillatore.

SYMBOL

È usato per creare pseudonimi, non può essere usato per creare variabili

ADIN

Istruzione essenziale per l'acquisizione. Preleva il segnale dal canale specificato, lo converte in digitale con la precisione impostata (vedere le opzioni), e salva il valore ottenuto nella variabile passata come parametro.

```
Var = ADIn Channel
```

SEROUT

Questa istruzione rappresenta il fulcro del sistema, in quanto gestisce l'invio di dati su seriale. Possono essere specificate numerose porte, in pratica tutti i pin di PORTA, PORTB e PORTC (più altri eventuali) che supportano l'IO.

```
SEROUT Pin,Mode,[Item{,Item...}]
```

Dove *Pin* indica il pin da utilizzare come collegamento, *Mode* la velocità di trasmissione su seriale e la rispettiva codifica di linea (vedi tabella qui sotto) e *Item* dati da inviare.

Mode	*Mode* No.	Baud Rate	State
T2400	0	2400	Driven True
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	Driven Inverted
N1200	5	1200	
N9600	6	9600	
N300	7	300	
OT2400	8	2400	Open True*
OT1200	9	1200	
OT9600	10	9600	
OT300	11	300	
ON2400	12	2400	Open Inverted*
ON1200	13	1200	
ON9600	14	9600	
ON300	15	300	

DelayMS

Questa istruzione è invece utile per introdurre un ritardo prefissato nella sequenza di programma. Si tratta di una funzione molto comoda per creare un firmware in grado di effettuare un *polling* delle porte e degli ingressi. Ne esiste anche una versione con altre unità di misura, diverse dai millisecondi.

DELAYMS *ms*

Dove *ms* rappresenta il numero di millisecondi da attendere prima di proseguire con il flusso di programma.

GoTo

Funziona in maniera identica al Basic tradizionale, e cioè effettua un salto incondizionato. Insieme a DelayMS è fondamentale per creare firmware che effettuino operazioni cicliche.

GoTo *label*

Dove *label* è un etichetta di codice definita in un altro punto del listato.

End

Ultima istruzione, che deve sempre comparire alla fine di ogni listato. Termina il flusso di programma.

Applicazione Windows

Descrizione Generale

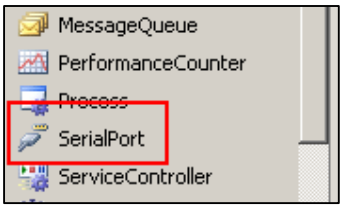
Veniamo ora all'ultima sezione del progetto: la parte software su computer. Abbiamo già parlato in precedenza di quali fossero le richieste, ma le riportiamo comunque anche qui:

- **Acquisizione** dei dati in arrivo, su seriale, dalla scheda di acquisizione
- **Elaborazione** delle informazioni in modo grafico
- **Controllo** in tempo reale dei dati, permettendo di impostare un range di valori accettabili, fuori dal quale si considera la situazione come fuori norma.

- **Allerta** in caso di pericolo, diffondendo in rete locale un messaggio di avviso che determinati *client* possano intercettare e mostrare all'utente.

Vista la complessità dei requisiti qui proposti, abbiamo scelto di implementare la soluzione in un linguaggio moderno, potente ad affidabile quale *Microsoft Visual C# 2005*. Si tratta in sostanza della versione 2.0 del *.NET Framework*, rilasciata soltanto a novembre 2005, che è stata esplorata e visionata proprio in virtù di questo progetto.

Classe SerialPort



Fra le oltre 5000 nuove funzionalità, C# 2005 presenta anche un nuovo namespace, *System.IO.Ports*, che include nuove classi adatta all'uso di interfacce e porte di sistema. Attualmente, però, l'unica disponibile è *SerialPort*, utile per la gestione di una porta seriale RS232.

Questa classe si è rivelata molto versatile e comoda per la gestione dell'input dal PIC, e ha permesso di velocizzare la realizzazione della parte di ricezione dei dati, permettendoci quindi di dedicare maggiore attenzione alle altre funzionalità.

Tra i membri da noi utilizzati, si evidenziano:

- L'evento **DataReceived**, scatenato quando la porta riceve dati validi.
- Il metodo **ReadLine**, che permette di leggere il buffer d'ingresso della porta.
- Il metodo **Open** e **Close**, per aprire e chiudere la porta stessa.
- Le proprietà **BaudRate** e **PortName** per impostare i parametri di trasmissione.

UserControl: ADCChart

Dovendo gestire l'acquisizione di *n* segnali analogici sarebbe stato necessario replicare altrettante volte l'interfaccia utente richiesta alla visualizzazione dei dati. In casi come questo è preferibile, invece, sviluppare uno *UserControl*. Cosi facendo si implementa un componente che fornisca le funzionalità richieste una singola volta, e quindi lo si rende *riutilizzabile*. Per questi motivi, nello sviluppo dello *UserControl* è stato fondamentale tenere in speciale considerazione la *flessibilità* e la *scalabilità*, per poter adattare il controllo utente agli utilizzi più svariati. Abbiamo così cercato di concentrare il maggior numero di funzionalità possibili all'interno dello *UserControl*.

Le funzionalità offerte dall'ADCChart sono:

- Visualizzazione istantanea dei segnali acquisiti.
- Possibilità di modificare in tempo reale i limiti di controllo.
- Evento software scatenato quando il valore acquisito esce dal controllo.
- Impostazione personalizzabile dei tipi di segnali in ingresso per qualsiasi trasduttore.
- Cattura dei valori acquisiti.

Server di monitoring

Avendo ora a disposizione tutti i dati con il relativo sistema di controllo, abbiamo pensato che sarebbe risultato più utile se i dati inerenti gli ambienti controllati fossero stati disponibili anche all'interno di una rete, in modo tale che in caso di situazione anomala la postazione su cui è installata la scheda di acquisizione possa comunicare ad altri host, e quindi a persone, la segnalazione di pericolo, diramando di fatti un allarme.

In quest'ottica si è reso necessario implementare un'apposita funzionalità nel sistema di controllo che inviasse il messaggio ad un numero imprecisato di host. La differenza rispetto ai tradizionali sistemi client – server, infatti, consiste proprio nel fatto che **è il server** a contattare i client di propria iniziativa, e non il contrario. Il problema, quindi, era far sì che un certo numero di client non conosciuti a priori, venissero notificati di un cambiamento in un ambiente dal server. In tal senso, allora, si è reso necessario utilizzare il protocollo **UDP** (User Datagram Protocol), che diversamente dal più conosciuto e diffuso TCP consente l'invio di pacchetti multicast e broadcast (a più destinatari).

Client di notifica

Una volta predisposto il servizio di invio allerta, abbiamo realizzato un semplice applicativo, sempre utilizzando le Windows Forms, che mostri all'utente un messaggio di avviso quando il server dirama un allarme. Anche qui è stato necessario ricorre al protocollo UDP, dei namespaces *System.Net* e *System.Net.Sockets*, oltre che ai *thread*. L'implementazione di UDP infatti non prevede una gestione a richiesta, con *interrupt*, dell'arrivo di messaggi. In tal senso è stato allora indispensabile creare un thread separato che sondasse periodicamente il client UDP alla ricerca di notifiche in ingresso. La scelta dei thread è stata forzata, perché non era ammissibile impostare un timer di polling, per due motivi: primo perché il thread garantisce un controllo continuo del buffer d'ingresso; secondo, il thread è *non bloccante*, e ciò significa che l'applicativo non rischia di andare in crash.

Dimensione dei componenti software

Componente	Dimensione (righe di codice)
ADCChart	500
Server	170
Client	75
TOTALE	745

Realizzazione

Fasi di Esecuzione

Ricerca

La prima fase di progetto è stata incentrata sulla ricerca dei componenti necessari la realizzazione dell'interfacciamento ed il risultato ha evidenziato il *microcontrollore programmabile (PIC)* quale componente essenziale. **Non essendo però in possesso né alcun manuale né di alcuno schema**, è stato necessario reperirli altrove.

