Giovanni Schgör (g.schgor)

PROVE APPLICATIVE DELLO SCU

4 June 2011

Già dall'apparire delle prime interfacce con la porta **USB** dei personal computer, sono convinto della potenzialità di questa soluzione che permette il colloquio del PC con il mondo fisico esterno.
Proprio in questa linea avevo illlustrato qualche anno fa in un minicorso le possibilità di utilizzo del dispositivo **USB-6008** della National Instruments, auspicando che anche in Italia venisse prodotto qualche dispositivo del genere (in particolare a costi competitivi per favorirne la diffusione anche in applicazioni hobbistiche).
Ho quindi accolto con estremo favore la segnalazione relativa allo sviluppo dello **SCU (Switch Control Unit)**, ampliamente illustrato in recenti articoli [1][2][3].

Caratteristiche dello SCU

Rimandando agli articoli citati per i dettagli, riassumo le principali caratteristiche dello SCU:
microcontrollore PIC18F14K50 a 20 piedini, preprogrammato per rilevare ingressi sia analogici che digitali e pilotare uscite digitali, in collegamento con la porta USB di un personal computer ed alimentato direttamente da questa.

Dei 20 piedini, 13 sono riservati per ingressi ed uscite, con la possibilità per alcuni di definire a programma la funzione (ingresso/uscita e digitale/analogico), in modo da consentire una notevole variabilità di configurazione.

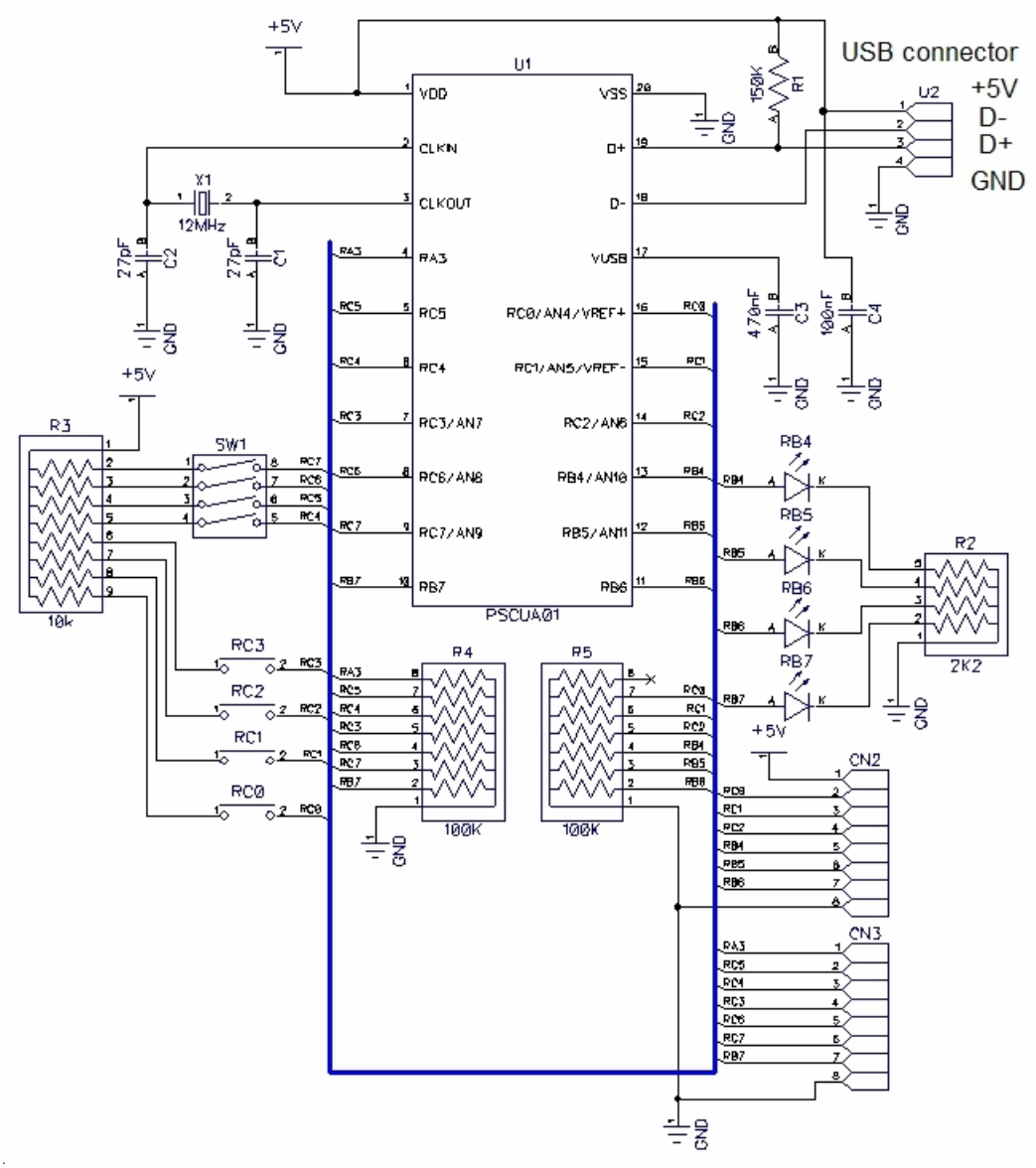
Il linguaggio di programmazione scelto per la gestione del modulo è Java, in particolare con l'utilizzo di NetBeans (ed..9.6.1).che ne facilita molto lo sviluppo. Personalmente non sono un entusiasta di questo linguaggio (lascia per esempio perplessi l'astrusità della sintassi, mutuata da C++, soprattutto nelle funzioni logiche) , ma devo riconoscere che la sua attuale diffusione è in notevole crescita e che se la programmazione è limitata alle funzioni di controllo (come vedremo più avanti) le difficoltà sono minime.

