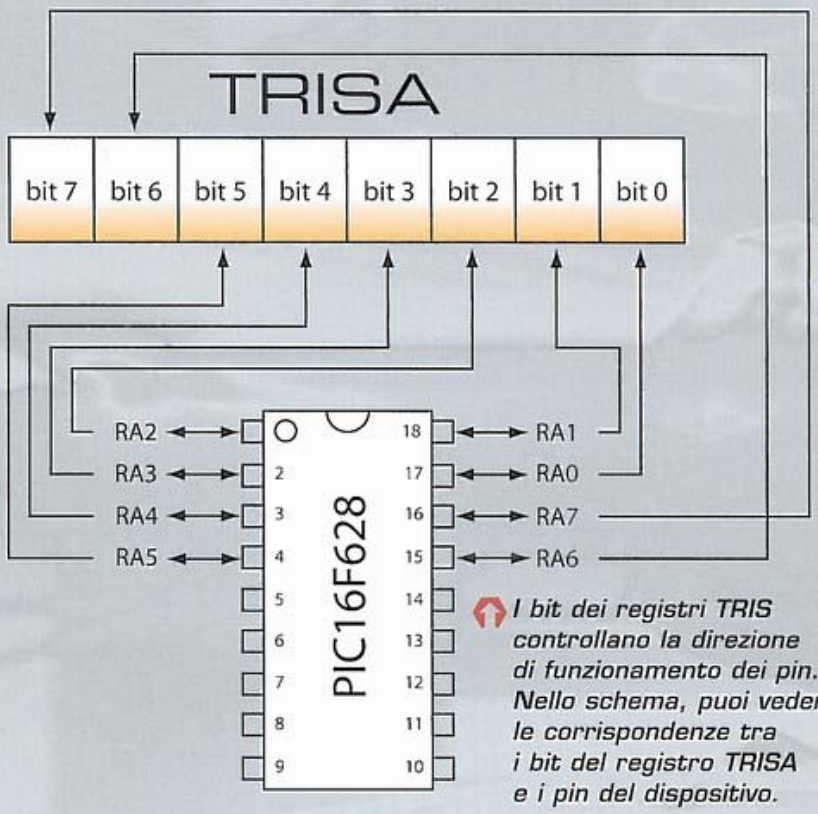
UN SEMPLICE FIRMWARE PER IL 16F628

Dopo aver presentato il microcontrollore 16F628 iniziamo a scrivere il nostro primo firmware con l'ausilio del linguaggio C dell'ambiente di sviluppo mikroC di mikroElektronika.



I bit dei registri TRIS controllano la direzione di funzionamento dei pin. Nello schema, puoi vedere le corrispondenze tra i bit del registro TRISA e i pin del dispositivo.

Nel fascicolo 56 abbiamo introdotto il PIC 16F628 e presentato alcune delle sue caratteristiche peculiari a livello di hardware interno. Facciamo ora un passo avanti e cominciamo ad affrontare i primi passi per la sua programmazione. Le pagine di Workshop dedicate al linguaggio C ti hanno permesso di acquisire le nozioni più elementari riguardanti lo sviluppo di software per i comuni personal computer. La creazione dei firmware per i microcontrollori non è, in sé, molto differente; tuttavia è necessario **conoscere a fondo la piattaforma hardware per cui si sta programmando e i 'meccanismi' logici alla base del suo funzionamento.**

UTILIZZARE L'OUTPUT LOGICO IN MIKROC

Introducendo il 16F628 abbiamo visto come alcuni dei suoi pin siano raggruppati in blocchi con funzionamento di tipo logico chiamati **'porte'**. Abbiamo anche detto che **ogni pin può operare sia in modalità di input, sia in modalità di output digitale**, utilizzando i livelli elettrici di riferimento. Ma in che modo è possibile decidere quale sia la funzione associata ai singoli piedini? Innanzitutto **ogni porta deve essere configurata**, impostando i singoli pin come input o come output. Tale configurazione deve essere effettuata **durante la scrittura del firmware inizializzando in modo mirato i valori di particolari registri** che prendono il nome di registri **TRIS** del PIC. In particolare, **esiste un registro TRIS per ogni porta del PIC, ognuno dei**