Grazie ad un massiccia raccolta dati, svolta in parte a scuola, in parte a casa, è stato ottenuto un quantitativo sufficiente di documentazione per iniziare il progetto. In particolare, abbiamo reperito un elenco dei più comuni PIC in commercio con i relativi datasheet e alcune applicazioni tipiche, ma soprattutto alcuni dettagliati circuiti di programmazione e il relativo software. Non bisogna dimenticare, infatti, che **ogni dispositivo programmabile necessita di un apposito circuito di programmazione**, necessario a trasferirvi il codice sorgente per il corretto funzionamento. Dal momento che **non avevamo ad disposizione alcun programmatore**, è stato indispensabile realizzarne uno.

Creazione del Programmatore

Abbiamo così dedicato la nostra attenzione alla realizzazione del circuito di programmazione necessario per abilitare le funzionalità di acquisizione e trasmissione seriale del PIC. Fra i vari schemi da noi reperiti, è stato scelto il più semplice ed adeguato alle nostre necessità.

Programmazione del PIC

Una volta realizzato il programmatore ed ottenuti gli integrati abbiamo immediatamente verificato il corretto funzionamento del circuito. Per farlo è stato utilizzato l'applicativo *ICProg*, dedicato alla programmazione di chip ed in grado di pilotare innumerevoli tipi di integrati. Servendoci delle specifiche recuperate durante la fase di ricerca, abbiamo opportunamente configurato ICProg per il PIC16F870 e quindi collegato il programmatore alla porta seriale del computer.

Abbiamo quindi scritto un semplice programma di esempio, il tradizionale *"Hello, World!"*, con l'intento di inviare un messaggio fisso al computer e verificare così la corretta configurazione del programmatore. Aabbiamo compilato l'eseguibile in un formato esadecimale compatibile con il PIC scelto ed attivato il programmatore. La verifica del codice ha dato esito positivo.

A questo punto, abbiamo iniziato a visionare il secondo software utile alla programmazione, *ProtonDS*: si tratta di un IDE veramente completo esclusivamente dedicato alla programmazione di PIC. Tra le svariate funzionalità, si evidenzia soprattutto il supporto ad un linguaggio specifico per la programmazione di PIC, *PIC Basic*, largamente ispirato al tradizionale Basic, ma comprensivo di istruzioni per la gestione dell'IO e dei registri interni al PIC. Il tutto ad un livello di astrazione superiore al codice assembly.

Alla fine di questa fase, il PIC era in grado di inviare correttamente i valori acquisiti tramite i propri ingressi, convertirli in digitale ed inviarli su seriale.

Sviluppo dell'applicativo software

L'ultima fase ha visto la realizzazione dell'applicazione utilizzabile come monitor per l'interfaccia hardware appena realizzata. Le funzionalità richieste erano:

1. Acquisire i dati in arrivo su seriale, nel formato utilizzato dal PIC.
2. Visualizzarne l'andamento istantaneo in modo grafico.
3. Predisporre un sistema statistico di controllo dei valori in ingresso, impostando cioè un massimo ed minimo valore tollerato.
4. Lanciare un allarme in caso di valori fuori controllo, sia in ambito locale con una schermata di avviso, sia trasmettendo un pacchetto di allerta in rete, supposto il computer connesso ad una rete locale.

Per sviluppare un applicativo così completo in modo efficace ed efficiente, si è scelto di utilizzare *Microsoft Visual Studio 2005* e il linguaggio *C# versione 2.0*, rilasciato nel Novembre 2005.

Materiale Utilizzato

Programmatore

Materiale	Qtà
Disaccoppiatori Ottici PC817	2
Trimmer 300Ω	3
Porta seriale DB9	1

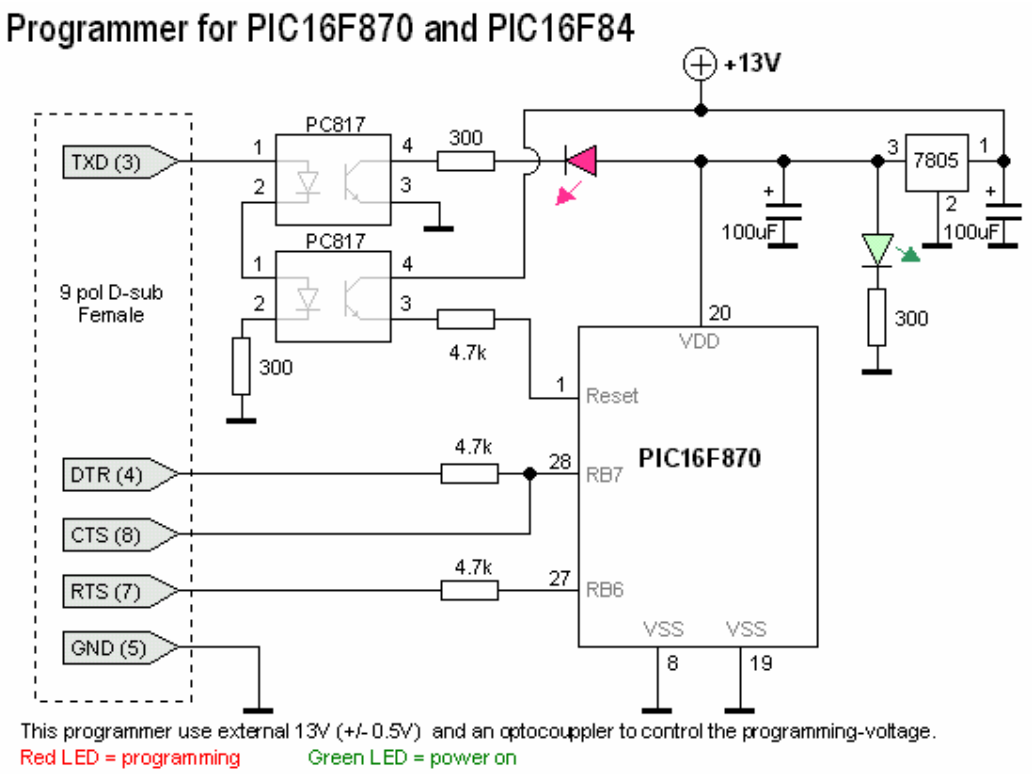
Materiale	Qtà
Led rosso	1
Led verde	1
Condensatore 100µF	2
Stabilizzatore L7805	1
Resistenza 4,7 kΩ	3

Scheda di Acquisizione

Materiale	Qtà
PIC 16F870	1
Condensatore 22pf	2
Condensatore 100µf	1
Condensatore 10µf	1
Resistenza 4,7kΩ	1
Porta seriale DB9 Maschio 90°	1
Porta USB Tipo N Femmina 90°	1
Quarzo 4Mhz	1
MAX232CP	1
Condensatore 1µf	4
Cavo Seriale	1
Cavo USB	1

Programmatore

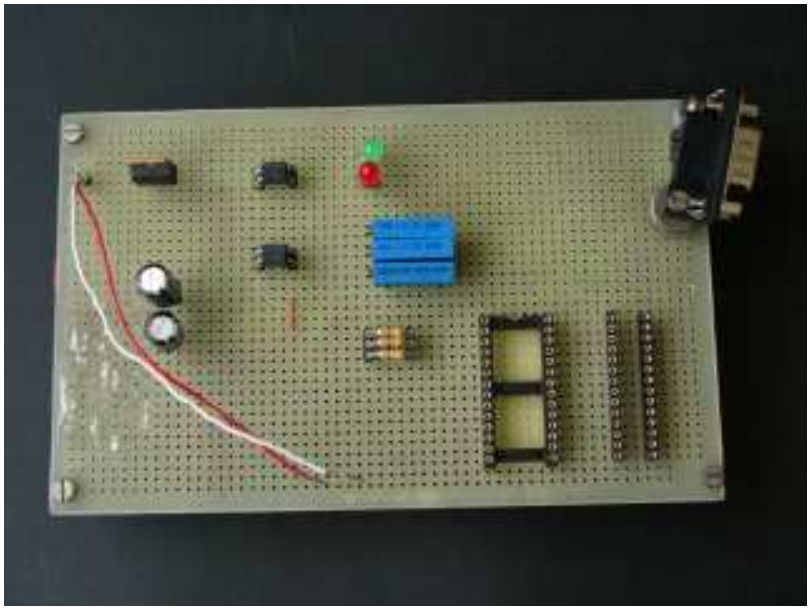
Come abbiamo già accennato poc'anzi, il primo circuito che abbiamo realizzato è stato il programmatore. Avendo a lungo cercato, siamo infine riusciti a trovarne uno facilmente realizzabile, caratterizzato da pochi componenti. Riportiamo ancora una volta lo schema:



