Steve Blackbird (TardoFreak)

PSCU01A - USB SWITCH CONTROL UNIT

11 March 2011

Introduzione

L' ultimo mio articolo descriveva una **Switch Control Unit** realizzata con un PIC16F877 ideata per essere utilizzata principalmente come dispositivo hardware d' ingresso/uscita per un personal computer. La comunicazione fra l' unità ed il computer avveniva tramite una **linea di comunicazione seriale RS232C**. Purtroppo questa linea di comunicazione è oggi obsoleta e sostituita dalla ormai onnipresente USB. Non ostante si trovino in commercio una moltitudine di adattatori USB-RS232, un' unità di comando e di acquisizione dati che si colleghi direttamente al bus USB sarebbe una soluzione migliore. Se poi questa unità avesse la possibilità di essere collegata ad un display alfanumerico e permettesse l' acquisizione di segnali analogici si potrebbero realizzare parecchie applicazioni interessanti.
Questo articolo è la recensione di un circuito integrato studiato per questo scopo.
Diverse aziende producono firmware e molte di queste vendono microcontrollori programmati per eseguire queste funzioni particolari. La **SANGON** produce questo circuito integrato e questo articolo lo analizza nel dettaglio. Anche i produttori di microcontrollori offrono micro programmati per svolgere compiti particolari. Un esempio è l' MCP2200 della Microchip, un chip di conversione USB-RS232 realizzato su un PIC e venduto come integrato dedicato.
La classe di dispositivi USB chiamata **CDC** aiuta la transizione fra la gloriosa **RS232** e l' USB rendendo il dispositivo gestibile come una porta seriale virtuale. Un esempio di gestione delle porte seriali virtuali è illustrato in **questo articolo** utilizzando il linguaggio **Java**.

Il PSCU01A

E' la sigla di questo chip realizzato dalla Sangon con un microcontrollore **PIC18F14K50** della Microchip e si comporta come una linea seriale virtuale mediante un collegamento USB full speed. Tramite questo chip si possono azionare uscite digitali, relè, leggere ingressi digitali ed acquisire segnali analogici. Può comandare un modulo LCD alfanumerico costruito intorno ad un **controller HD44780U** compatibile, può essere interfacciato tramite SPI con un circuito integrato per la misura della temperatura e può pilotare un driver per LED **MM5450** collegato con interfaccia microwire. In contenitore a 20 pin DIL può essere montato su zoccolo o saldato direttamente in un circuito stampato di tipo tradizionale (non

SMD) e quindi adatto anche ad applicazioni hobbistiche.
Nota: per brevità il modulo. da qui in avanti, lo indicherò semplicemente con la sigla **SCU** che sta per **S**witch **C**ontrol **U**nit.

Le applicazioni

Le applicazioni possibili con questo chip sono molte. Si va dalla semplice periferica USB che aziona relè e legge lo stato di contatti o ingressi digitali, al dispostivo complesso che permette di leggere tensioni analogiche, visualizzare messaggi su un display LCD alfanumerico, rilevare temperature mediante un sensore e pilotare un massimo di 34 LED o carichi compatibili come, ad esempio, optoisolatori a corrente costante. In pratica vengono sfruttate appieno le caratteristiche di I/O digitali ed analogiche del microcontrollore e cioe':

- 1 ingresso digitale
- 4 I/O digitali
- 8 I/O digitali programmabili come ingressi analogici con risoluzione 10bit. Tre di questi possono funzionare come ingressi trigger per la cattura di segnali digitali.

Il collegamento al modulo LCD alfanumerico avviene utilizzando 6 delle linee di I/O. Il sensore di temperatura **TC77** utilizza una linea di CS che può essere scelta fra quelle disponibili, una linea dati ed una linea di clock condivisa con l' eventuale modulo LCD. Il driver **MM5450** utilizza due linee (clock e dati) condivise con l' eventuale modulo LCD ed una linea di ENABLE scelta fra quelle disponibili.
Per le misure di tensione si può scegliere fra 3 tensioni di riferimento di precisione interne o utilizzare una tensione di riferimento esterna.
E' quindi possibile realizzare interfacce per collaudi, stazioni per domotica, rilevazione della temperatura di circuiti di potenza, azionamenti di potenza controllati dal PC, effetti luminosi con un massimo di 34 uscite per LED o per attuatori di potenza optoisolati. Dipende molto dalla fantasia dell' utilizzatore. Pilotando opportunamente i pin di I/O tramite un programma su PC ci si può sbizzarrire e collegare una moltitudine di circuiti integrati, espandere il numero di ingressi e uscite e quant' altro. L' unico limite è rappresentato dal numero di pin dell' integrato.

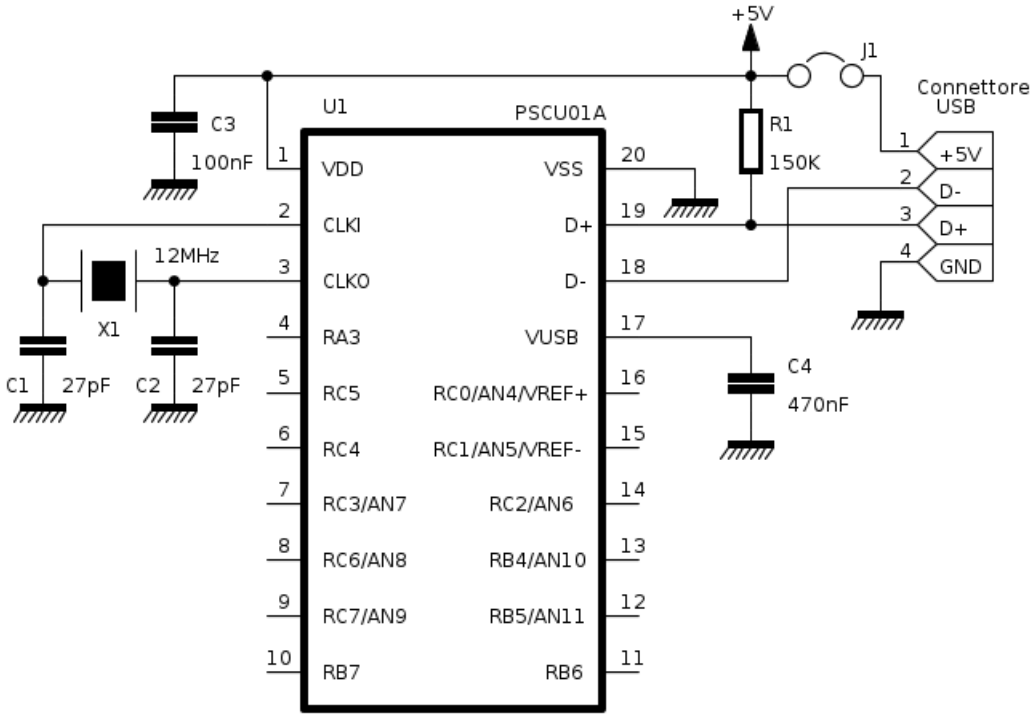
Come si comanda

Il chip è visto dal computer come una linea seriale virtuale ed i comandi sono inviati sotto forma di stringhe di caratteri seguiti da un carattere <CR> (il tasto <Invio> della tastiera del PC). I comandi sono di facile comprensione, di tipo "umano", infatti è possibile pilotarlo tramite un programma di emulazione di terminale come l' **HyperTerminal** di Windows o con un programmino di terminale realizzato da me

e scritto in Java. Il modulo risponde con OK o messaggi di errore ed è implementata una semplice funzione di editing della linea di comando.
Nota: La SCU ha la possibilità di inviare l' eco dei caratteri digitati ma all' atto dell' accensione questa funzione non è abilitata, quindi il primo comando che deve dare per poterlo utilizzare in modo manuale deve essere "ECHO ON". Mentre lo si digita sul terminale non compare nessuna scritta ma la SCU lo riceve. Dopo aver dato l' invio premendo il tasto <Invio> la SCU effettuerà l' eco dei caratteri e quindi sarà possibile vedere i comandi inviati.

Il circuito minimo

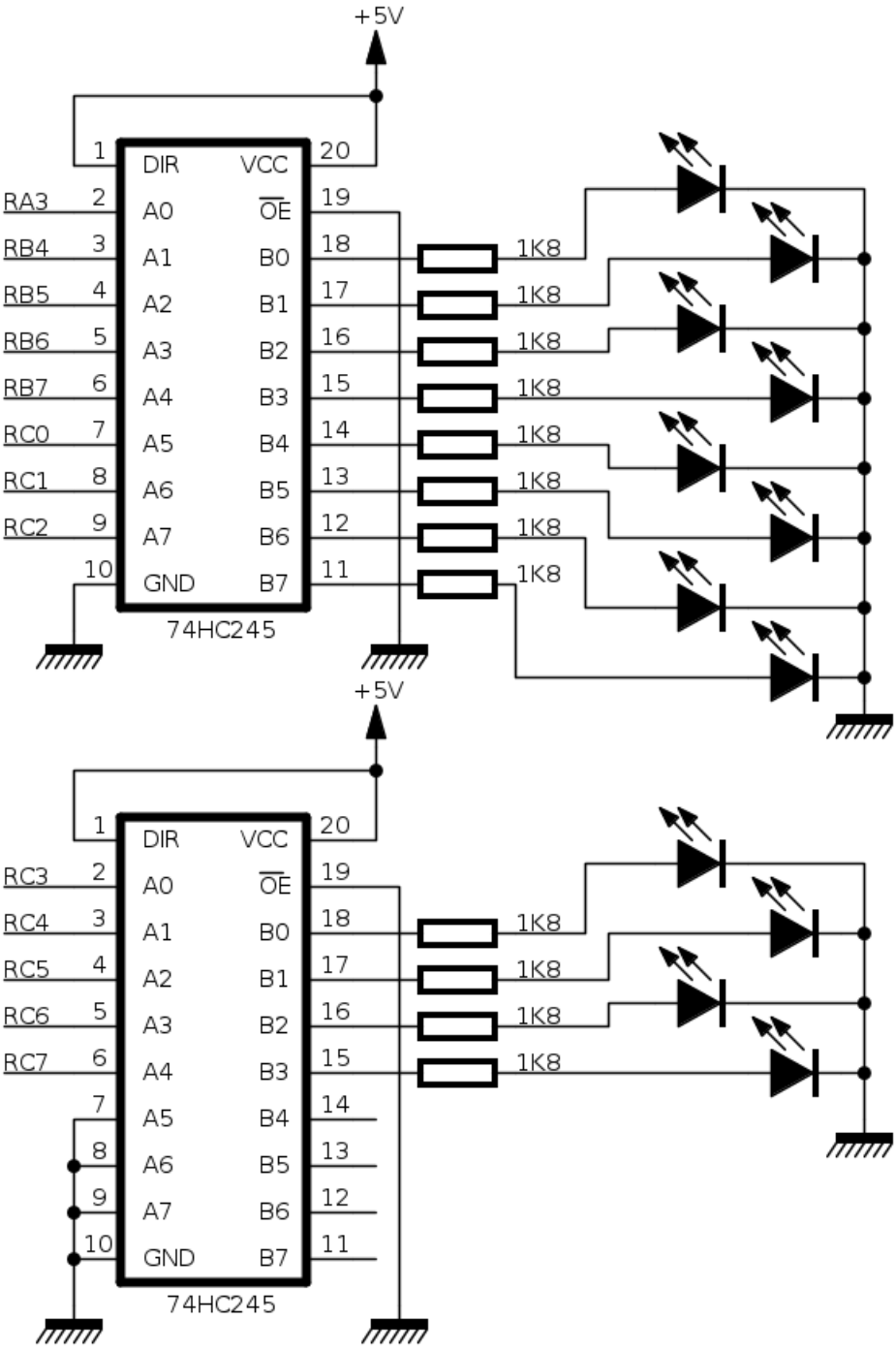
Il primo passo è quello di realizzare un circuito che funzioni. Il chip in questione è fatto per essere collegato al USB e si deve comportare come una porta seriale virtuale. Il circuito minimo per farlo funzionare è illustrato nella figura qui sotto ed è composto da: Il chip PSCUA01
un quarzo da 12MHz
due condensatori da 27pF
una resistenza
un condensatore da 100nF per filtrare l' alimentazione
un condensatore da 470nF necessario per il regolatore interno della tensione USB
Il circuito si può montare su una piastra millefori o una bredboard. Io, per ragioni "storiche" preferisco sempre montare i circuiti sul millefori ed utilizzare la tecnica di wire-wrap per i collegamenti. Il BUS USB fornisce l' alimentazione. Ho comunque montato un jumper (J1) per potere eventualmente alimentare il chip con una tensione esterna ma, a dire il vero, questa opzione non l' ho mai utilizzata durante la sperimentazione, ho sempre e solo utilizzato l' alimentazione fornita dal USB.