Fra i diversi possibili schemi di utilizzo (alcuni dei quali descritti nella documentazione citata) vorrei sottolineare l'emulazione di funzioni di PLC. E' questo una delle applicazioni didatticamente più interessanti in quanto introduce nella conversione da circuiti "cablati" a "programmati" (propedeutico quindi per l'utilizzo dei microcontrollori). Con la disponibilità di un'apposita scheda (attualmente in

preparazione) dotata di interruttori e LED per simulare rispettivamente ingressi ed uscite, è infatti semplice realizzare alternative di controllo. A questo scopo è stato sviluppato un programma di base (Base_SCU_EVB) che incorpora tutte le funzioni di colloquio con il PC, e che può essere personalizzato aggiungendo solo la parte di controllo. Esempi di tipici programmi applicativi (trasferibili con un semplice Copia/ Incolla) verranno illustrati in questa serie di articoli per facilitarne ulteriormente l'utilizzo.

Scheda "didattica"

Per facilitare l'approccio all'utilizzazione dello SCU è stata infatti sviluppata un'apposita scheda che consente sia di svolgere una serie di esercitazioni preprogrammate, sia verifiche su prototipi di schede dedicate ad impieghi particolari.
La configurazione scelta è di 4 uscite digitali (in particolare il comando di 4 LED) e di 8 ingressi digitali, di cui 4 previsti con pulsante e 4 con interruttori (DIP switches).
Viene ovviamente lasciata la possibilità di portare all'esterno (su connettori) tutti i segnali onde consentirne un'eventuale diversa utilizzazione.
Si sottolinea però che per un impiego di segnali esterni, sarebbe bene ricorrere all'interposizione di optoisolatori in modo da isolare la scheda alimentata, come già detto, direttamente dai 5V della porta USB.
Una visione generale della scheda è questa:



Come si può vedere, i LED corrispondono alle uscite RB4...RB7, i 4 pulsanti agli ingressi RC0...RC3 e i 4 interruttori a RC4...RC7. La chiusura di ciascun contatto porta l'alimentazione (+5V dalla porta USB) sui piedini del microprocessore (quindi un "1" logico al rispettivo ingresso).