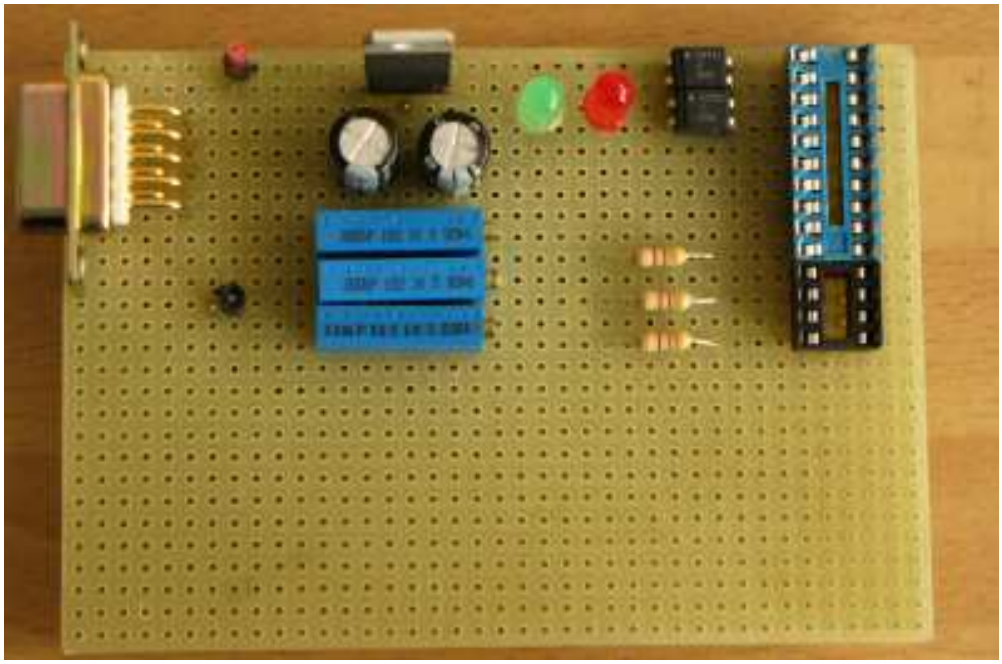
(Per la spiegazione si veda la parte teorica)

Le difficoltà sono state comunque non poche. Innanzitutto è stato ostico reperire tutti i componenti necessari, dai *disaccoppiatori ottici (PC817)* allo *stabilizzatore di tensione (7805)* e al connettore della *porta seriale (DB-9)*. In quest'ultimo caso, abbiamo provato ad adattare un cavo esistente, rimuovendo i collegamenti già presenti ed effettuando una operazione di saldatura per i nuovi. Il risultato era però alquanto insoddisfacente. Avendo però a disposizione tempo, visto il mancato arrivo dei componenti programmabili, abbiamo deciso di riprogettare il dispositivo utilizzando la tecnica della *wrappatura*. Essendo novizi alla tecnica, la prime fasi hanno visto qualche difficoltà tecnica, ma in seguito le operazioni si sono velocizzate e presto il nuovo programmatore è stato terminato. Per quanto riguarda la porta seriale, poi, abbiamo acquistato un connettore vero e proprio, non un fascio di fili, e lo abbiamo fissato con alcune visti alla piastra, assicurandoci così che non vi fosse pericolo di rottura. Non bisogna dimenticare, infatti, che la porta è il componente soggetto alle sollecitazioni più forti, sottoposta ad un continuo connettere e disconnettere del cavo di connesione da parte dell'utente.

Alla fine è venuto il momento di provare il funzionamento del circuito. Il primo tentativo ha dato esito negativo, in quanto erano stati erroneamente collegati due pin in modo inverso (colpa del capovolgere la piastra durante la *wrappatura*). Prova ne è che dopo le opportune correzioni è stato possibile programmare il PIC con un eseguibile di prova.



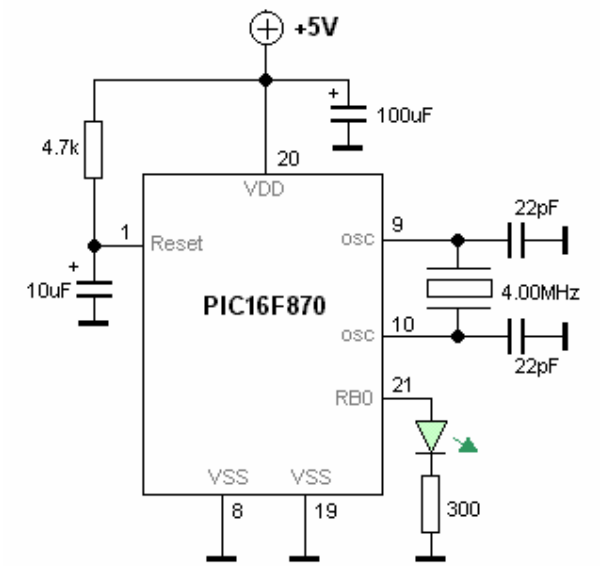
La foto qui sopra mostra come appariva il programmatore realizzato con wrappatura. In effetti, nelle ultime fasi di progetto, abbiamo deciso di ricostruire un'ultima volta il programmatore tramite *saldatura*. L'idea ci è venuta considerando la labilità dei collegamenti *wrappati* e rilevando anche la pessima organizzazione dei componenti sulla millefori. Ci siamo quindi procurati il materiale necessario e nell'arco di una sola giornata il programmatore era già operativo, in un nuovo package decisamente più curato ed efficiente.



Scheda di acquisizione

Passiamo ora alla scheda di acquisizione vera e propria. Inizialmente non abbiamo tentato di realizzare lo schema completo, ma ci siamo limitati a verificare la configurazione e il funzionamento del PIC. Precisiamo ancora che le nostre conoscenze in fatto di dispositivi programmabili erano pressoché basilari e che non avevamo mai realizzato alcun circuito in logica programmabile. Per questi motivi, il primo schema implementato è stato un semplice *flasher*, ossia un dispositivo che accenda un LED ad intervalli di tempo prefissati.

Step 1 – Flasher



Lo schema è in realtà molto semplice. La maggior parte dei componenti è necessaria al funzionamento del PIC, mentre soltanto il LED e la impedenza da 300 Ω sono distaccati e si accenderanno al comando del microcontrollore ("1" logico sul pin 21, PORTB.0).

La prova ha avuto immediatamente esisto positivo e ciò ci ha incoraggiati a tentare direttamente con la trasmissione seriale.

Step 2 – Trasmissione Seriale

Abbiamo allora programmato il PIC in modo che trasmettesse una sequenza di caratteri fissa, "*Hello, World*". A tal scopo abbiamo riconvertito lo schema sopra, coinvolgendo stavolta il pin 17, PORTC.0, ossia la linea TXD seriale. In questa occasione, invece, l'esito è stato negativo: nonostante ProtonDS integrasse anche un piccolo tool per il controllo della porta seriale, il PC non riceveva nulla. Dopo alcune riflessioni, abbiamo però individuato l'errore: non era presente alcun adattatore di livello. In effetti, la presenza dell'integrato MAX232 in tutti gli schemi da noi presi in considerazione ci ha indotto a pensare che la sua utilità fosse più che marginale ed infatti dopo averne scoperto l'impiego ne abbiamo acquistato un esemplare (l'unico rimasto nei negozi di tutta la città). E così, dopo l'aggiunta di questo stadio intermedio, anche la comunicazione seriale è riuscita.

Step 3 – Scheda di acquisizione

Dal passo precedente alla scheda di acquisizione, il passaggio è stato rapido. Ciò che mancava era l'impiego dei convertitori analogico – digitali e un protocollo di comunicazione adatto a trasmettere i dati registrati. Di quest'ultimo abbiamo già parlato e quindi ci concentreremo sulla multiplazione dei segnali e la loro acquisizione.