Una volta montato il circuito lo si collega ad una presa USB. Il prototipo che ho realizzato utilizza una cavo USB recuperato da una vecchia tastiera di qualità dozzinale e collegato ad un connettore SIL. Una soluzione semplice ma sufficiente per provare il componente. Il PC utilizzato gira sotto WindowsXP quindi richiede l' installazione del driver. Se il PC ha richiesto il driver vuol dire che il circuito funziona e, una volta installato il driver, il circuito è pronto a svolgere le sue funzioni.
Come suggerito dal datasheet si individua la porta COM (pannello di controllo -> sistema-> hardware -> Gestione periferiche -> Porte COM e LPT) che corrisponde al dispositivo e si lancia HyperTerminal oppure il programmino JavaTerm. Quando si è connessi alla seriale virtuale si possono inviare comandi.

Monitorare i pin

Per monitorare i piedini del chip senza caricarli ho montato due integrati **74HC245** che pilotano dei LED. In questo modo riesco a visualizzare gli stati degli ingressi e delle uscite senza caricarle con LED o resistenze, cosa utile nel caso che si usino ingressi analogici. Questo circuito non è strettamente necessario ma mi permette di evitare di verificare lo stato dei pin con l' oscilloscopio, operazione tutt' altro che comoda.



I comandi di base

Con il programma di terminale aperto ed il modulo connesso possiamo iniziare a sperimentare alcuni comandi. In una periferica come questa la prima cosa da fare è impostare i pin I/O. Per risucire a vedere i comandi che inviamo è necessario, dopo l' accensione, inviare il comando **ECHO ON** che abiliterà il ritorno dei caratteri. All'

atto dell' accensione tutti i pin sono configurati come ingressi digitali. Per impostare la funzione di un pin si utilizza il comando "PIN" la cui sintassi è:

PIN <porta><bit> <tipo>

Il valore di <porta> deve essere "B" o "C" poiché la porta A ha solo un pin disponibile RA3 e che è un ingresso digitale. il carattere <bit> è un numero che va da 0 a 7 ed il campo <tipo> può essere "IN", "OUT" o "AN" che indicano rispettivamente ingresso digitale, uscita digitale, ingresso analogico.
Nella pedinatura dell' integrato i pin delle porte di ingresso uscita sono indicati con R, ad esempio il bit 0 della porta C viene indicato RC0, quindi se volgiamo indicare il pin RC3 il valore di <porta> sarà C ed il valore di <bit> sarà 3. Ora supponiamo di voler configurare il pin RC3 come un' uscita digitale, dobbiamo quindi inviare il comando:

PIN C3 OUT

seguito dal tasto <Invio> (carattere <CR> codice ASCII 13). La SCU risponde con un OK. Da questo momento in poi il pin RC3 si comporterà come un uscita digitale. Lo possiamo verificare facilmente inviando l' appsito comendo per portare l' uscita a livello logico 1 che, nel prototipo che ho realizzato, farà accendere il led corrispondente al pin RC3. La sintassi del comando che attiva l' uscita digitale e:

SET <porta><bit>

Digitando il comando

SET C3

seguito da <Invio> la SCU risponderà con un OK e, meraviglia dell' elettronica ... il led corrispondente si accenderà eh eh eh. A questo punto proviamo a spegnerlo, cioè a mettere a 0 logico il pin RC3. La sintassi è questa:

CLR <porta><bit>

E quindi digitando il comando

CLR C3

sempre seguito da <Invio> (e da ora in poi non lo ripetero' più) assisteremo ad un fenomeno eccezionale: il LED che si spegne. :)
L' ultimo dei comandi di base è quello che serve per leggere lo stato degli ingressi che è:

GET <porta><bit>

Quindi prendiamo uno spezzone di filo, colleghiamo il pin 4 dell' integrato cioè RA3 al +5V ed inviamo il comando

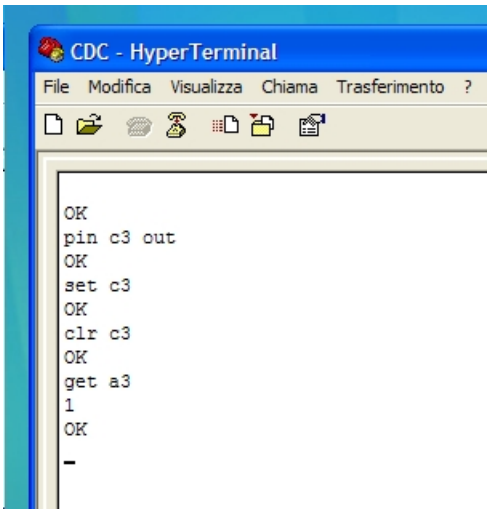
GET A3

e la SCU risponderà con

1

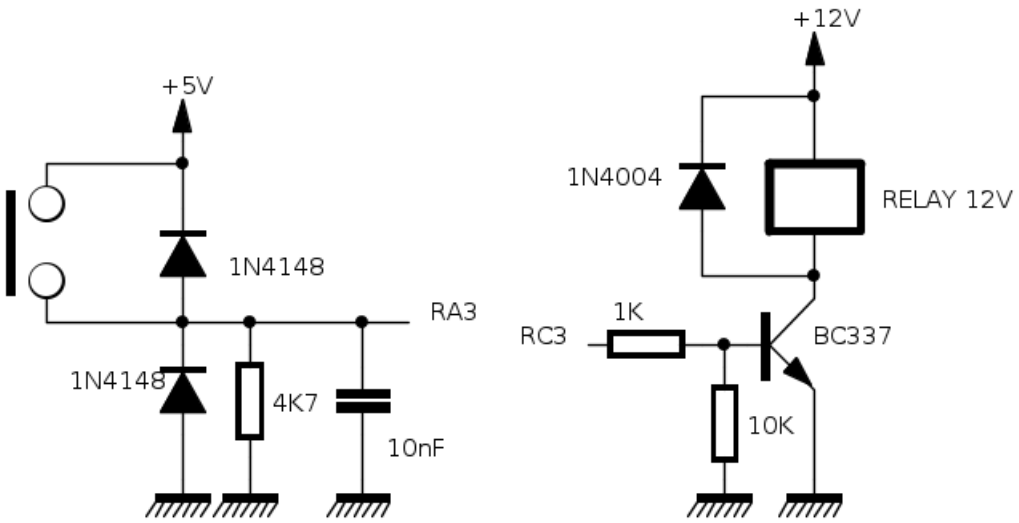
OK

Qui di seguito riporto lo screen shot del terminale con i comandi che abbiamo eseguito.



screen_com_base.jpg

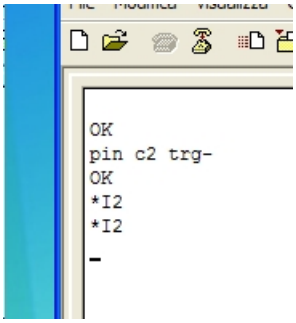
Dall' immagine possiamo notare alcune cose. I comandi sono stati scritti utilizzando caratteri minuscoli. La SCU non fa differenza fra maiuscole o minuscole eccetto in un comando che vedremo più avanti. Per dirla tutta la SCU ignora anche gli spazi fra i vari campi, semplicemente li elimina quindi scrivere "SET C3" o "s e t c 3" o anche "SeTc3" è la stessa cosa, il comando funziona comunque. Se si sbaglia a scrivere un carattere si può utilizzare il tasto backspace <BS> che nelle tastiere e' in alto a sinistra indicato con una freccia <-.
Bene, a questo punto abbiamo tutto quello che ci serve per creare, ad esempio, una basetta con dei relè che si possono pilotare tramite PC oppure leggere contatti/pulsanti. Per azionare i rele tramite le uscite dei PIC c'è il solito circuito realizzato con il glorioso BC337 mentre per gli ingressi basta una resistenza che mantenga uno stato stabile. Se vogliamo fare i fighi possiamo metterci anche un paio di diodi di protezione ed un condensatorino da 10nF per eliminare eventuali disturbi. Questi sono gli schemi per un ingresso a pulsante protetto che, in condizione di riposo da uno zero logico ed un' uscita che pilota un relè.



Nel disegno ho indicato i pin che ho utilizzato nell' esempio dei comandi: RA3 come ingresso e RC3 come uscita digitale.