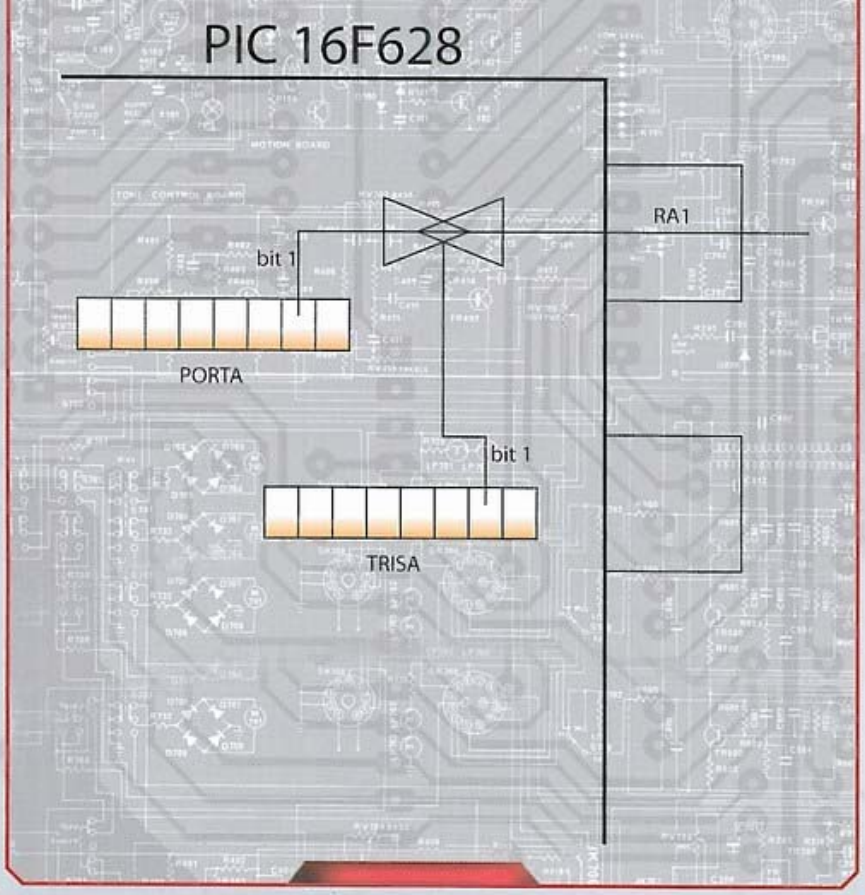
quali è identificato dal nome TRIS seguito dalla lettera identificativa della porta di riferimento (ad esempio 'TRISA' per la porta A, 'TRISB' per la porta B ecc.). Vediamo come utilizzare correttamente questo sistema di configurazione. Abbiamo già osservato come i pin delle porte dei microcontrollori siano 'collegati' per mezzo di un'associazione biunivoca con i bit di alcune variabili specifiche PORT. Il principio alla base dei registri TRIS è praticamente lo stesso: a ogni bit di ogni registro di configurazione è associato uno specifico pin del dispositivo e, in base al valore in esso contenuto, il programmatore può comunicare al microcontrollore la direzione logica del piedino. Nello specifico, impostando a '1' i bit del registro TRIS si configurano i pin corrispondenti come ingressi digitali; al contrario con l'attribuzione del valore '0' si impostano i pin come uscite. Come nel caso delle variabili PORT, il bit meno significativo della variabile TRIS configura il funzionamento del pin 0 della porta, mentre il bit più significativo si occupa di settare la direzione del pin 7. L'ambiente di sviluppo mikroC permette di accedere ai registri di porta e di configurazione utilizzando direttamente i loro nomi mnemonici, come se fossero variabili di 8 bit. Prendendo come esempio il 16F628, che dispone delle due porte A e B, possiamo utilizzare l'I/O digitale intervenendo sui valori delle variabili PORTA e PORTB (per leggere/scrivere i valori dei bit di ingresso/uscita) e impostare il loro verso di funzionamento forzando a '0' o a '1' i bit specifici delle variabili TRISA e TRISB, corrispondenti all'omonimo registro.

IL PRIMO FIRMWARE

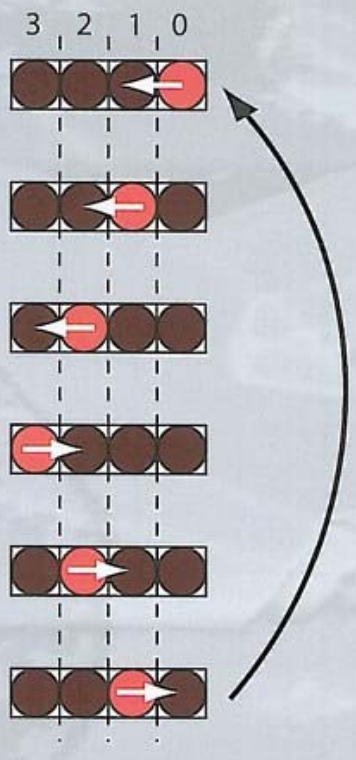
Passiamo ora a realizzare in modo pratico il nostro primo firmware, che ci consentirà di utilizzare un 16F628 (unito a pochi altri componenti) per ottenere un semplice gioco di

I REGISTRI TRIS

L'impostazione del verso di trasferimento dei dati avviene per mezzo dei registri TRIS dei PIC. I registri TRIS sono elementi di memoria di capacità pari a 8 bit. Ogni 'cella' di ogni registro TRIS è associata in modo univoco a un preciso pin del dispositivo e ne controlla il verso di passaggio dei dati. Più dettagliatamente, se un bit di un registro TRIS ha valore '0' il pin a esso associato si comporterà come pin di output digitale. Al contrario, se tale bit viene impostato a valore '1', il pin diviene un ingresso. Nello schema sottostante è rappresentata in modo ideale la relazione logica tra i registri TRISA e PORTA all'interno del PIC 16F628. In particolare è preso come riferimento il pin RA1 (pin 1 della porta A), la cui direzione viene controllata dal bit 1 del registro TRISA. Il bit 1 del registro PORTA, invece, contiene il valore logico usato come output su RA1 (se il bit 1 di TRISA = 0) o il valore logico letto in input dal medesimo pin (se il bit 1 di TRISA = 1).