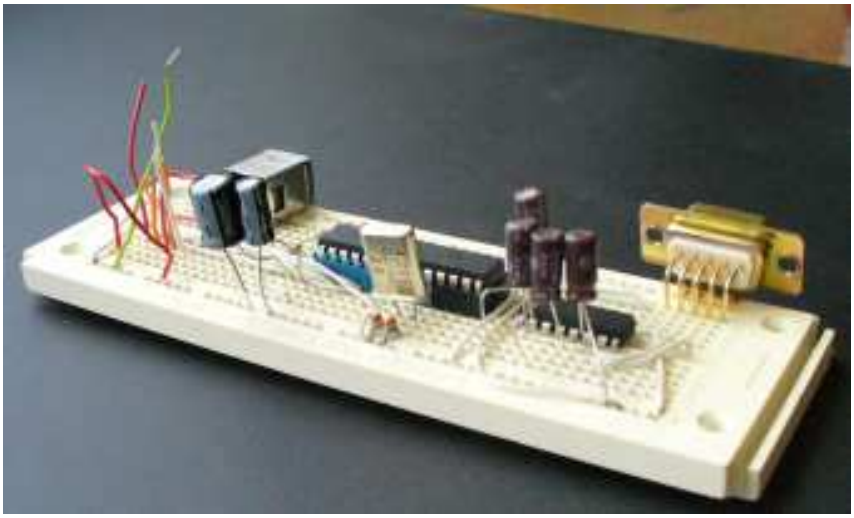
A livello materiale, entrambe le funzioni sono svolte internamente dal PIC16870, che dispone di 5 ingressi collegati ad un ADC con una precisione massima di 10 bit. È stato quindi

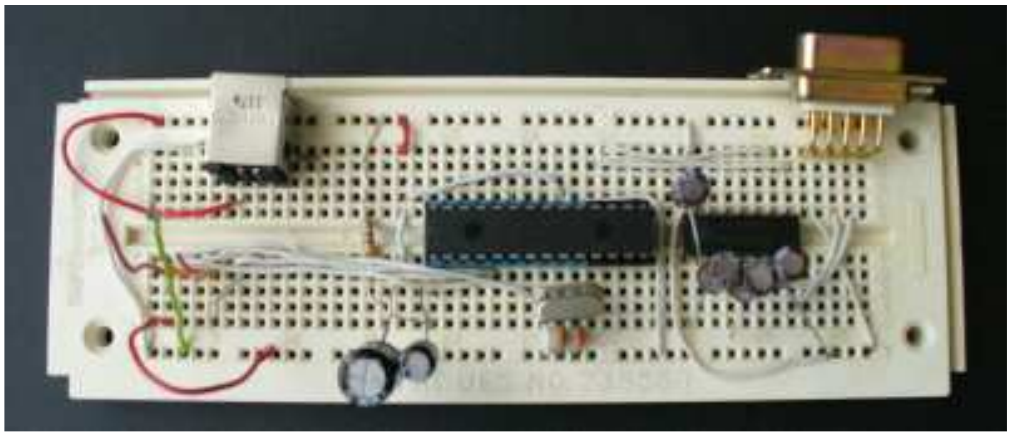
necessario individuare quali tra i 25 pin del dispositivo fossero abilitati e verificare la tensione accettata in ingresso. Riassumendo:

Nome	Pin	Tensioni
AN0	2	0 – 5 V
AN1	3	
AN2	4	
AN3	5	
AN4	7	

La difficoltà era invece insita nella comprensione delle istruzioni necessarie alla configurazione del microcontrollore e alla conversione dei dati. A questo proposito non avevamo molte informazioni, e abbiamo quindi effettuato una serie di prove. Alle fine, però, si è rivelato fondamentale lo studio del datasheet del PIC (in lingua inglese), nel quale erano contenute tutte le spiegazioni riguardo ai registri da utilizzare e alle istruzioni necessarie. Come sarà precisato tra poco, il problema consiste nell'indicare al microcontrollore quale sia la funzione svolta da uno specifico IO, ed eventualmente se considerare lo stesso come analogico o digitale.

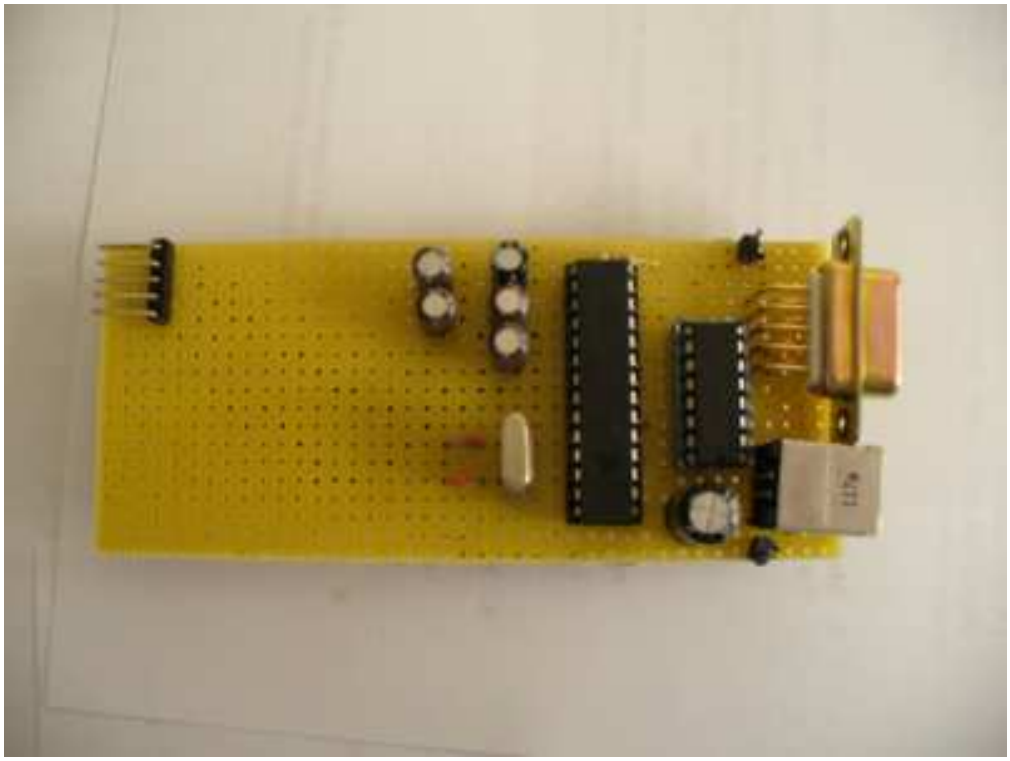
In ultimo, abbiamo poi deciso di programmare il PIC a prescindere da quale potesse essere la grandezza fisica misurata. In tal senso, il PIC acquisisce i dati, li converte in digitale e li invia senza effettuare alcun altra elaborazione. Sarà poi il software su PC ad essere informato del tipi di valore misurato. È questa una soluzione sufficientemente ovvia, tenendo conto della difficoltà di rendere scalabile un dispositivo come il microcontrollore rispetto al un personal computer.





Step 4 – Miglioramenti finali

Anche in questa circostanza, abbiamo preferito ricostruire il circuito su di una millefori tramite saldatura, per ottenere ancora una volta un dispositivo più ordinato ed efficiente in quanto a stabilità di collegamenti.