Gli ingressi trigger

I tre pin RC0, RC1 e RC2 possono essere utilizzati come ingressi sensibili ai fronti in grado di catturare eventi esterni in modo asincrono senza la necessità di leggerli ripetutamente e continuamente (polling). Per impostare l' ingresso come trigger si usa sempre il comando PIN ma al posto di impostare <tipo> come AN, IN o OUT lo impostiamo come **TRG+** per sentire i fronti di salita e **TRG-** per i fronti di discesa. Per esempio, se vogliamo che il pin **RC2** sia un ingresso trigger sensibile ai fronti di discesa dovremo inviare il comando **PIN C2 TRG-**. Da quel momento in poi, quando la SCU rivelerà un fronte di discesa sull' ingresso RC2 invierà la stringa *I2 seguita chiaramente dai caratteri<CR> e <LF> di ritorno a capo. Nello screenshot qui sotto si vede come reagisce la SCU ai fronti di discesa sull' ingresso RC2.



screen_trg2.jpg

Dopo che il pin è stato configurato come ingresso trigger ho inviato due segnali sul pin RC2. Appena la SCU ha sentito il primo fronte di discesa ha inviato sul terminale la prima scritta *I2, la seconda l' ha inviata al secondo fronte di discesa che ha intercettato. La funzione di trigger può servire per misurare tempi o per evitare di campionare continuamente gli ingressi. Supponiamo, ad esempio, di voler misurare il tempo impiegato da un' automobile che passa attraverso sue barriere a fotocellula. Una prima barriera la potremmo collegare, tramite opportuna circuiteria, all' ingresso RC0 e la seconda all' ingresso RC1, e che il passaggio attraverso le barriere generi un segnale digitale e noi dobbiamo sincronizzarci con il suo fronte di salita. Per fare questo dobbiamo inviare alla SCU questa serie di comandi:

PIN C0 TRG+

PIN C1 TRG-

Al passaggio attraverso la prima barriera la SCU invierà al programma di monitoraggio la stringa

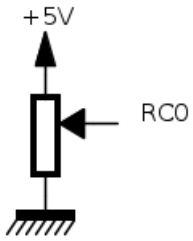
***I0** seguita da i caratteri <CR> e <LF>

A questo punto il programma fa partire un timer che fermerà quando riceverà la string

***I1** seguita da i caratteri <CR> e <LF>

Gli ingressi analogici

La SCU contiene un convertitore Analogico/Digitale a 10 bit e, come ogni convertitore, effettua la conversione in relazione ad una tensione di riferimento. All' atto dell' accensione la SCU utilizza come tensione di riferimento negativa il GND, cioè 0V e come riferimento positivo la tensione di alimentazione VDD. Questa è la condizione tipo per rilevare la posizione di potenziometri per esempio. Un potenziometro collegato fra VDD e GND con il cursore collegato ad un ingresso analogico permetterà di ottenere un valore che va da 0 (potenziometro ruotato verso GND) e 1024 (potenziometro ruotato verso la VDD). Per provare velocemente la funzionalità degli ingressi analogici è quindi sufficiente un potenziometro. Il valore di resistenza non deve essere molto elevato infatti l' impedenza d' ingresso è piuttosto bassa e si aggira intorno ai $25K\Omega$, quindi per stare tranquilli è meglio scegliere un potenziometro di basso valore, ad esmpio da $1K\Omega$. Colleghiamo quindi il potenziometro come indicato nello schema e procediamo con la prova.



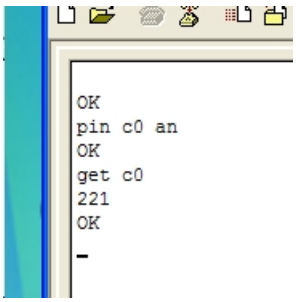
Come abbiamo visto in precedenza nei comandi di base, per impostare un ingresso come ingresso analogico dobbiamo inviare il comando PIN. Per esempio per impostare il pin RC0 indicato nello schema come ingresso analogico il comando da inviare è:

PIN C0 AN

Il comando per leggere gli ingressi analogici è lo stesso che si usa per gli ingressi digitali, solo che il risultato non sarà 1 o 0 ma un valore che va da 0 a 1023. Quindi per leggere il valore dell' ingresso RC0 il comando da inviare è:

GET C0

Ora ruotiamo il potenziometro a circa un quarto della corsa (cursore verso GND), inviamo il comando e la SCU risponderà con un numero che dovrebbe essere circa 250. Qui di seguito ho riportato lo screenshot della prova effettuata.

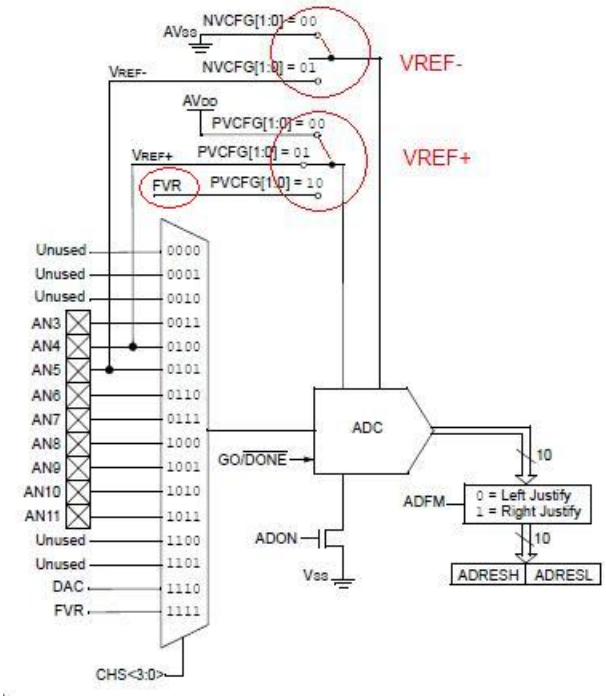


screen_an01.jpg

Chiaramente se si cerca di impostare un pin per una funzione che non è prevista la SCU, invece di rispondere con **OK** risponderà con **EF** che sta a significare un errore di funzione.

Le tensioni di riferimento

Il PIC su cui è implementata la SCU prevede diverse configurazioni della tensione di riferimento del convertitore A/D, per capirla meglio questo è lo schema del modulo interno di conversione. Ho evidenziato in rosso le due parti che permettono la scelta delle tensioni riferimento per il convertitore. La possibilità di scegliere diverse configurazioni rende questo modulo molto flessibile ed in grado di fare cose molto simpatiche ed utili. Il resto non ci interessa perché è gestito dal programma della SCU che si fa carico di tutti i bit e registri facendo per noi il resto del lavoro.



PIC_ADC.jpg

Al momento dell' accensione il modulo, come accennato in precedenza, usa il VSS (la massa elettrica) come tensione di riferimento negativa e la VDD (il +5V di alimentazione) come tensione di riferimento positiva. I due trattini rossi all' interno dei due circolini indicano la posizione dei selettori all' accensione. Un circolino più piccolo indica una sigla estremamente importante **FVR** che significa tensione fissa di riferimento (Fixed Voltage Reference). La tensione di riferimento interna è una tensione di precisione scelta fra due disponibili e cioe' 1.024V, 4.096V tramite la quale si può misurare la tensione agli ingressi analogici come si fa adoperando un multimetro. E' come avere un voltmetro digitale già pronto per l' uso integrato all' interno del micro. Ma andiamo con ordine.
Avere le tensioni di riferimento collegate alla VDD e a VSS va bene per leggere la posizione di un potenziometro ma non va bene per misurare una tensione perché la variabilità della tensione di alimentazione manda a pallino la precisione. L' unica è utilizzare una tensione di riferimento precisa e stabile. Una soluzione è quella di prelevarla dall' esterno ed il micro prevede questa eventualità, infatti dallo schema si vede chiaramente che la VREF+ (riferimento positivo) può essere presa dall' esterno e precisamente dall' ingresso AN4 che corrisponde al pin 16 della SCU (RC0). Anche il riferimento negativo VREF- può essere preso dall' esterno dall' ingresso AN5 che corrisponde al pin 15 della SCU (RC1). LA SCU prevede alcuni comandi che servono proprio per selezionare le tensioni di riferimento che sono:

VREF- <VSS/EXT>

Inviando il comando **VREF- VSS** in pratica si collega la tensione di riferimento

negativa a VSS, condizione che è poi quella iniziale, mentre il comando **VREF- EXT** imposta il pin RC1 come ingresso per la tensione di riferimento negativa. Dopo questo comando il pin in questione non potrà più essere utilizzato come ingresso analogico.

VREF+ <VDD/FVR/EXT>

Questo comando è quello più interessante. Inviando **VREF+ VDD**, come nel comando precedente, ci troveremo ad avere la tensione di riferimento positiva collegata all' alimentazione, che sarebbe poi la condizione iniziale della SCU all' accensione, mentre il comando **VREF+ EXT** imposta il pin RC0 come ingresso per la tensione di riferimento esterna. L' opzione FVR merita qualche attenzione in più. Un esempio può aiutare, il comando:

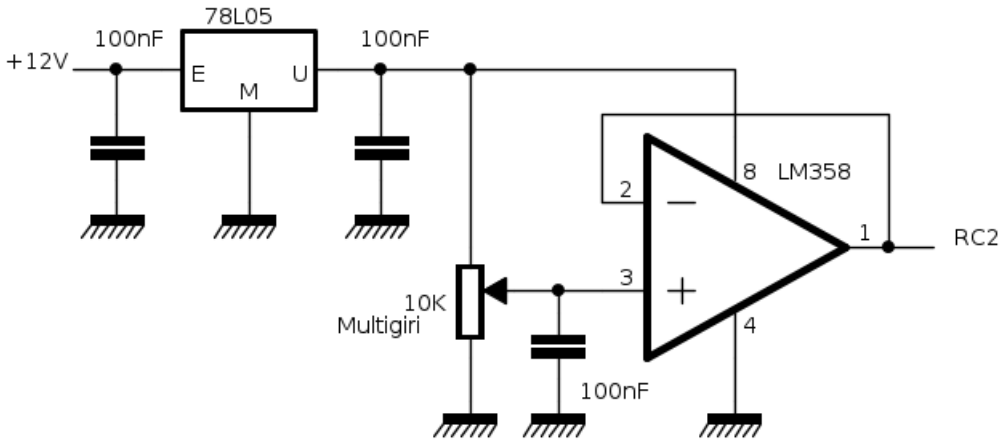
VREF+ FVR 1

Collega la tensione di riferimento positiva alla sorgente interna di precisione a 1.024V, almeno così la definisce la Microchip. 10 bit di risoluzione significa avere 1024 possibilita' (un numero cha va da 0 a 1023) quindi avendo a disposizione una tensione di riferimento a 1.024V potremo misurare tensioni (che vanno da 0V a 1,023V) con risoluzione 1mV. In pratica è come **avere a disposizone un multimetro digitale a 3 cifre e mezza**, con risoluzione 1mV. Questa risoluzione permette, ad esempio, di utilizzare **un sensore di temperatura LM35 che fornisce 10mV °C ed avere una risoluzione di 0.1 °C**.

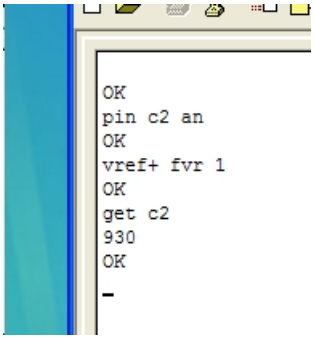
C' è un' altra opzione per la FVR:

VREF+ FVR 4 che seleziona la tensione di riferimento a 4.096V, per misurare tensioni da 0 a 4V con risoluzione 4mV

Per fare qualche prova ho utilizzato degli operazionali collegati come inseguitori di tensione ed alimentati da una sorgente esterna in modo da non sentire le variazioni della tensione che arriva dal BUS USB.



Ho regolato il trimmer con il multimetro a (la lettura mi dava0,9312 V) ed ho inviato i comandi per misurare questa tensione. Il risultato lo si può vedere sullo screenshot del terminale.