La scheda può essere fornita in kit di montaggio con tutti i componenti necessari ed è quindi facilmente assemblabile.

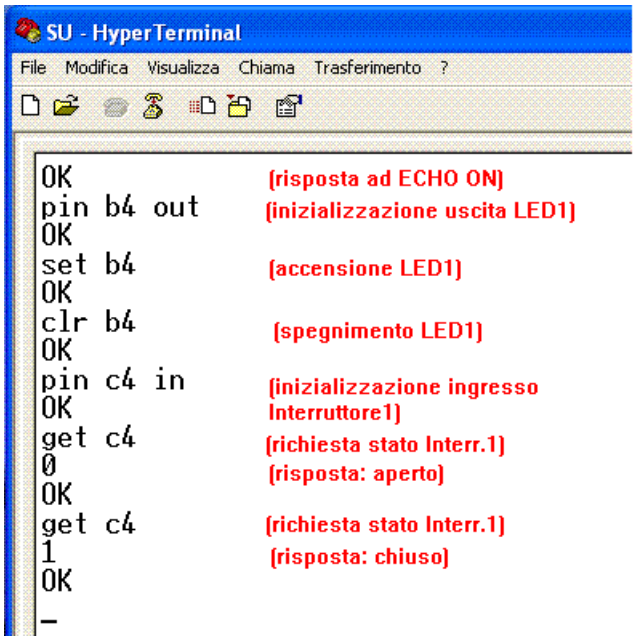
Preparazione dell'ambiente JAVA

Una volta montata la scheda, si pone il problema di creare l'ambiente di software necessario al suo utilizzo.
Si è già detto che come linguaggio di programmazione è stato scelto **JAVA**, quindi

bisogna che nel PC a cui la scheda deve essere collegata siano caricati i programmi di sviluppo ed esecuzione di questo linguaggio. Le modalità di caricamento sono dettagliatamente descritte nel già citato articolo [2] (paragrafo "Installazione"). In particolare vanno caricati il **JDK** (Java Development Kit), il **JRE** (Java RunTime Environment) ed il programma di sviluppo **NetBeans**.
Dopo aver caricato questi programmi bisogna procedere all'inserimento di quello di gestione delle linee seriali **RxTx**, come dettagliatamnte descritto nello stesso articolo (paragrafo "Il pacchetto RxTx").

Si è così pronti ad effettuarev l'inizializzazione della scheda stessa, seguendo le istruzioni date nel Manuale utente **PSCU01A**, scaricabile dal file usb-scu.zip (vedi [3] paragrafo "Download").
Questo permette il "controllo manuale" della scheda utilizzando ad esempio l' Hyper Terminal.
(Per accedere a questo dal desktop di Windows scegliere: Start/Tutti_i_programmi/Accessori/Comunicazioni/Hyper Terminal. Vedi comunque il manuale PSCU01A, paragrafo: "utilizzo del modulo Hyper Terminal").

Dopo aver digitato **ECHO ON** (e tasto "Invio") ed avere ricevuto **OK** come risposta, è infatti possibile comandare le uscite (accensione e spegnimento dei LED) semplicemente digitando gli appositi comandi.
Come già ampiamente illustrato in [3], bisogna prima "inizializzare" le uscite e poi dare il comando di accensione e di spegnimento, seguiti dalla sigla dell'uscita.
Un esempio consigliato della produra è questo:



Il primo LED (LED1) corrisponde infatti al piedno RB4 (come si può vedere dallo schema) programmabile come uscita mediante l'istruzione "pin b4 out", dopodichè possono seguire le relative istruzioni di accensione (set) e spenimento (clr).
L'esempio mostra anche il caso di acquisizione dello stato di un ingresso (l'interruttore entrante in RC4), dopo averlo inizializzato come ingresso ("pin c4 in")
Nell'esempio la prima richiesta è fatta con interruttore aperto, la seconda con interruttore chiuso.