Infine abbiamo anche pensato ad una soluzione per rendere trasportabile il prodotto e renderlo indipendente dall'alimentatore standard. Nell'ottica di un utilizzo via PC quale è il nostro progetto, ci siamo accorti di poter impiegare la potenza elettrica resa disponibile dal computer ed in particolare dalla porta **USB**. Nella fattispecie, questa connessione è in grado, per sua natura, di alimentare periferiche fino a 5 V con 500 mA, più che sufficienti per il nostro caso. Il PIC consuma appena qualche mA, ma sfrutta appieno i 5 V a sua disposizione. Certo la stabilità della tensione fornita da USB non è paragonabile a quella di un buon alimentatore, ma è risultata comunque efficace, come del resto ampiamente dimostrato dalle prove. Tuttavia dobbiamo ammettere che i componenti necessari ad una simile operazione sono stati molto ardui da reperire. Nei negozi di elettronica e componentistica hardware di tutta la città non ve n'era traccia ed è stato possibile procurarsi una porta USB lato periferica solo rimuovendola da una vecchia ed inutilizzata stampante…

Firmware PIC

Listato

```
'*****************************************************************
'*  Name    : Acquire_Card.bas                                   *
'*  Author  : Chiodaroli Riccardo, Gatti Massimo                 *
'*  Notice  : Copyright (c) 2006                                 *
'*          : All Rights Reserved                                *
'*  Date    : 16/05/2006                                         *
'*  Version : 1.0                                                *
```

```
'*  Notes   :                                                      *
'*          :                                                      *
'*****************************************************************

    Device 16F870

    Declare ADIN_RES 10
    Declare ADIN_TAD 2_FOSC

    Dim temp0 As Word,temp1 As Word,temp2 As Word,temp3 As Word,temp4 As Word

    'Set RS232 transfer rate to 9600 baud
    Symbol N9600 = 104 - 20

    ' Configure AN0-AN1-AN2-AN3-AN4 as inputs
    TRISA = %00011111

    ' Set analogue input on PORTA.0
    ADCON1 = %10000000

loop:

    temp0 = ADIn 0
    temp1 = ADIn 1
    temp2 = ADIn 2
    temp3 = ADIn 3
    temp4 = ADIn 4

    SerOut PORTC.6,N9600,["0:",Dec temp0,";1:",Dec temp1,";2:",Dec
temp2,";3:",Dec temp3,";4:",Dec temp4,10,13]

    DelayMS 2000
    GoTo loop
    End
```

Spiegazione passo – passo

`Device 16F870`

Imposta il dispositivo target, il PIC16870 utilizzato nel progetto.

`Declare ADIN_RES 10`

Imposta la precisione dei convertitori Analogico – Digitali a 10 bit, 1024 livelli.

`Declare ADIN_TAD 2_FOSC`

Imposta il tipo di oscillatore. Può essere `2_FOSC`, `8_FOSC` o `32_FOSC` oppure `FRC`. Le prime quattro selezionano come clock un oscillatore esterno, mentre l'ultima indica al PIC di utilizzare il proprio oscillatore interno. Questa scelta è però sconsigliata in quasi tutti i casi. [rivedere!!!]

`Dim temp0 As Word,temp1 As Word,temp2 As Word,temp3 As Word,temp4 As Word`

Dichiara 5 variabili, che saranno utilizzate per contenere i dati acquisti dai convertitori. La dimensione Word è necessaria per supportare valori fino a 65535, quindi anche 1024. Il tipo *Byte*, invece, raggiunge invece solo 255.

`Symbol N9600 = 104 – 20`

Crea un *alias* sulla modalità di trasmissione seriale utilizzata. In questo caso abbiamo scelto 9600 perché [aggiungere].

```
TRISA = %00011111
```

Questa istruzione un poco criptica è necessaria per far sì che il PIC consideri gli input analogici AN0, AN1, AN2, AN3, AN4. Ricordando infatti che molti ingressi del PIC hanno funzioni multiple, occorre un sistema che permetta di segnalarne al PIC quella scelta. Per questi casi si utilizzano specifici registri interni, indicati nel datasheet. Si consideri la seguente tabella come mappa.

Analog Input	Pin
AN0	2
AN1	3
AN2	4
AN3	5
AN4	7

```
ADCON1 = %10000000
```

Altro registro. In questa situazione imposta il registro dell'ADC 1 affinché riconosca come analogico il proprio ingresso. La documentazione è ancora una volta contenuta nel datasheet del PIC.

```
loop:
```

Definisce un'etichetta di codice, che sarà poi utilizzata per creare un loop.

```
temp0 = ADIn 0, ...
```

Acquisisce i valori dagli ADC nelle rispettive variabili, pronte per l'invio.

```
SerOut PORTC.6,N9600,["0:",Dec temp0,";1:",Dec temp1,";2:",Dec temp2,";3:",Dec
temp3,";4:",Dec temp4,10,13]
```

Il "cuore" del sistema, l'invio dei dati su seriale. Il comportamento è già stato esplicato in precedenza, ma si notino ora le istruzioni **Dec**, obbligatorie per ottenere un valore decimale dei dati digitali acquisiti. In ultimo, si faccia caso alla combinazione 10,13 finale, che indicata un ritorno a capo – nuova riga.

```
DelayMS 2000
GoTo loop
```

Queste ultime istruzioni creano il loop, il ciclo continuo, che definisce il carattere di controllo dell'applicazione. Il delay a 2000 ms è necessario per garantire l'invio corretto dei dati e la loro elaborazione. D'altra parte, però, le temperature sono grandezze lentamente variabili e perciò il basso refresh non influisce negativamente sull'efficacia del dispositivo.

Applicazione Windows

Classe SerialPort

Nonostante l'apparente semplicità d'uso di questa classe, si sono verificati non pochi problemi nel suo utilizzo. Fatta eccezione per alcune incongruenze circa le impostazioni dei parametri di connessioni, le difficoltà maggiori hanno riguardato la lettura dei dati dal buffer d'ingresso. Dopo alcune analisi da noi effettuate, abbiamo individuato che la classe stessa funzionava su un thread separato e che andava quindi in conflitto con il thread del form quando si tentava di assegnare il valore contenuto nel buffer ad un controllo. Si trattava in effetti di una

chiamata di interoperabilità tra thread ed è stato necessario ricorre al metodo *Invoke*, utilizzando opportuni *delegate* ed alcune funzioni separate. Riportiamo un esempio per chiarire:

```
string s = Porta.ReadLine();
string[] values = s.Replace("\r","").Split(';');

this.Invoke(new NextValueDelegate(
      adc0.NextValue), float.Parse(values[0].Substring(2)) / 1024f);
```

La prima istruzione legge il dato dal buffer della porta e lo memorizza in una stringa. Questa viene poi letteralmente spezzata, in accordo con il protocollo di comunicazione da noi pensato. Ora, per inserire il nuovo valore nel ADCChart occorre chiamare il metodo *NextValue* dello stesso. Tuttavia, poiché il thread attivo è quello della porta seriale e non il form su cui è disegnato lo user control, è necessario utilizzare il metodo *Invoke* del form per effettuare una chiamata dal proprio thread all'altro. Il metodo in questione, però, richiede un *delegate* appropriato per il metodo da chiamare e perciò abbiamo dovuto definirne uno:

```
private delegate void NextValueDelegate(float val);
```

Il secondo argomento del metodo *Invoke*, poi, rappresenta i valori passati come parametri alla funzione puntata dal delegate, in questo caso il valore acquisito.

ADCChart

Gestione dei valori acquisiti

Uno dei problemi base da risolvere per rendere il nostro progetto scalabile e flessibile secondo ogni esigenza è quello di offrire la possibilità all'utente di specificare i parametri del segnale in ingresso. L'unico vincolo rimanente, necessario per una corretta acquisizione, è che il segnale analogico sia compreso negli 0 - 5 V TTL. I valori acquisiti sono così digitalizzati in un *range* di valori compresi tra 0 e 1024 (perché il convertitore è da 10 bit) e quindi per gestirli vengono divisi per 1024. In questo modo, infatti, otteniamo un valore a virgola mobile a precisione singola compreso tra 0 e 1, quale che sia il tipo di segnale acquisito.

Esempi
256 -> 0.25
512 -> 0.5
768 -> 0.75

L'utente deve però poter impostare l'unità di misura, il valore minimo e il valore massimo del segnale, al fine di fornire una rappresentazione grafica specifica e corretta. È stato pertanto necessario individuare una formula che permettesse di convertire i valori di tensione trasmessi dal PIC in informazioni conformi ai parametri fissati dall'utente. La formula trovata è la seguente:

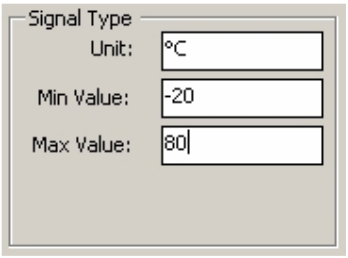
$$x = Min + k * \left(|Max| + |Min| \right)$$

Dove x è il valore incognito della grandezza specifica, *Min* e *Max* sono i valori di limite decisi dall'utente, mentre k è il valore in virgola mobile nell'intervallo 0 – 1 acquisito. In questo modo il funzionamento è indipendente dal tipo di segnale in ingresso.