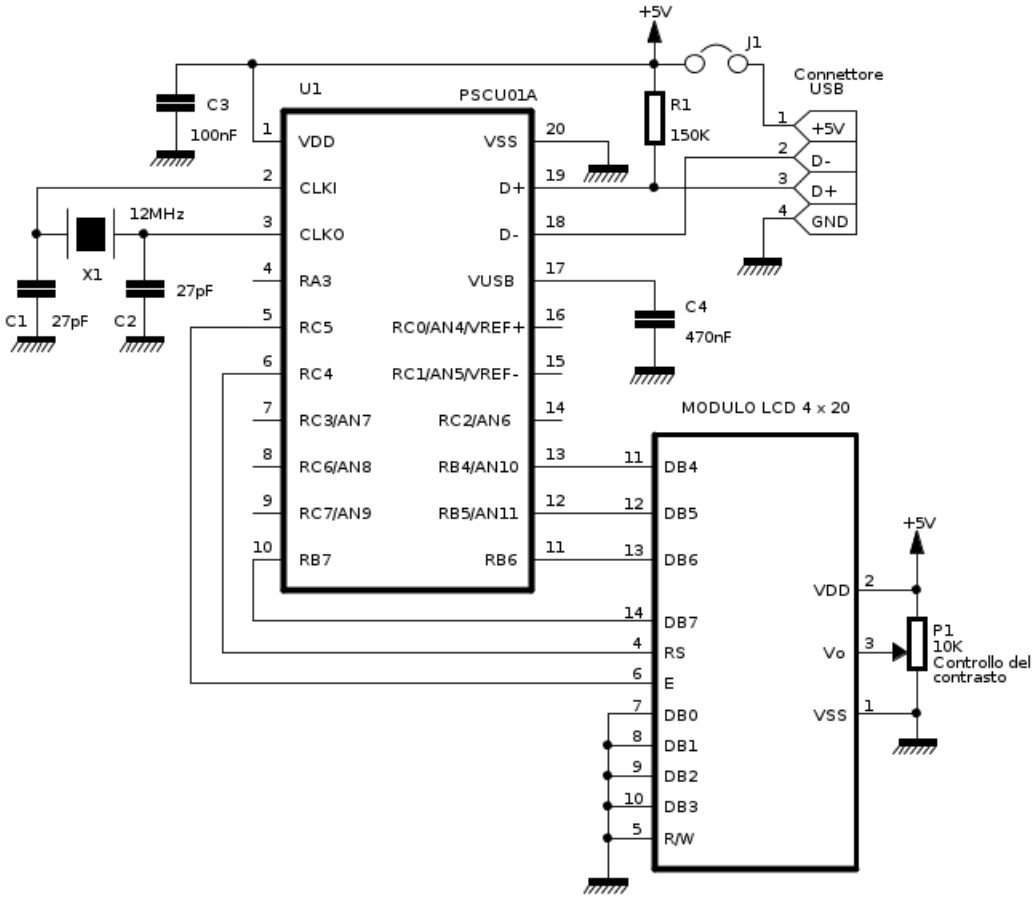


screen_test_fvr.jpg

Nota: per fare misure di tensione accurate è sempre meglio seguire una procedura che permetta di tarare il sistema tramite software utilizzando un ottimo multimetro di precisione. Per valutare opportunamente la precisione e la linearità del modulo di conversione è meglio fare riferimento al datasheet del PIC18F14K50.

Pilotare un modulo LCD

In commercio esisteno parecchi modli display LCD alfanumerici. I più diffusi i 16x1 (una linea da 16 caratteri) 20x1, 16x2, 20x2, 20x4 e 40x2. Ce ne sono diversi modelli con retroilluminazioni di vari colori e di varie dimensioni. Il cuore di questi moduli è un circuito integrato che solitamente è compatibile con il vecchio quanto glorioso HD44780 che provvede al pilotaggio del LCD vero e proprio e si interfaccia facilmente con sistemi a microprocessore o ai microcontrollori. Questo integrato è diventato praticamente uno standard per questi moduli e lo troviamo anche con sigle diverse, come il KS00070, e che si utilizza allo stesso identico modo. Se interfacciare questi display con un micro è cosa semplice, meno semplice è interfacciarli con un PC tramite la porta USB. Esistono in commercio moduli (a volte con circuiti di interfaccia) che si possono comandare tramite USB ma di solito sono abbastanza costosi. La SCU di questo articolo è predisposta per collegarsi ad un modulo LCD HD44780 compatibile per poterlo gestire tramite USB. per le prove ho utilizzato un display dalla Displatech da 4 linee di 20 caratteri che ho collegato (come indicato dal datasheet) in questo modo:



Per comandare il modulo LCD bisogna prima far sapere alla SCU che il modulo è collegato. Per fare questo si usa il seguente comando:

USE LCD <1/2>

il parametro 1 o 2 serve ad indicare alla SCU il numero di linee fisiche del controller. Internamente il controller può segtire una o due linee fisiche di caratteri a seconda del cristallo liquindo che pilota. Ogni linea fisiche contiene 64 caratteri, sufficienti per un display 1x12 o 2x20 ma insufficienti per un display 4x20 per esempio. Nei primi due casi il display utilizzarà una sola linea fisiche, con il 20x4 deve per forza utilizzare due linee fisiche. Per sapere quante linee fisiche vengono utilizzate è necessario leggere il datasheet del modulo LCD. I caratteri sono contenuti all' interno della memoria del modulo ed ogni poszione sullo schermo corrisponde ad un indirizzo di memoria. Il display che ho utilizzato per la prova usa due linee fisiche disposte in modo non proprio intuitivo. La figura qui sotto indica la corrispondenza fra i caratteri sullo schermo e gli indirizzi in memoria. Queste informazioni sono importanti per poter posizionare correttamente le scritte sul display.

0	1	2	3			16	17	18	19
64	65	66	67			80	81	82	83
20	21	22	23			36	37	38	39
84	85	86	87			100	101	102	103

Quindi per inizializzare opportunamente il mio modulo dovrò inviare il comando **USE LCD 2** che cancella il display, porta il cursore nella posizione in alto a sinistra e lo rende invisibile. Per rendere visibile il cursore si usa il comando **CURSOR ON**. Dopo aver inviato questo comando il cursore apparirà come un trattino lampeggiante. Chiaramente esiste anche il comando **CURSOR OFF** per renderlo invisibile ed il comando **LCD CLR** che cancella il display. Il comando per scrivere un testo nel display è:

PRINT "<testo>

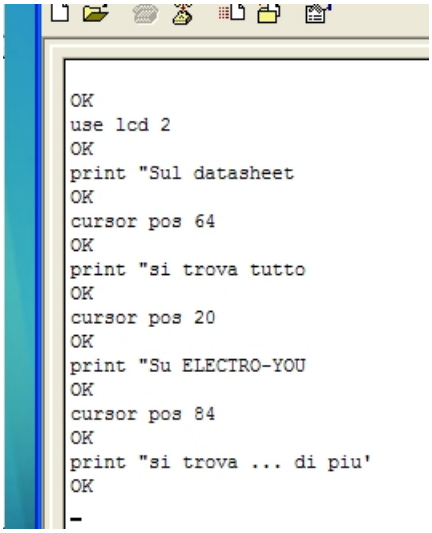
Che visualizza la stringa che segue gli apici a partire dalla posizione del cursore. Il cursore, dopo un comando PRINT sarà posizionato al termine della stringa visualizzata. l' ultimo comando del display serve per posizionare il cursore sullo schermo, la sintassi è:

CURSOR POS <pos>

Dove <pos> è la posizione in memoria. E qui bisogna fare riferimento alla mappa disegnata sopra per sapere quale valore dare al parametro <cur> per posizionare correttamente il cursore nella posizione che desideriamo. Per esempio, per visualizzare questo testo nel mio display.

lcd_out.jpg

Ho dato questa serie di comandi di cui metto lo screenshot



screen_print_lcd.jpg

Esiste anche un comando che per mette d' inviare i comandi all' LCD così come sono scritti nel datasheet del HD44780 che permette a questo punto di gestire il controller senza alcuna limitazione. Il comando è:

LCD CMD <n>

dove <n> è un numero esadecimale di due cifre. Inviare il comando **LCD CMD 01** equivale ad inviare il comando **LCD CLR**, cosi' come il comando **LCD CMD 0D** equivale al comando **CURSOR ON**. Questo comando a basso livello permette quindi il controllo totale del modulo senza limitazioni. Così come esiste la possibilità d' inviare un comando a basso livello esiste anche un comando per l' invio grezzo di un dato, il comando è:

LCD DATA <n>

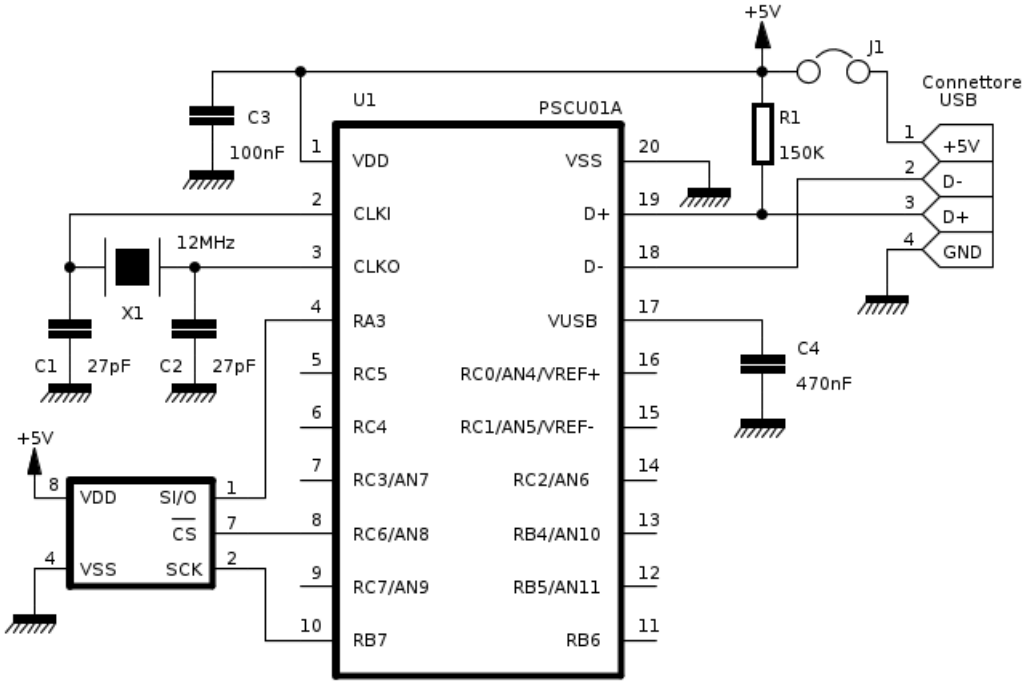
dove <n> è un numero esadecimale di due cifre. L' invio del comando **LCD DATA 31** equivale a scrivere **PRINT"1** in quanto il codice ASCII del numero '1' è 31 in esadecimale. In buona sostanza il comando PRINT non fa nient' altro che inviare i codici ASCII dei caratteri della stringa (come dato) uno dietro l' altro.