Abbiamo così raggiunto già un importante risultato: il comando della scheda mediante la tastiera del PC. Ma non è certamente questa l'applicazione più importante. L'obbiettivo principale è infatti è quello di "programmare" il funzionamento della scheda in modo che questa segua "automaticamente" un determinato funzionamento, ed è per questo che abbiamo preparato appunto l'ambiente JAVA.

Il programma Base_SCU_EVB

Per facilitare la programmazione della scheda è di estrema importanza disporre di un pacchetto software che svolga tutte le funzioni di base necessarie al controllo, tra le principali la gestione della comunicazione fra PC e scheda (tramite porta USB)e l'interpretazione dei relativi messaggi.
Tale programma deve essere quindi "trasparente" per chi deve occuparsi solo delle funzioni vere e proprie che la scheda deve compiere, limitando al minimo lo sviluppo del software di questa parte.
Il pacchetto **Base_SCU_EVB** risponde a queste esigenze permettendo, come vedremo, almeno nei casi più semplici la scrittura di poche righe di programma "esecutivo".

A questo punto è indispensabile ricorrere all'uso di NetBeans (citato all'inizio) che permette di accedere al programma di base e di modificarlo con l'inserzione del programma esecutivo.

Va sottolineato che nel programma di base è compresa l'inizializzazione degli ingressi ed uscite della scheda, che vale la pena di riportare qui per chiarire le funzioni e le sigle di ciascuno:

```
// Inizializzazione Ingressi/Uscite
// Pulsanti
  scu.sendCmd("pin c0 in\r");
  scu.sendCmd("pin c1 in\r");
  scu.sendCmd("pin c2 in\r");
  scu.sendCmd("pin c3 in\r");
// Interruttori
  scu.sendCmd("pin c4 in\r");
  scu.sendCmd("pin c5 in\r");
  scu.sendCmd("pin c6 in\r");
  scu.sendCmd("pin c7 in\r");
// Uscite (LED)
  scu.sendCmd("pin b4 out\r");
  scu.sendCmd("pin b5 out\r");
  scu.sendCmd("pin b6 out\r");
  scu.sendCmd("pin b7 out\r");
```

Questo permette infatti di assegnare la funzione di ogni ingresso od uscita, inviando messaggi del tipo già visto nell'uso di Hyper Terminal.
Siamo così ora in grado di passare ai programmi esecutivi.

Primi esempi applicativi

Rimandando a più avanti l'esame sul programma di base, si vuole qui mostrare subito qualche esempio di programma applicativo da "incollare" in quello di base:

```
// Loop
   for (int i=0;i<4;i++) {
   if (i==0)scu.sendCmd("set b4\r");
   if (i==1)scu.sendCmd("set b5\r");
   if (i==2)scu.sendCmd("set b6\r");
   if (i==3)scu.sendCmd("set b7\r");
   // aspetta 1s
   scu.sendCmd("DELAY 1000\r");
    }
// Fine Loop
```

Come si può vedere, il programma in Java delimitato fra "Loop" e "Fine Loop" richiama un'istruzione **for** per valori della variabile (intera) **i** da 0 a 3, e fa corrispondere ad ogni valore di i l'accensione progressiva dei 4 LED della scheda, con un intervallo di 1 secondo (istruzione "DELAY 1000") fra le accensioni.
Il programma applicativo va "copiato" dal file sorgente ed "incollato" in quello di base, in corrispondenza delle diciture Loop e Fine Loop (cancellando prima quello eventualmente già presente).

Per l'attivazione dell'intero programma (base + applicativo) da NetBeans occorre puntare su Run/RunFile con sui appare la finestra di abilitazione alla connessione (riquadro "Connect", poi riquadro "Start Processo")

Dopo l'ultimo Start dovrebbe partire l'esecuzione del programma, cioè l'accensione sequenziale dei LED della scheda.
Alla fine del programma i LED rimangono accesi e per ricominciare bisogna prima interrompere l'esecuzione del programma base (con "Stop Processo") e poi riavviarlo (con "Start Processo").