Esempio:
Si deve acquisire un segnale 0 – 5 Volt di temperatura proveniente da un amplificatore operazionale con caratteristica lineare per cui:

- 0 Volt corrispondono a –20 °C
- 5 Volt corrispondono a +80 °C

La procedura consiste nell'aprire la scheda *Settings* del canale scelto ed impostare i campi del gruppo *Signal Type* nel modo seguente:



Grafica

Tenendo fede alla politica di realizzazione adottata finora, abbiamo deciso di non avvalerci di alcuno strumento già esistente anche per lo svilluppo necessario per generare il grafico con l'andamento dei valori. Abbiamo invece fatto ricorso a *GDI+*, ovvero la libreria grafica del *.NET Framework*, con il namespace *System.Drawing*.

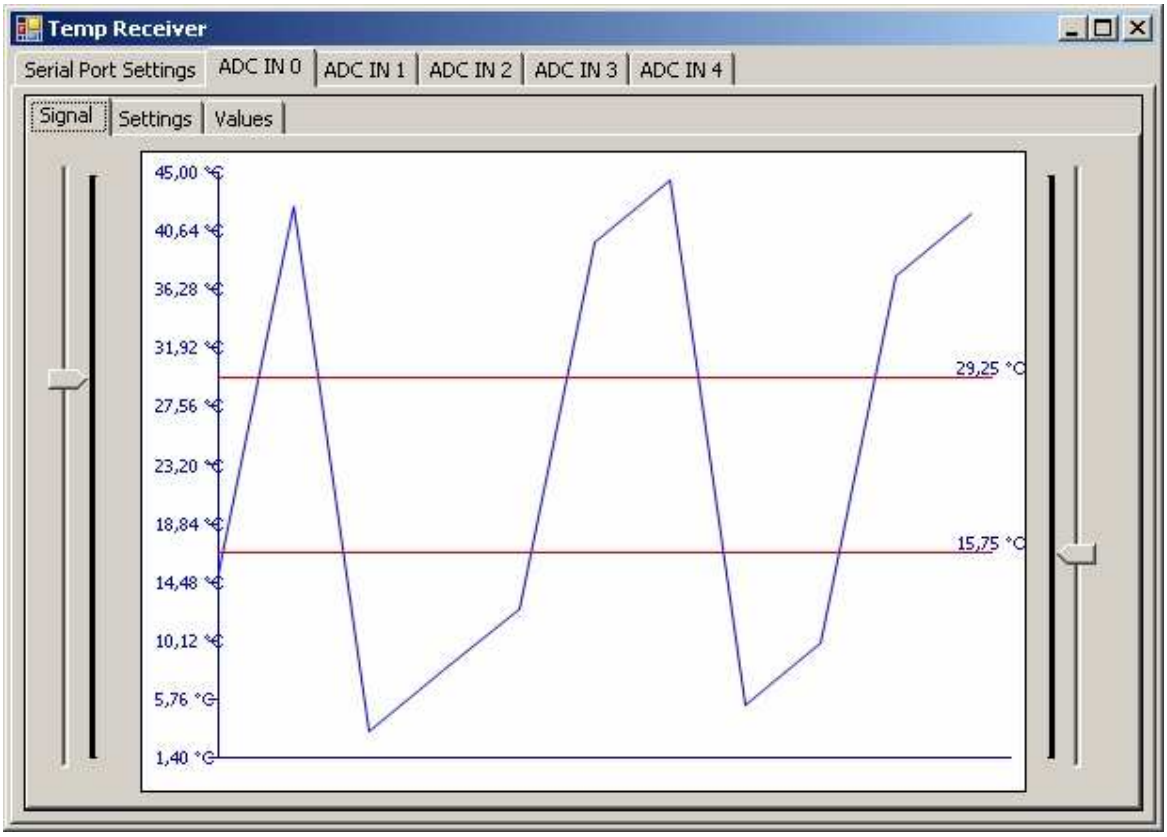
Le problematiche che sono stata affrontante sono state numerose. Solo per citarne alcuni:

- **Problema delle coordinate:** nei sistemi operativi Windows, il sistema di riferimento è diverso da quello adottato normalmente in geometria; l'asse delle *ordinate* infatti è invertito, e cosicché le ordinate negative diventano positive e viceversa.

- **Effetto scorrimento**: il grafico da mostrare, infatti, non poteva limitarsi ad un insieme di dati fisso, ma doveva consentire di accodare continuamente nuovi valori e proporre quindi un effetto di traslazione temporale all'utente.

- **Assi di riferimento:** in un grafico di qualunque tipo sono fondamentali gli assi di riferimento. Nel nostro caso, poi, era necessario che gli assi fossero correttamente graduati secondo le impostazioni dell'utente e che risultassero inerti rispetto all'effetto scorrimento.

- **Impostazione dei limiti di controllo:** ovviamente era richiesta la possibilità dell'utente di fissare i limiti di controllo dei dati acquisiti, indispensabili per distinguere valori in controllo da valori fuori norma, oltre che mostrare gli stessi vincoli sul grafico.



La soluzione non è stata semplice ed ha richiesto diverse sforzi, risultando in un componente di circa 800 righe di codice.

Istruzioni Utilizzate

Riportiamo a scopo informativo le istruzioni di *GDI+* più significative utilizzate nello sviluppo del componente grafico.

```
Graphics g = pb.CreateGraphics();
```

In questo modo si dichiara un oggetto *Graphics* e lo istanzia a partite dalla *PictureBox* pb. L'oggetto *Graphics* è il cuore del sistema grafico di *.Net*, e permette di disegnare praticamente qualunque cosa. È importante sottolineare che questo oggetto non può essere istanziato con un normale costruttore ma solo attraverso metodi particolari. Nel nostro caso, trattando *Windows Forms*, abbiamo fatto ricorso alla storica *PictureBox*, che possiede appunto un metodo adatto a creare una *Graphics* utile per disegnare al suo interno.

```
g.Clear(Colore);
```

Con il metodo *Clear* è possibile cancellare il contenuto della immagine associata alla Graphics e riempirla con il colore di sfondo indicato.

```
Point p= new Point([x],[y]);
```

Crea un oggetto *Point*, che rappresenta un punto nel piano. È questo un altro oggetto fondamentale del namespace *System.Drawing*, utilizzato da moltissime classi e metodi.

```
Pen LinePen = new Pen([Colore], [Spessore]);
```

Creazione di un oggetto penna da utilizzare per disegnare oggetti grafici. Il tipo di penna utilizzato determina sia il colore delle linee sia il loro spessore.

```
g.DrawLines([Penna da utilizzare],[Vettore di Punti]);
```

Serve per disegnare una linea spezzata che congiunge i punti contenuti nel Vettore. Questo metodo si è rivelato particolarmente efficace nel disegno del grafico a linee.

```
g.FillRectangle([Colore], [x1], [x2], [Altezza], [Larghezza]);
```

Specificando colore, coordinate dell'angolo superiore a sinistra, Larghezza e Altezza è possibile riempire un rettangolo.

```
g.DrawString([Testo], [Font],[Colore],[coordinate]);
```

Specificando testo, colore, coordinate dell'angolo superiore a sinistra e Font è possibile scrivere una stringa. Questa funzionalità è stata invece utile per creare la scala di valori sugli assi di riferimento e sui limiti di controllo.

```
g.SmoothingMode=System.Drawing.Drawing2D.SmoothingMode.[Modalità];
```

SmoothingMode permette di impostare la qualità di rendering della *Graphics* o di un *GraphicsContainer*, migliorandone la resa visiva.

```
GraphicsContainer gc = g.BeginContainer();
...
[Istruzioni relative al graphicsContainer]
...
g.EndContainer(gc);
```

I *graphics container* sono utili per impostare particolari parametri di *rendering* solo su un sottoinsieme di elementi grafici della *Graphics*. In questo modo migliorano le performance.

Effetto scorrimento

Per realizzare lo spostamento della linea è stato necessario procedere nel seguente modo:

1. Disegnare la linea con i vecchi valori con colore di sfondo
2. Traslare tutti i punti contenuti nel vettore
3. Ridisegnare la nuova linea con colore linea

Vista la dimensione del codice completo dell'algoritmo, non ne riportiamo il listato.

Gestione dei limiti di controllo

È possibile modificare i limiti di controllo con due *Slider* posti a lato della *PictureBox*. Quando si inserisce un nuovo valore con il metodo *NextValue* si verifica che il valore rientri nei limiti di controllo, in caso contrario si scatena l'evento *OutOfControl*.