Il vantaggio di avere la possibilità d' inviare comandi e dati a basso livello permette di poter sfruttare la RAM interna (chiaramente solo in scrittura) per programmarci i propri caratteri speciali o, più semplicemente, di studiare e provare il modulo LCD (che magari si utilizzerà in altre applicazioni comandato ad esempio da un microcontrollore) direttamente, un comando dietro l' altro e vedere il risultato che si ottiene passo dopo passo.

Il sensore di temperatura TC77

E' un circuito integrato della Microchip per la misura della temperatura a basso costo e con buona risoluzione. La sua applicazione principale è quella di monitoraggio della temperatura di circuiti o dissipatori per evitare problemi di sovra riscaldamento. L'

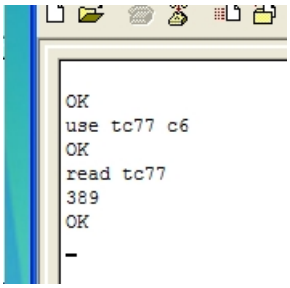
intervallo di temperatura che può misurare va da -55°C a +125°C e se si rimane nel range da +25°C a +65°C la precisione è di ±1°C. La risoluzione è di 13 bit e corrisponde a 0.065°C. Anche se la risoluzione è elevato non si può propriamente definire un campione di precisione ma comunque in rapporto al prezzo è un ottimo sensore. Le sue applicazioni tipiche sono il controllo della temperatura ambiente, protezioni termiche per hard-disk o moduli di potenza, controllo di ventole, sensore di temperatura per acquari, controlli industriali e per rimpiazzare i termistori in generale. Il collegamento con la SCU è realizzato tramite tre linee: una linea dati, un clock ed una linea di chip select. La SCU vuole la linea dati collegata al pin RA3, il clock al pin RB7 mentre il chip select si può collegare ad un pin a scelta. Questo integrato può coesistere con il display LCD. Il TC77 viene venduto in contenitore SMD SOIC quindi, per fare la prova, l' ho saldato su un adattatore in modo da poterlo filare sulla millefori. Il circuito di prova è questo:



Ho collegato il pin di CS al pin RC6 della SCU perché avevo già montato il display LCD e volevo provarli insieme. Per far sapere alla SCU che c'è un TC77 collegato e quale pin viene utilizzato per il CS si usa il comando

USE TC77 <porta><bit>

Quindi per farle sapere che il CS è collegato al pin RC6 bisogna inviare il comando **USE TC77 C6**. La lettura della temperatura avviene mediante il comando **READ TC77** e restituisce un valore espresso in unità da 0.0625°C. Lo screenshot del terminale illustra bene la procedura.

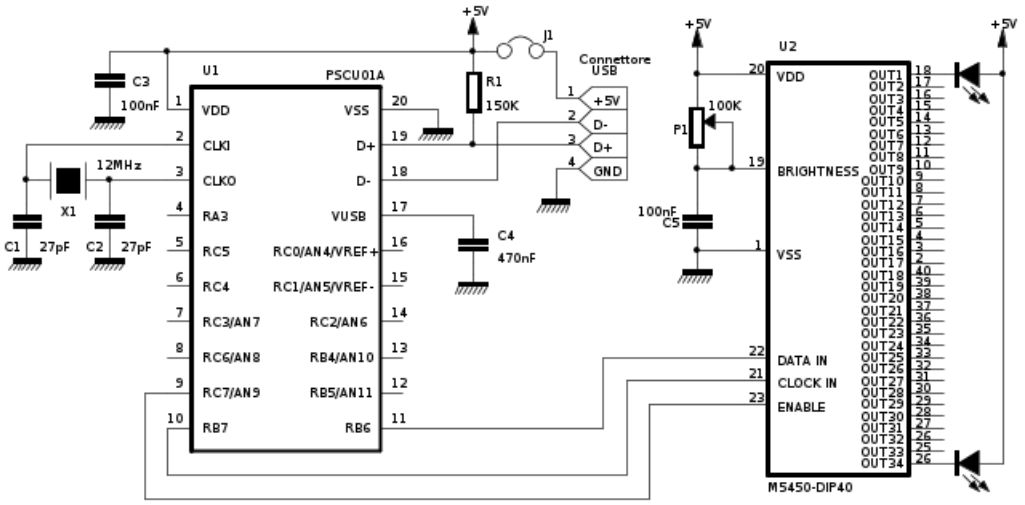


screen_tc77.jpg

Quindi per sapere la temperatura ambiente si va di calcolatrice: 354 x 0,0625 = 22,125°C.
Per curiosità sono andato a prendere la bomboletta del ghiaccio spray, quello per lenire le contusioni, che mi porto sempre dietro quando vado in montagna a divertirmi a fare salti e fesserie varie con gli scietti (snow blades). E' normale che ogni tanto prenda qualche botta per un salto sbagliato o perché mi lancio un pò troppo. Ho voluto misurare la temperatura a cui arriva ed ho letto un -300 e qualcosa. In buona sostanza la superficie della pelle arriva a 20 gradi sotto zero! D' ora in poi non lo chiamerò più "ghiaccio spray" ma "freezer portatile" eh eh eh.

Pilotare LED con il M5450

L' MM5450 della National o l' M5450 dell ST è un driver per LED a corrente costante in grado di pilotare 34 LED. Ha un pin deputato alla regolazione della corrente dei LED e si interfaccia tramite la Microwire che è, in buona sostanza, un collegamento SPI. Utilizza infatti un segnale di clock, una linea dati ed un chip select chiamato ENABLE. La particolarità di questo integrato è il pilotaggio a corrente costante. I LED sono collegati fra le uscite e la tensione di alimentazione (o altra tensione per limitare la dissipazione di potenza). Pilotare LED con corrente costante, identica per tutti, permette di evitare i problemi legati alla diversa caduta di tensione sui LED stessi: a parità di corrente avranno la stessa luminosità anche se provengono da lotti differenti con diverse cadute dirette di tensione. Al posto dei LED si possono anche pilotare degli optoisolatori utilizzati, per esempio, per pilotare lampade alimentate dalla tensione di rete. Volendo le uscite possono essere utilizzate anche per pilotare piccoli relè. Diciamo che le applicazioni sono molte e 34 uscite in più potrebbero far comodo. Come per il TC 77 la linea di clock va collegata al pin RB7 della SCU mentre la linea dati deve essere collegata al pin RB6. La linea dell' ENABLE è scelta dall' utente come con il TC77. Ho quindi realizzato una basetta millefori con un M5450 della ST, 34 LED, un trimmer per il controllo della corrente (e quindi della luminosita) ed ho collegato il piedino ENABLE al pin RC7 della SCU.



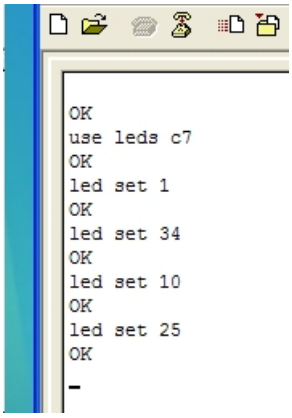
Nel circuito ho disegnato solo il primo e l' ultimo LED perché gli altri sono montati nello stesso modo. Nella mia basetta li ho indicati da 1 a 34. Per impostare la SCU per essere utilizzata insieme ad un M5450 si usa un comando simile a quello del sensore di temperatura e cioè:

USE LEDS <porta><bit>

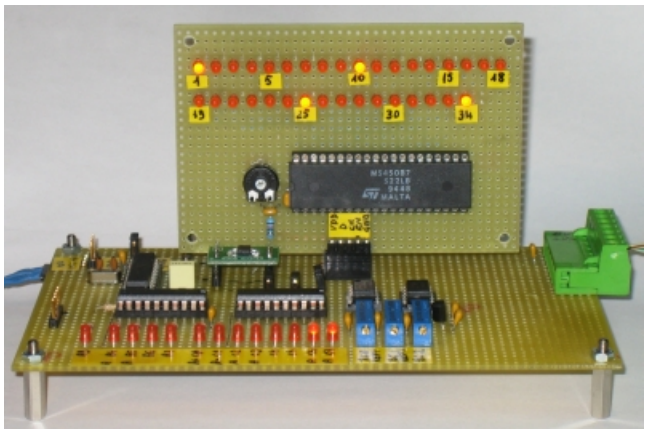
<porta> e <bit> sono per specificare a quale piedino viene collegato il segnale di ENABLE. Nel mio caso, essendo collegato a RC7 devo dare il comando **USE LEDS C7**. Da questo momento in poi si potranno dare i comandi per utilizzare il driver. I comando per accendere e spegnere i LED sono questi.<\br> **LED SET <n>**

LED CLR <n>

E' facile intuire che <n> rappresenta il numero dell' uscita come indicata nel datasheet. Qui di seguito ci sono lo screenshot dei comandi che ho inviato ed il risultato ottenuto.



screen_M5450.jpg



LED5450.jpg

Ci sono ancora due comandi per la gestione del M5450 e sono:
LEDS ON che accende contemporaneamente tutti i LED
LEDS OFF che li spegne tutti
Ed infine c'è l' ultimo camando, quello che permette di caricare in un sol colpo tutti l LED contemporaneamente. E' un comando che penso che sia più utile per effetti di luce piuttosto che per pilotaggi di relè la sintassi è:

LEDS LD <outputs>

Dove <outputs> è un numero esadecimale di 9 caratteri. La tabella qui sotto indica la corrispondenza dei leds con i vari digit esadecimali.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
MSb LSb DIGIT #1				MSb LSb DIGIT #2				MSb LSb DIGIT #3				MSb LSb DIGIT #4				MSb LSb DIGIT #5				MSb LSb DIGIT #6				MSb LSb DIGIT #7				MSb LSb DIGIT #8				b4 b3 DIGIT #9	

Quindi per accendere il LED 1 ed il LED 34 lasciando tutti gli altri spenti il comando da inviare è **LEDS LD 800000004**. E ancora, per accendere il LED 2, il LED 17, il LED 33 e lasciare tutti gli altri spenti il comando sarà **LEDS LD 400080008**. Con un programma che invia le configurazioni dei LEDS con le opportune tempistiche si possono così generare divertenti effetti di luce. Se poi le uscite dei LED sono collegate a MOC che pilotano dei triac ... it's party time!
Piccola nota: come si vede (forse) nella fotografia, sotto il trimmer alla sinistra del M5450 c'è un basettino piccolino di colore verde. Quello è il sensore di temperatura TC77. Dietro la basetta dei LED trova posto un connettore a 10 pin che utilizzo per collegare il modulo LCD che ho montato in una specie di consolle di legno. Il modulo LCD, il sensore di temperatura e il driver M5450 possono coesistere nello stesso circuito. Rimangono ancora disponibili 4 pin che possono essere I/O digitali o ingressi analogici.