Proviamo ora con un altro programma applicativo, questo:

```
// Loop
   boolean p=true;
   for (int i=60;i>0;i--) {
   if (p) p=false;
   else p=true;
   if (p) scu.sendCmd("set b4\r");
   else
         scu.sendCmd("clr b4\r");
   // aspetta 500ms
   scu.sendCmd("DELAY 500\r");
     }
  // Fine Loop
```

Anche qui viene usata l'istruzione **for** per stabilire la durata del ciclo (60 cicli di 500ms, quindi 30s) in cui viene alternativamente acceso o spento il LED1 (RB4). E' in definitiva un programma di lampeggio a frequenza 1Hz come potrebbe essere ottenuta da un 555, ed è evidente la possibilità di variare a piacere tale frequenza variando semplicemente il dato dei millisecondi dell'attesa (DELAY).
Ma qui abbiamo una sorpresa: dopo i 30s il lampeggio non si ferma ma continua all'infinito, e se si tenta di arrestarlo (con "Stop Programma") vediamo che l'arresto (con LED1 acceso) avviene solo quando termina il Loop.
Per spiegare questo comportamento dobbiamo quindi capire meglio le particolarità del programma di base.

Particolarità del programma Base_SCU_EVB

E’ stato detto che il programma base deve essere trasparente a chi programma la applicazioni, e questo è vero per la parte che riguardala comunicazione, l’avvio e

l'arresto,ma per quanto si è visto in precedenza, occorre capire almeno la struttura per usufruire delle speciali caratteristiche del programma stesso.
Tralasciando quindi la prima parte, si evidenzia la routine **userProgram** (vedi riga: private void userProgram()), che nel suo interno comprende l'inizializzazione (già citata), la gestione dei timers e la routine interna **userProcess** (vedi riga: private void userProcess()). Quest'ultima contiene infine il programma di controllo, che abbiamo visto delimitato da Loop e Fine Loop.

In particolare l'attivazione nell'userProgram di un timer che fa da base dei tempi (vedi righe con base TempiTimers) lancia l'userProcess ogni100ms, ed è questo che permette allo SCU di emulare un PLC.
Il programma di controllo viene infatti eseguito ogni decimo di secondo (a meno ché la sua esecuzione non duri più a lungo, nel qual caso il lancio viene ritardato finché questa è terminata).
Ed ecco allora come ciò può essere utilizzato nella programmazione dell'accensione ad intermittenza del LED1 (a confronto di quello visto in precedenza):

```
// Loop
      if  (timers[0]==0)
      {
      timers[0]=5;
      LED1 = ! LED1;
      }
      if (LED1) scu.sendCmd("set b4\r"); else scu.sendCmd("clr b4\r");
 // Fine Loop
```

Uno dei timers (dei 99 previsti), indicato con **[0]**, viene "caricato" a 5 (x100ms della base Tempi) quindi per un tempo do 0.5s, e viene invertito la stato della variabile (booleana) LD1 quando il suo contenuto risulta =0.
La relativa uscita (RB4) viene poi attivata o disattivata in base allo stato di LED1, accendendo o spegnendo alternativamente il corrispondente LED.
Come si è detto, l'intero loop viene eseguito ogni decimo di secondo e l'esecuzione può essere fermata in qualsiasi momento con Stop Process.
Per contro però la dichiarazione delle variabili e la loro eventuale inizializzazione non può più essere fatta all'inizio del programma applicativo, ma devono essere poste rispettivamente prima dell'userProgram (vedi: Dichiarazione delle variabili) e prima dell'userProcess (vedi: Inizializzazione).

Ritengo per il momento sufficiente come primo approccio quanto detto, rimandando a prossimi articoli la programmazione della logica per l'emulazione di PLC e di piccole automazioni.