Creazione di un Evento

Come si ricorderà, l'ultimo punto nel processo di esecuzione è la segnalazione di allarme in caso di valori fuori controllo. Dal un punto di vista implementativo, questa funzionalità di realizza tramite un *evento* scatenato da ADCChart quando uno dei valori cade al più fuori del range valido.

Innanzitutto è necessario dichiarate un *delegate*, un puntatore a funzione, che descriva la tipologia di evento.

```
public delegate void OutOnControlEventDelegate(object sender, EventArgs e);
```

Quindi, si dichiara all'interno della classe un membro pubblico con la parola chiave *event*.

```
public partial class Chart : UserControl
      {
            public event OutOfControlEventDelegate OutOfControl;
            ...
            ...
            ...
      }
```

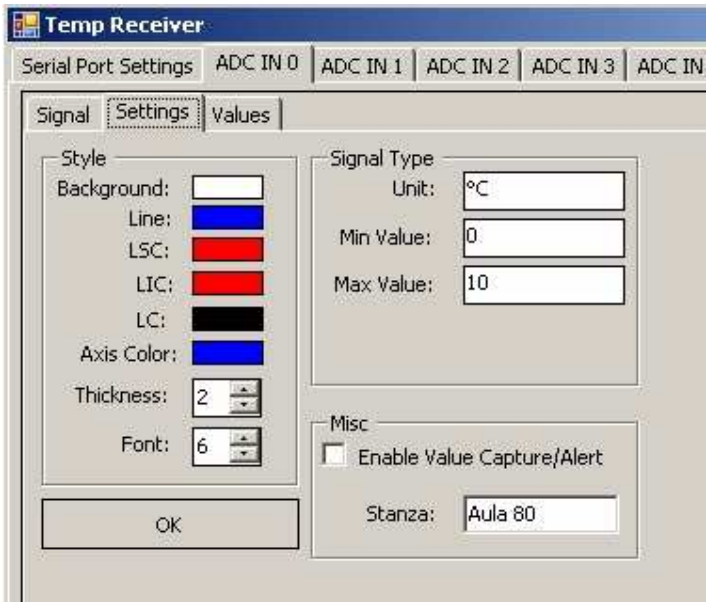
Quando si verifica una circostanza anomale, il flusso di codice scatena l'evento, nel seguente modo, come si trattasse di una normale funzione.

```
OutOfControl();
```

In effetti, quando si inserisce un ADCChart nel form e si associa all'evento *OutOfControl* un *handler*, non si fa altro che definire la funzione puntata dall'evento stesso nel codice originale della classe.

Opzioni

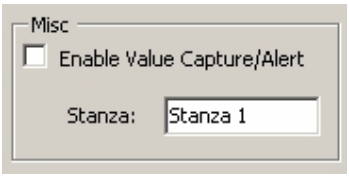
Sempre nell'ottica di fornire all'utente un controllo flessibile e scalabile secondo le necessità, abbiamo anche inserito una schermata che consente di impostare le opzioni possibili per l'ADCChart.



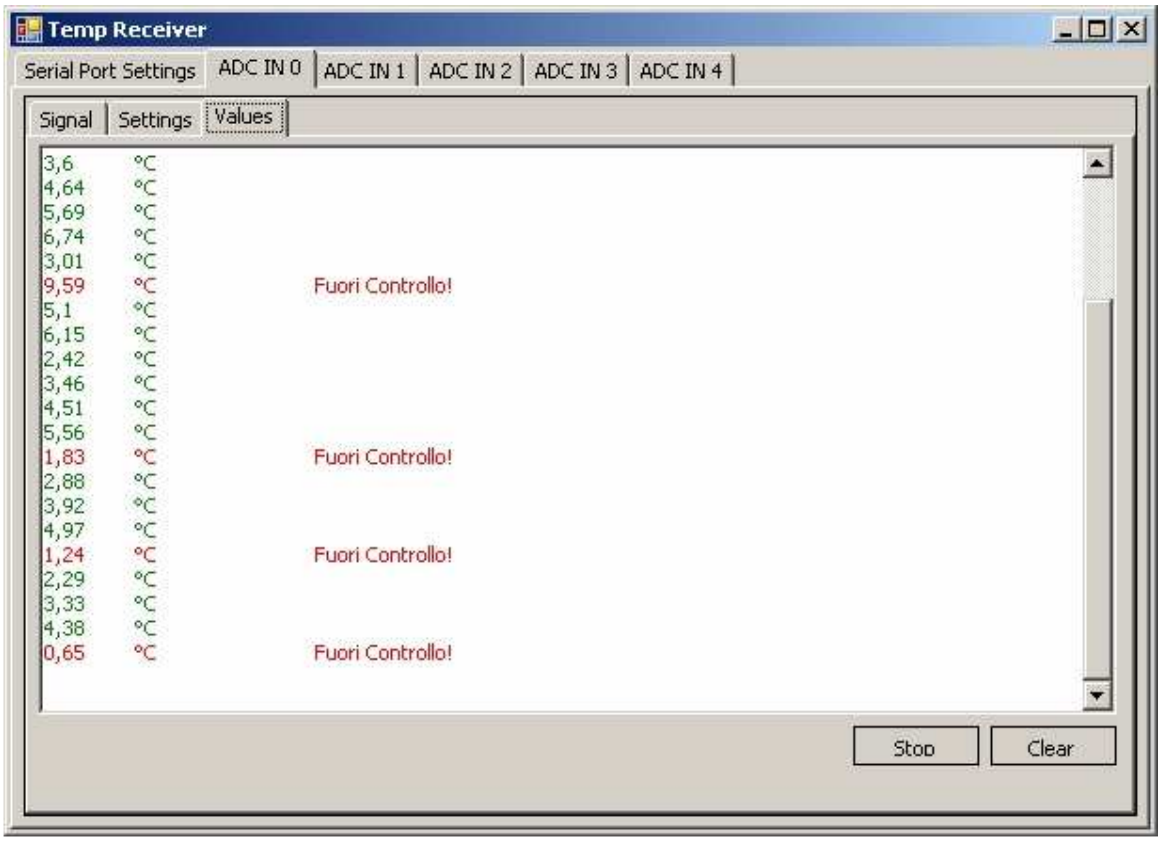
Abbiamo già illustrato il significato del gruppo *Signal Type*, adatto alla personalizzazione del segnale in ingresso ed illustreremo tra poco il gruppo *Misc*. Dando uno sguardo al gruppo *Style*, invece, possiamo notare come sia possibile regolare ogni aspetto del grafico, dai colori di ogni suo componente (sfondo, assi, limiti, etc…) fino allo spessore delle linee alla dimensione del

carattere per i valori mostrati sugli assi e sui limiti di controllo.

Il Server di monitoring



ADCChart consente anche di tenere traccia di ogni valore acquisito, riportando una distinta nell'apposita scheda Values. Affinché però queste informazioni siano visibili occorre attivare l'opzione contenuta nella scheda *Settings*, *Enable Value Capture/Alert*. In questo modo, quando uno dei valori gestisti dal controllo ADCChart esce dai margini di controllo, viene scatenato l'evento *OutOfControl*, "fuori controllo". Si attiva quindi il *monitor server*, che attiva il protocollo di comunicazione e crea un pacchetto UDP da spedire in rete.



Lo *screenshot* qui riportato mostra il *monitor server*. Se da un lato, infatti, un grafico a linee è più leggibile, sia perché fornisce una visione complessiva dell'andamento dei dati sia per l'intrinseca natura grafica, dall'altro non permette di avere una indicazione precisa dei dati ricevuti. In tal senso, allora, abbiamo deciso di permettere la lettura diretta delle misure numeriche dei dati acquisiti presentandoli in una tabella, sempre distinguendo tra valori considerati "normali" e valori fuori controllo (i dati mostrati sono esemplificativi e generati casualmente dal software).