I comandi accessori

Fino ad ora la SCU è stata utilizzata manualmente tramite un programma di terminale ma il suo scopo è quello di essere comandata da un programma che gira su un PC. Ci sono quindi una serie di comandi accessori che possono essere utili per lo scopo che sono:

RESET

Questo comando svolge la stessa funzione di togliere e ridare alimentazione, Dopo un comando di reset la SCU viene riportata alla condizione di partenza e cioè

- Tutti gli ingressi definiti come input digitali
- Tensione di riferimento positiva dell' ADC (VREF+) connessa a VDD
- Tensione di riferimento negativa dell' ADC (VREF-) connessa a VSS
- Eco dei caratteri disattivato
- Risposte ai comandi attivate
- Nessuna periferica collegata.

ID restituisce il modello di SCU, in questo caso restituisce la stringa "PSCU01A"
VER restituisce la versione del firmware, ad esempio "1.0"

ECHO <ON/OFF>

Questo comando serve per attivare o disattivare l' eco dei caratteri inviati. Quando si usa manualmente l' eco è importante perché permette di visualizzare il comando che si sta inviando e per correggere eventuali errori di battitura. Nel caso in cui la SCU è controllata da un programma che gira su di un PC l' eco è inutile e complica solo le cose. Il comando **ECHO ON** serve appunto per attivare l' eco dei caratteri. Anche con l' eco disabilitato la SCU risponde comunque con OK quando il comando è corretto ed è stato eseguito. Come accennato all' inizio dell' articolo la SCU all' avvio non restituisce l' eco dei caratteri quindi, per poterla utilizzare agevolmente in modo manuale il primo comando da inviare è **ECHO ON**

ANS <ON/OFF>

Serve per attivare o disattivare le risposte OK ed i messaggi di errore dei comandi inviati. I risultati delle funzioni GET vengono comunque inviati. Può essere utile nel caso in cui si ha la certezza di inviare comandi corretti ed evita quindi di attendere la risposta della SCU. Su questo è opportuno spendere qualche parola in più. La comunicazione tramite USB contiene, a livello di protocollo, diversi controlli di integrità dei messaggi realizzati con checksum ed ammennicoli vari. La connessione USB si può considerare una connessione sicura quindi, una volta verificato che il collegamento è ben fatto, si ha in pratica la sicurezza di avere una comunicazione priva di errori. Inoltre l' emulazione di linea seriale accetta caratteri quando è in grado di accettarli e quindi non c'è neanche il rischio che qualche comando non venga ricevuto. Per provarlo basta scrivere un file di testo con una serie di comando uno dietro l' altro (andando a capo alla fine di ogni comando), scaricarlo nella SCU tramite la funzione di trasferimento di file di testo e si vedrà che nessun comando

viene perduto. Tutti verranno eseguiti. Anche per questo motivo c'è il comando

DELAY <time>

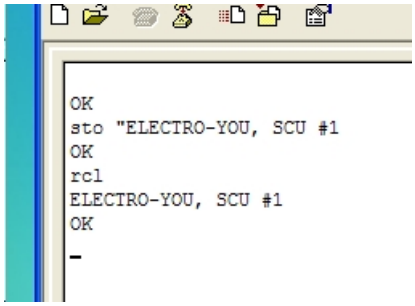
Dove <time> indica il numero di millisecondi che può andare da 1 a 65535 ms. Se bisogna comandare dei relè ed essere certi che il contatto si sia attivato prima di passare al prossimo comando è sufficiente inserire, ad esempio, il comando DELAY 100. La SCU si bloccherà per 100ms e solo dopo sarà di nuovo in grado di eseguire altri comandi. Si evita così di far eseguire il ritardo dal programma di gestione.

I comandi **STO** e **RCL**.

Nel caso di piu' SCU collegate all' USB sarebbe opportuno avere la possibilità di memorizzare una stringa nella memoria non volatile di ciascuna di esse, il che verrebbe utile, ad esempio, per identificarle in automatico. Per fare questo c'è un comando apposta che permette di memorizzare una stringa e, chiaramente, un comando per leggere questa stringa. Il comando di memorizzazione è il comando **STO** la cui sintassi è identica al comando **PRINT** utilizzato per la gestione del display.

STO "<testo>

<testo> è la stringa da memorizzare nella memoria non volatile. Per recuperare la stringa si usa il comando **RCL**. Ecco lo screenshot di come funzionano questi due comandi.



screen_sto_rcl.jpg

DA questo momento in poi la stringa rimarrà memorizzata anche quando viene tolta l' alimentazione e può essere modificata in qualsiasi momento con un nuovo comando **STO**.

Gestire la SCU con un programma su PC

Usare questo dispositivo con un programma di terminale ha senso solo per imparare ad utilizzarne i comandi e per verificarne le sequenze. Una sequenza di comandi può essere scritta in un file di testo ed inviata tramite il comando "Invia file di testo" dell' HyperTerminal. La sequenza scritta nel file verrebbe correttamente eseguita ma rimarrebbe comunque una sequenza e basta. Più interessante è utilizzare questo oggetto tramite un programma su PC con il quale fargli svolgere le funzioni di un controllore programmabile o altro. Inoltre il programma su PC sarebbe ingrado di leggere i valori degli ingressi o gli eventi di trigger ed utilizzarli per vari scopi.

La classe Java ComandoSCU

Per realizzare un programma di prova in Java ho scritto una classe chiamata ComandoSCU. Questa classe permette di effettuare tutte le operazioni necessarie per utilizzare la SCU e gestirne le comunicazioni e le risposte in modo trasparente. Qui di seguito illustrerò nel dettaglio la sua implementazione.

serialEvent

Questo metodo gestisce la ricezione dei caratteri dalla linea seriale. L' array di bytes **chunk** contiene tutti i caratteri presenti nel buffer di ricezione quindi devono essere acquisiti uno alla volta sopratutto per intercettare gli eventi trigger eventualmente inviati dalla SCU e, chiaramente, ricevere le risposte che questa invia. Dopo aver "scremato" gli eventi trigger gli altri caratteri vengono introdotti in una stringa chiamata **rxBuffer**.

```
/**
 * Metodo per la gestione dell' evento di ricezione dalla porta seriale
 * con monitoraggio dei messaggi di trigger
 * @param oEvent
 */
public synchronized void serialEvent(SerialPortEvent oEvent)
{
  int i,pt;
  String s,risp;

  if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE)
  {
    try
    {
      int available = input.available();
      byte chunk[] = new byte[available];
      input.read(chunk, 0, available);
      for (i=0;i<available;i++)
      {
        s = ""+(char)chunk[i];
        rxBuffer += s;
        if (rxBuffer.contains("*I0\r\n"))
        {
          pt = rxBuffer.indexOf("I0\r\n");
          risp = rxBuffer.substring(pt+4);
          rxBuffer = risp;
          trgI0 = true;
        }
        if (rxBuffer.contains("*I1\r\n"))
```

```
          {
            pt = rxBuffer.indexOf("I1\r\n");
            risp = rxBuffer.substring(pt+4);
            rxBuffer = risp;
            trgI1 = true;
          }
          if (rxBuffer.contains("*I2\r\n"))
          {
            pt = rxBuffer.indexOf("I2\r\n");
            risp = rxBuffer.substring(pt+4);
            rxBuffer = risp;
            trgI2 = true;
          }
        }
      }
      catch (Exception e)
      {
        System.err.println(e.toString());
      }
    }
  }
```

I tre flag di trigger si possono leggere tramite il metodo **getTrigger(int n)** dove n può valere 0, 1 o 2. Una volta riconosciuto l' avvenuta intercettazione dell' evento di trigger il flag corrispondente deve essere resettato utilizzando il metodo **clrTrigger(int n)**.

```
  /**
   * Legge il valore del flag di ricezione del messaggio di trigger
   * @param n numero dell' ingresso trigger (0/1/2)
   * @return true se ricevuto il trigger
   */
  public boolean getTrigger(int n)
  {
    boolean valore;

    valore = false;
    switch(n)
    {
      case 0: valore = trgI0; break;
      case 1: valore = trgI1; break;
      case 2: valore = trgI2; break;
    }
```

```
    return(valore);
  }

  /**
   * Cancella il flag di trigger
   * @param n numero del flag (0/1/2)
   */
  public void clrTrigger(int n)
  {
    switch(n)
    {
      case 0: trgI0 = false; break;
      case 1: trgI1 = false; break;
      case 2: trgI2 = false; break;
    }
  }
```

Questi sono metodi molto semplici che ho implementato per evitare di dare accesso diretto alle tre variabili.

portInit

Il metodo **portInit(String comPort)** serve per inizializzare ed aprire la porta di comunicazione indicata da comPort. Per esempio, per aprire la porta di comunicazione COM6 bisogna chiamare il metodo portInit("COM6"). La prima parte ricerca la porta di comunicazione il cui nome corrisponde a quella selezionata. Se non viene trovato il metodo interrompe l' esecuzione ritornando un valore false. La parte fra il **try** ed il **catch** è la parte che effettivamente inizializza ed apre la porta in ingresso ed in uscita.

```
  /**
   * Inizializzazione ed apertura della porta di comunicazione.
   * @param comPort stringa con il nome della porta da connettere
   * @return true se la connessione è andata a buon fine
   */
  public boolean portInit(String comPort)
  {
    CommPortIdentifier portId = null;
    Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();
    connesso = false;

    // continua a cilclare fino a quando trova la porta
    while (portEnum.hasMoreElements())
    {
      CommPortIdentifier currPortId = (CommPortIdentifier) portEnum.nextElement();
```

```
      if (currPortId.getName().equals(comPort))
      {
        portId = currPortId;
        System.out.println("Connesso a "+comPort);
        connesso = true;
        rxBuffer = "";
        break;
      }
    }

    if (portId == null)
    {
      System.out.println("Could not find COM port.");
      return(false);
    }

    try
    {
      // open serial port, and use class name for the appName.
      serialPort = (SerialPort) portId.open(this.getClass().getName(),TIME_OUT);

      // set port parameters
      serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_RTSCTS_IN | SerialPort.FLOWC
      serialPort.setSerialPortParams(DATA_RATE,SerialPort.DATABITS_8,SerialPort.STOPBIT

      // open the streams
      input = serialPort.getInputStream();
      output = serialPort.getOutputStream();

      // add event listeners
      serialPort.addEventListener((SerialPortEventListener) this);
      serialPort.notifyOnDataAvailable(true);
    }
    catch (Exception e)
    {
      System.err.println("Connessione: "+e.toString());
      connesso = false;
    }
    return(connesso);
  }
```

Per verificare se la SCU è connessa si usa il metodo **isConnected()** che restituisce un valore true se la SCU è connessa. Anche questo è un metodo semplicissimo che,

in pratica, resistuisce il valore della variabile boolean **connesso**.

sendCmd

Per inviare i comandi si usano il metodo **send(String s)** che semplicemente invia la stringa attraverso la linea seriale mentre il metodo **sendCmd(String s)** invia la stringa e si preoccupa di verificare che la risposta al comando sia OK, in tal caso ritorna il valore true.

```
/**
 * Invia una stringa alla SCU senza attendere nessuna risposta
 * @param s stringa da inviare
 */
public void send(String s)
{
  int i,l;
  char ch;

  l = s.length();
  for (i=0;i<l;i++)
  {
    try
    {
      ch = s.charAt(i);
      output.write(ch);
    }
    catch (IOException ex)
    {
      Logger.getLogger(ComandoSCU.class.getName()).log(Level.SEVERE, null, ex);
    }
  }
}

/**
 * Invia un comando alla SCU verificando che la risposta sia OK
 * @param s stringa da inviare alla SCU
 * @return true se ha ricevuto la risposta OK
 */
public boolean sendCmd(String s)
{
  boolean fine, errore;
  String risp;
  int pt;
  long ct;
```

```
    errore = false;
    send(s);
    fine = false;
    risp = "";
    ct = System.currentTimeMillis();
    while(!fine && ((System.currentTimeMillis() - ct) < 2000))
    {
      if (rxBuffer.contains("OK\r\n"))
      {
        clrRxBuffer();
        fine = true;
      }
    }
    if (!fine) errore = true;
    return(errore);
  }
```

La parte che culmina con **while(!fine && ((System.currentTimeMillis() - ct) < 2000))** è una porcheria che mi è costata anche un bel cazziatone dai guru del forum. In effetti implementa una timeout di 2 secondi per la ricezione della risposta. Funziona bene ma è scritta male, in modo rozzo (impegna totalmente la CPU) e non pulito come sarebbe giusto. Chiedo quindi umilmente venia ed per la mia pochezza e rozzezza nella programmazione in Java. Però mi metterò d' impegno per correggere questo obbrobrio informatico che, per dirla tutta, non è il solo presente nel software che ho scritto.

getAns()

Il metodo **getAns()** serve invece per ricevere una risposta dlla SCU. Si usa quando si invia un comando che deve ritornare un valore numerico, come ad esempio un comando di GET per leggere il valore di una tensione o lo stato di un pin di ingresso digitale. In tal caso prima si invia il comando con il metodo **send** (non sendCmd che attenderebbe la risposta OK) e dopo si attende la risposta con getAns.

```
  /**
   * Attende la risposta ad un comando di lettura comprendente l' OK o il messaggio di
   * ed elimina i caratteri <CR> e <LF>
   * @return stringa con la risposta seguta da OK o dal messaggio di errore
   */
  public String getAns()
  {
    boolean fine;
    String risp;
    int pt;
    long ct;
```

```
    fine = false;
    risp = "";
    ct = System.currentTimeMillis();
    while(!fine && ((System.currentTimeMillis() - ct) < 2000))
    {
      if (rxBuffer.contains("OK\r\n")||
          rxBuffer.contains("ES\r\n")||
          rxBuffer.contains("EP\r\n")||
          rxBuffer.contains("EF\r\n")||
          rxBuffer.contains("ED\r\n"))
      {
        risp = rxBuffer;
        risp = risp.replace("\r\n","");
        fine = true;
        clrRxBuffer();
      }
    }
    return(risp);
  }
```

Anche questo metodo dovrebbe essere scritto meglio. In effetti eliminino le risposte in modo brutale evitando d' intercettare errori di sintassi, porte e tutto il resto. Non è un gran male visto che i programmi non dovrebbero contenere questi grossolani errori ma sarebbe comunque meglio individuarli e dare in qualche modo un' indicazione di un errore ricevuto. La stringa di risposta non contiene i caratteri <CR> e <LF> perché sono stati eliminati, il che rende più confortevole (almeno per me lo è) l' analisi della risposta stessa.

reset()

Il metodo **reset()** invia prima un carattere di ritorno a capo per essere sicuri che il successivo comando di reset sia correttamente eseguito.

```
  /**
   * Resetta la SCU
   */
  public void reset()
  {
    sendCmd("\r");
    sendCmd("reset\r");
  }
```

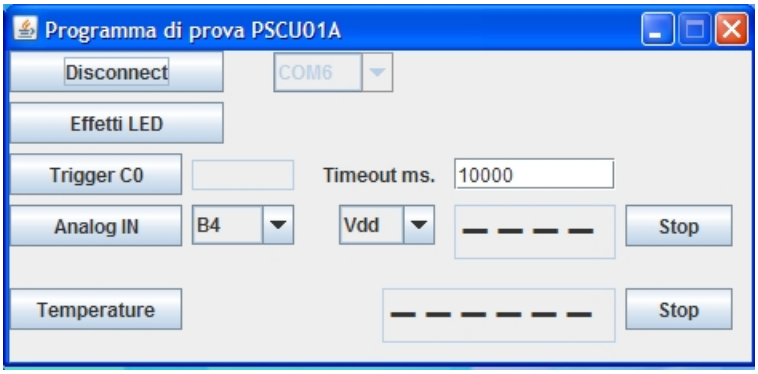
Come si può notare i comandi vengono inviati tramite il metodo sendCmd.
C'è anche un metodo chiamato **portList()** che ritorna una Enumeration contenente

le porte seriali installate nel computer. Io l' ho usato nel mio programma di prova per popolare il menù a tendina contenete la lista delle porte seriali disponibili.

```
/**
 * Questa serve per l' elenco delle porte
 * disponibili.
 */
public Enumeration portList()
{
  Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();
  return(portEnum);
}
```

Il programma ProvaSCU

Ho scritto questo programma per provare a gestire la SCU attraverso un programma scritto in Java. Il software e' contenuto nella cartella "ProvaSCU" ed è il progetto completo scritto con NetBeans. Per poterlo fare funzionare è necessario installare opportunamente il pacchetto RxTx. Si può anche utilizzarlo senza NetBeans andando nella cartella Pr_SCU e lanciando il programma ProvaSCU.jar. Una volta lanciato apparirà una finestra con una serie di pulsanti ed aree. L' unico pulsante abilitato è quello di connessione, infatti, prima di lanciare il programma, bisogna assicurarsi che la SCU sia collegata al PC. Nel menu a tendina sono elencate le porte seriali installate nel computer, si seleziona la linea seriale della SCU e quindi si preme il pulsante **Connect**. A quel punto gli altri pulsanti del programma si potranno utilizzare per le varie funzioni. Inserisco anche il codice di come ho implementato le varie prove perchè li ritengo interessanti nel senso che sono esempi reali di come si può comandare ed utilizzare la SCU. Per l' interfaccia grafica ho utilizzato la utility di disegno della GUI di NetBeans che è molto comoda e permette di ottenere velocemente ottime risultati.



screen_provascu.jpg

Il pulsante Connect è diventato Disconnect. La sua funzione e quella d' interrompere la connessione seriale ma quello che interessa maggiormente sono gli altri pulsanti.

Effetti LED

Effettua due cicli accendendo uno ad uno tutti i LED collegati al M5450. Il programma non permette di scegliere il piedino di ENABLE perchè ho utilizzato la mia basetta dove l' ENABLE è collegato a RC7 ed il CS del sensore di temperatura al pin RC6. Il programma è stato scritto per questo hardware.

```
/**
  * Effettua per due volte la scansione dei leds accendedoli
  * tutti e spegnendoli tutti
  */
 private void effettiLeds()
 {
   boolean errore;
   Integer i;
   int cicli;

   errore = false;
   if (scu.isConnected())
   {
     scu.reset();
     scu.clrRxBuffer();
     errore = scu.sendCmd("usetc77c6\ruseledsc7\r");
     for(cicli=0;cicli<2;cicli++)
     {
       for(i=1;i<35;i++)
       {
         scu.sendCmd("ledset"+i.toString()+"\r");
       }
       for(i=1;i<35;i++)
       {
         scu.sendCmd("ledclr"+i.toString()+"\r");
       }

     }
   }
   else System.out.println("effeetiLeds - non connesso");
 }
```

Da notare la riga **errore = scu.sendCmd("usetc77c6\ruseledsc7\r");** dove istruisco la SCU sui segnali di chip select del M5450 e anche del TC77. Questo perché sto utilizzando il circuito di prova ed ho montato l' hardware in questo modo.

Trigger C0

Questa funzione permette di misurare il tempo in ms. che intercorre fra un fronte di salita sul pin RC0 ed il successivo fronte di salita. A fianco del pulsante c' è un' area di testo dove verrà visualizzato il tempo trascorso fra i due eventi.

```
/**
 * Legge il tempo trascorso fra due eventi sul pin RC0
 * impostato sul fronte di salita
 */
private void leggiTempoC0()
{
  Long st,tempo,stTimeout,tmax;
  boolean abort;

  if(scu.isConnected())
  {
    tmax = Long.valueOf(txtTimoutTrig.getText());
    scu.reset();
    scu.sendCmd("pinc0trg+\r");
    scu.clrTrigger(0);
    abort = false;
    stTimeout = System.currentTimeMillis();
    while(!scu.getTrigger(0)&& !abort)
    {
      if ((System.currentTimeMillis()-stTimeout)> tmax) abort=true;
    }
    scu.clrTrigger(0);
    st = System.currentTimeMillis();
    //while(!scu.getTrigger(0));
    stTimeout = System.currentTimeMillis();
    while(!scu.getTrigger(0)&& !abort)
    {
      if ((System.currentTimeMillis()-stTimeout)> tmax) abort=true;
    }
    scu.clrTrigger(0);
    tempo = System.currentTimeMillis()-st;
    if (!abort) txtTempoC0.setText(tempo.toString()); else txtTempoC0.setText("Time
  }
}
```

Ecco, questo codice funziona ma e' la seconda procheria di cui parlavo prima. Anche qui ho utilizzato un metodo rozzo per la timeout che di fatto blocca la CPU fino a quando non sono stati ricevuti i due trigger dall' ingresso RC0 o allo scadere della

timeout. Pero' il tempo lo legge.

Analog IN

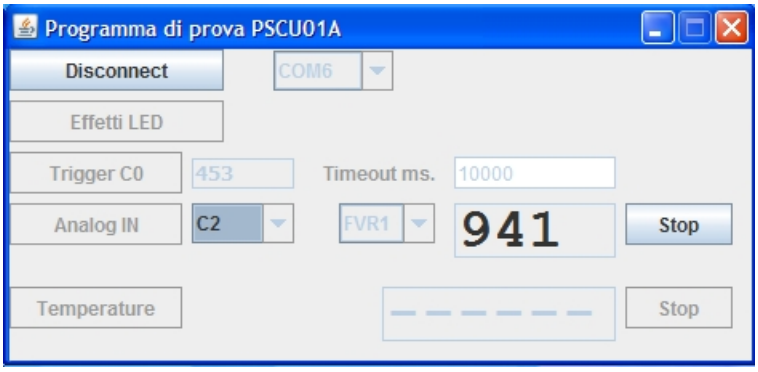
Questa funzione permette di monitorare un ingresso analogico. A fianco del pulsante si trova un menù a tendina dal quale selezionare l' ingresso da monitorare. A destra si trova un altro menù a tendina che serve per selezionare la tensione di riferimento positiva. L' acquisizione del valore viene effettuata ogni 100ms. Nell' immagine ho selezionato l' ingresso RC2 e la tensione di riferimento a 1.024V in modo da effettuare una misurazione assoluta. Il risultato mi indica il valore della tensione espresso in unità cioè, in questo caso, in mV. Per implementarla ho utilizzato un timer che genera un evento ogni 100ms.

```
private void monitorAnIn()
{
  String vref,pin;

  if(scu.isConnected())
  {
    // disabilita i controlli che non servono
    cmdLED.setEnabled(false);
    cmdTrigC0.setEnabled(false);
    txtTempoC0.setEnabled(false);
    jLabel1.setEnabled(false);
    txtTimoutTrig.setEnabled(false);
    cmdAnIn.setEnabled(false);
    cmbAnIn.setEnabled(false);
    cmbVrefP.setEnabled(false);
    cmdTemp.setEnabled(false);
    txtTemp.setEnabled(false);
    cmdStopTemp.setEnabled(false);

    baseTempiAn = new Timer(100, this);
    vref = cmbVrefP.getSelectedItem().toString();
    pin = cmbAnIn.getSelectedItem().toString();
    scu.reset();
    scu.sendCmd("pin"+pin+"an\r");
    scu.sendCmd("vref+"+vref+"\r");
    baseTempiAn.start();
  }
}
```