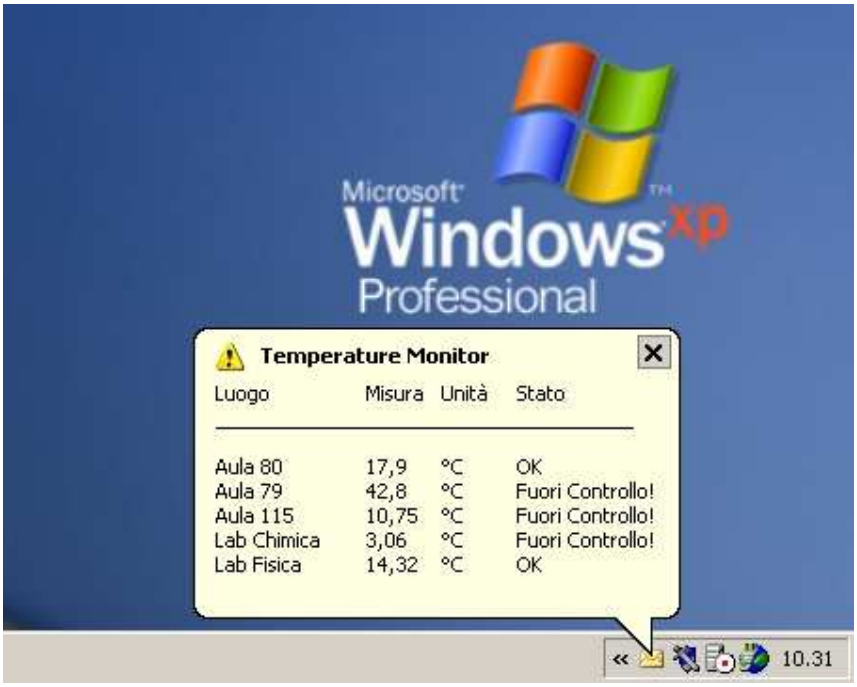
Notifica via UDP

I cinque UserControl hanno lo stesso handler per l'evento *OutOfControl*. Ne riportiamo il codice:

```
private void adcAll_OutOfControl(object sender, EventArgs e)
        {
                UdpClient udpsender = new UdpClient();
                udpsender.EnableBroadcast = true;
```

```
                udpsender.Connect(IPAddress.Broadcast, 6000);
                Byte[] b = Encoding.UTF8.GetBytes(
                    "Luogo\t\tMisura\tUnità\tStato\r\n"
                    +"_____\r\n\r\n"
                    +adc0.Message
                    +adc1.Message
                    +adc2.Message
                    +adc3.Message
                    +adc4.Message
                    );
                udpsender.Send(b, b.GetLength(0));
                udpsender.Close();
        }
```

Client di notifica



L'immagine descrive efficacemente le funzionalità del *Temperature Monitor*: l'applicativo funziona in *background*, senza mostrare finestre, e si limita ad inserire una piccola icona (*tray icon*) vicino all'orologio di Windows. Nel momento in cui il server notifica una situazione di pericolo, dall'icon si apre un *popup* che fornisce un prospetto riassuntivo della situazione, ponendo particolare attenzione su quegli ambienti le cui condizioni sono considerate fuori norma.

Si nota anche in questo caso la scalabilità del servizio: a lato di ciascun ambiente viene segnalato *un valore*, corrispondente alla grandezza fisica misurata, con la relativa unità di misura impostata sul server.

Da un punto di vista implementativo, il *client* utilizza un particolare oggetto di *.Net*, il *BackgroundWorker*. Si tratta di utile e versatile controllo che permette di avviare un *thread* che possa funzionare in secondo piano (*background*), nel nostro caso il servizio di ascolto dei messaggi in arrivo dal *server*.

```
private void Listener()
{
    udpc = new UdpClient(6000);
    udpc.EnableBroadcast = true;
    while (true)
```

```
        {
         IPEndPoint RemoteEndPoint = new IPEndPoint(IPAddress.Any, 6000);
         byte[] received = udpc.Receive(ref RemoteEndPoint);
         string Data = Encoding.UTF8.GetString(received);
         if (!string.IsNullOrEmpty(Data))
         {
              nico.ShowBalloonTip(10000, "Temperature Monitor", Data,
System.Windows.Forms.ToolTipIcon.Warning);
         }
    }
}
```

La spiegazione è abbastanza semplice. Viene innanzitutto istanziato un oggetto *UDPClient*, necessario per accedere alla rete. La porta specificata è la stessa utilizzata dal server per inviare i dati. La proprietà *EnableBroadcast* è necessaria per abilitare *UDPClient* alla ricezione di pacchetti *broadcast*, come quelli diffusi dal server. Ora, la funzione entra in un blocco ciclico infinito: il thread tenta continuamente di ricevere dati da un qualunque mittente (`IPAddress.Any`), non conoscendo a priori l'indirizzo IP del server. Se il messaggio letto non è vuoto, quindi, mostra un popup con il prospetto riassuntivo ricevuto.

Prospettive future

Per ragioni di tempo non siamo riusciti a sfruttare fino in fondo le potenzialità offerte dai microcontrollori, ma ci riserviamo questo spazio per descrivere alcuni possibili miglioramenti applicabili alla scheda di acquisizione.

1. **Aumento delle linee d'ingresso**: sostituendo l'attuale PIC 16F870 con il 16F877, ad esempio, si può arrivare fino a 10 ingressi, raddoppiando di fatto la capacità della scheda di acquisizione.

2. **Incremento della precisione dei convertitori AD:** sempre utilizzando un modello migliore, è possibile aumentare la precisione fino a 12 bit, e quindi 4096 livelli, un campionamento 4 volte più preciso rispetto ai 10 bit attuali.

3. **Connessione USB:** la porta RS232, sebbene comoda vista la sua semplicità d'uso, sta via via scomparendo dai moderni PC, in favore di connessioni più veloci e scalabili quali USB. Abbiamo già illustrato come questa tecnologia abbia permesso di fornire un'alimentazione adeguata alla scheda, eliminando l'obbligo di un alimentatore, ma esiste anche la possibilità di utilizzarla per la trasmissione dei dati. In tal senso, esistono due soluzioni.

    a. Una prima prevede il riutilizzo del firmware esistente, basato sulle istruzioni **`SerOut`** e **`SerIn`**, con un nuovo dispositivo driver, l'FT232, un adattatore di livello USB. Questi non fa altro che utilizzare USB come fosse una porta COM e, previa l'installazione di un driver di periferica virtuale, permette di mantenere il software su PC.

    b. Una seconda soluzione, ben più interessante, che punta sull'impiego di una specifica serie di PIC, la 16C, che integra al suo interno un sottosistema USB vero e proprio, in grado di gestire sia la comunicazione USB reale sia di fornire il supporto *Plug'n'Play*. Il risultato sarebbe eccezionale, ma richiederebbe comunque la riprogettazione di parte del firmware e del software. Nel primo occorrerebbe semplicemente impiegare le istruzioni **`UsbOut`** e **`UsbIn`**, già predisposte per la trasmissione USB. Nel secondo,

invece, occorrerebbe reperire un componente di terze parti in grado di sostituire la classe *SerialPort* nella parte di acquisizione dati. Le ricerche da noi svolte hanno permesso di individuarne alcuni, ma nessuno di questi sembra garantire la medesima chiarezza e semplicità d'uso di *SerialPort*.

4. **Pilotaggio di dispositivi:** un'altra interessante possibilità sarebbe quella di creare un flusso di dati complementare dal PC verso il PIC, nell'ottica di fornire una risposta ad una segnalazione di allerta. Un esempio potrebbe essere l'accensione di un ventilatore oppure del sistema antincendio di un locale nel caso la temperatura salga oltre i limiti previsti.

5. **Programmatore USB:** il discorso appena fatto circa il declino della porta seriale ed i vantaggi della connessione USB valgono anche per il programmatore stesso. Sostituendo l'attuale collegamento con una porta USB e modificando lo schema circuitale si otterrebbe un dispositivo veramente efficiente e compatibile con ogni moderno computer. Esistono però due problematiche: primo, il programmatore necessità di una tensione elevata, 13 V, per effettuare il RESET del PIC, e secondo occorre reperire un *nuovo loader* compatibile con lo standard USB. Se per il primo inconveniente esiste già una soluzione, PIC programmabili a basse tensioni, è invece più ostico risolvere il secondo…

Fonti/Bibliografia

Sito	Argomento Trattato
http://www.microchip.com	Azienda Produttrice dei PIC
http://www.maxim.com	MAX232