La lettura del valore avviene nel metodo che gestisce l' evento timer e nel quale c' è anche la lettura del sensore di temperatura. Questa è la parte di lettura:

```
if(e.getSource()==baseTempiAn)
{
  scu.send("get"+cmbAnIn.getSelectedItem().toString()+"\r");
  // toglie l' OK o il messaggio di errore
  risp = scu.getAns();
  pos = risp.indexOf("OK",0);
  if (pos != -1) risp = risp.substring(0,pos);
  // questi di seguito non servono ma li ho usati per intercettare
  // eventuali errori fatti prima.
  pos = risp.indexOf("ES",0);
  if (pos != -1) risp = risp.substring(0,pos);
  pos = risp.indexOf("EP",0);
  if (pos != -1) risp = risp.substring(0,pos);
  pos = risp.indexOf("EF",0);
  if (pos != -1) risp = risp.substring(0,pos);
  pos = risp.indexOf("ED",0);
  if (pos != -1) risp = risp.substring(0,pos);
  txtAnVal.setText(risp);
}
```

E questo è quello che compare nella finestra del programma quando si monitorizza l' ingresso.



screen_provaAn.jpg

Premendo il pulsante di Stop a destra del valore visualizzato s' interrompe il monitoraggio.

Temperature

Con questo pulsante si avvia il monitoraggio della temperatura attraverso il sensore TC77 ogni 500ms. Come ho accennato prima il programma suppone che il segnale di CS del TC77 sia collegato al pin RC6. Come per l' esempio precedente viene utilizzato un timer che qui viene fatto partire, insieme all' inizializzazione della SCU per l' uso del sensore di temperatura.

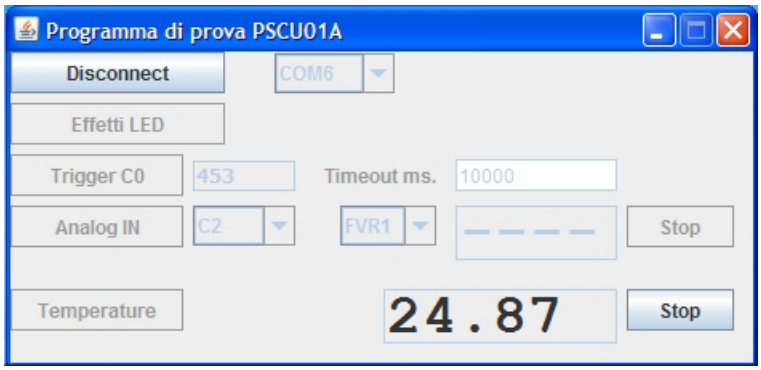
```
private void monitorTemp()
{
  cmdLED.setEnabled(false);
  cmdTrigC0.setEnabled(false);
  txtTempoC0.setEnabled(false);
  jLabel1.setEnabled(false);
  txtTimoutTrig.setEnabled(false);
  cmdAnIn.setEnabled(false);
  cmbAnIn.setEnabled(false);
  cmbVrefP.setEnabled(false);
  txtAnVal.setEnabled(false);
  cmdStopAnIn.setEnabled(false);
  cmdTemp.setEnabled(false);
  //txtTemp.setEnabled(false);
  //cmdStopTemp.setEnabled(false);

  baseTempiTemp = new Timer(500, this);
  scu.reset();
  scu.sendCmd("usetc77c6\r");
  baseTempiTemp.start();
}
```

Il programma effettua la conversione del valore in unità ottenuto dalla lettura dalla SCU in °C moltiplicandolo per 0,0625 e tendo solo le due cifre dopo la virgola.

```
  if(e.getSource()==baseTempiTemp)
  {
    scu.send("readtc77\r");
    // toglie l' OK o il messaggio di errore
    risp = scu.getAns();
    System.out.println("Prima rispos:"+risp);
    pos = risp.indexOf("OK",0);
    if (pos != -1) risp = risp.substring(0,pos);
    // questi di seguito non servono ma li ho usati per intercettare
    // eventuali errori fatti prima.
    pos = risp.indexOf("ES",0);
    if (pos != -1) risp = risp.substring(0,pos);
    pos = risp.indexOf("EP",0);
    if (pos != -1) risp = risp.substring(0,pos);
    pos = risp.indexOf("EF",0);
    if (pos != -1) risp = risp.substring(0,pos);
    pos = risp.indexOf("ED",0);
    if (pos != -1) risp = risp.substring(0,pos);
```

```
        System.out.println("Valore:" +risp);
        temp = Float.valueOf(risp.trim()).floatValue();
        temp *= 0.0625;
        s = String.valueOf(temp)+"00";
        // lascia solo due cifre decimali
        pos = s.indexOf(".");
        if (pos != -1)
        {
          risp = s.substring(0,pos+3);
        }
        txtTemp.setText(risp);
      }
```

E questo è l' output della prova.



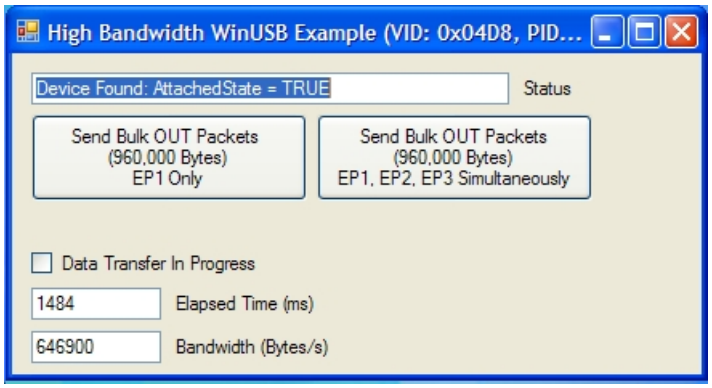
screen_provaTC77.jpg

Anche in questo caso il pulsante Stop a destra del display della temperatura interrompe il monitoraggio.
Il programma ... diciamo ... è scritto un pò così, "alla carlona" come si dice dalle mie parti poiché io non scrivo programmi per personal computers, e comunque **se sono riuscito a scrivere qualcosa di funzionante lo devo sopratutto ai preziosi consigli ed agli utenti di buona volontà che scrivono nel forum di ElectroYou, e che mi hanno insegnato, consigliato e seguito con pazienza in questa che per me è un' impresa. A loro va il mio ringraziamento sincero.**

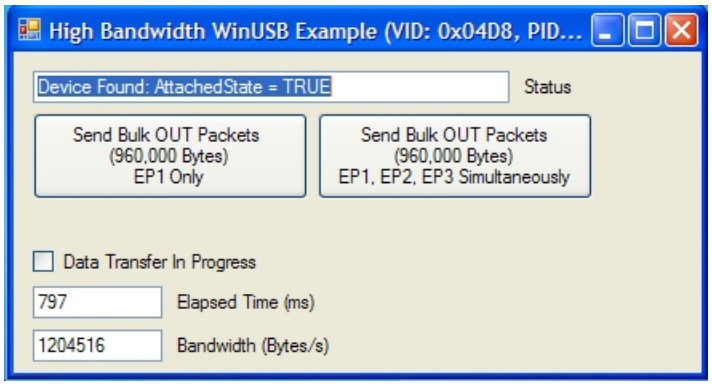
Problemi riscontrati

Per quanto riguarda altri linguaggi di programmazione ho anche provato a gestire le linee seriali con Visual Basic 6 utilizzando un esempio di uso del controllo MScomm. Il programma funziona ma non lo posso includere in questo articolo perché è protetto da copyright. Si può comunque trovare nella cartella MSDN che viene creata con l' installazione di Visual Studio 6. Con Visual Basic 2005 è possibile ma non lo conosco e non ne conosco le prestazioni. Con Java non ho avuto problemi. Diciamo

che Java mi ha soddisfatto parecchio (sopratutto in termini di affidabilità) anche se ho notato che la velocità di comunicazione è molto bassa. Ho fatto anche delle prove utilizzando la libreria della Microchip sia con i PIC18 che con il PIC32 ma la velocità resta comunque molto bassa. Di sicuro non dipende dal microcontrollore poiché ho avuto modo di sperimentare la spaventosa velocità del PIC32 e quindi penso che sia un problema di linguaggio e comunque non dipende dall' implementazione USB sui microcontrollori. Per essere sicuro di questo ho sperimentato un firmware che, come l' implementazione della classe CDC su cui si basa la SCU, utilizza il trasferimento "bulk" ad alta velocità e l' ho provato sul PIC18F14K50. I risultati che ho ottenuto sono questi, di cui inserisco lo gli screenshots. Il primo utilizza un solo endpoint, mente il secondo utilizza 3 endpoints contemporaneamente. Due immagini valgono più di mille parole parole.



screen_banda1.jpg



screen_banda2.jpg

Ad occhio e croce direi che con Java e RxTx si viaggia intorno ai 10Kbs o poco più, comunque più che sufficienti per realizzare apparecchiature per comandare relè, realizzare qualche simpatico effetto di luce e tutto il resto o realizzare collaudi automatizzati. Può darsi che con altri linguaggi di programmazione, oppure utilizzando le DLL per l' accesso diretto alle porte seriali o ai trasferimenti tramite USB, le prestazioni possano anche risultare nettamente superiori. Ma tutto questo

questa è materia che deve essere trattata dagli esperti di programmazione su PC, categoria della quale, ahimè, non faccio parte.
Mi auguro che qualche esperto pubblichi qualche interessante articolo sulla gestione delle linee seriali, magari con esempi utilizzanti diversi linguaggi di programmazione. La possibilità di sfruttare al massimo e con facilità la velocità dell' USB 2.0 potrebbe aprire le porta ad applicazioni gustose e performanti.

Note

Documentazione, downloads e contatti

Contatto

Per informazioni rigurdanti l' integrato PSCU01A visitare il **sito web della SANGON** oppure scrivere una e-mail a **questo indirizzo**
Lo si può comprare nel negozio online a **questo indirizzo**

Download

Il file **usb_scu.zip** contiene:

- Una cartella denominata **PSCUA01** che contiene la cartella **inf** con il driver USB per Windows ed il **manuale**.
- Una cartella denominata **Pr_SCU** che contiene il file .jar eseguibile del programma di prova
- La cartella **Prova_SCU** contenente il progetto per NEtBEans del programma di prova.

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Tardofreak:usb-switch-control-unit"