luci, simile alla celebre banda luminosa che negli anni "80 ha reso inconfondibile l'automobile robotizzata K.I.T.T., protagonista della serie televisiva 'Supercar'. Partiamo dall'**analisi del problema** e ricaviamo da essa una semplice soluzione. Innanzitutto immaginiamo di voler ottenere questo effetto sfruttando una striscia di **4 LED rossi, ognuno dei quali controllato da uno specifico pin del PIC.** Ricordiamo che in modalità di funzionamento digitale, **i pin dei PIC si comportano in modo analogo alle comuni porte logiche utilizzate finora.** Anche i PIC sono quindi caratterizzati da **specifiche correnti di source e di sink,** che nel caso del 16F628 si aggirano attorno ai **25 mA per ogni pin,** con un limite di 250/300 mA complessivi (in pratica, sommando tutte le correnti uscenti dai pin non si deve superare tale soglia limite; i valori dettagliati di source e sink sono disponibili all'interno dei datasheet del dispositivo). **Le porte di I/O del PIC, di conseguenza, possono essere utilizzate come generatori di tensione per alimentare l'accensione di diodi** (ovviamente serve, come sempre, un resistore che limiti la corrente): quando l'output è alto il pin si comporterà come un generatore di tensione da Vcc volt, mentre quando lo stato di output è basso il pin si troverà posto a massa. Nello schema sottostante vediamo una rappresentazione schematizzata della sequenza luminosa che vogliamo riprodurre. In esso vengono rappresentati i 4 LED, che utilizzeremo nel corso dell'esperimento, ordinati e numerati progressivamente da 3 a 0. Ora proviamo a immaginare la medesima sequenza osservandola non sotto un profilo grafico, ma da un punto di vista 'binario'. Lo stato dei 4 LED può essere schematizzato simbolicamente utilizzando una sequenza di quattro bit, nella quale il valore '1' corrisponde all'accensione e il valore '0' allo stato di disattivazione dei diodi corrispondenti. Riesaminiamo tale sequenza. Osservando la tabella binaria in alto, si può notare come ciò che è stato proposto sia, tecnicamente parlando, **un'operazione di 'shift' binario,** operazione che in C è possibile utilizzando i due appositi operatori '<<'



La sequenza di illuminazione dei LED che vogliamo riprodurre equivale a un susseguirsi iterativo di shift binari da una cella. La sequenza binaria è visibile nella tabella in alto a destra.

LA SEQUENZA BINARIA

BIT 3	BIT 2	BIT 1	BIT 0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	1	0	0
0	0	1	0

Il nostro primo firmware ci permetterà di ottenere un effetto simile alla banda luminosa di K.I.T.T., la robo-auto del telefilm 'Supercar'.

(shift a sinistra) e '>>' (shift a destra). Il problema può essere, di conseguenza, scomposto in due fasi distinte e sequenziali: partendo dalla sequenza binaria 0001 effettueremo tre passi da uno shift ciascuno verso sinistra (con il primo otterremo 0010, dopo il secondo 0100 e con il terzo 1000). In seguito invertiremo il processo eseguendo tre nuove operazioni di shift verso destra (partendo da 1000 otterremo 0100 dopo la prima, 0010 dopo la seconda e con l'ultima 0001). Ripetendo 'in loop' questi due blocchi di istruzioni otterremo l'effetto di luce cercato. Ma come scriviamo concretamente tutto ciò? Innanzitutto dobbiamo decidere quali pin utilizzare come uscite digitali addette al controllo dei LED. Una possibilità può essere quella di utilizzare i primi 4 pin della porta B del 16F628 (RB0, RB1, RB2, RB3). A essi collegheremo una semplice rete LED/resistore analoga a quelle mostrate nella figura in basso a destra a pagina 5. Portando 'alto' l'output di questi quattro pin, provocheremo l'accensione dei LED corrispondenti.

PASSIAMO AL CODICE ▸▸▸

Una volta stabilito il semplice circuito elettrico, passiamo alla vera e propria scrittura del codice C che implementerà gli algoritmi visti finora. Prima di dedicarci al sorgente completo, facciamo presente che, per ciò che riguarda le procedure di dichiarazione delle variabili, mikroC rispetta le regole del C e non del C++. Le variabili devono quindi essere dichiarate all'inizio dei blocchi di codice, prima di ogni altra istruzione. Detto ciò dedichiamoci finalmente al sorgente nel nostro primo firmware (mostrato nel box a fondo pagina), in modo da poterlo analizzare passo per passo. Come puoi vedere, la struttura tipica del C è identificabile già dalla prima riga di codice. Come sempre, infatti, la funzione principale del programma viene dichiarata con il nome main. L'uso del tipo 'void' nella sua dichiarazione indica semplicemente che tale funzione non restituisce alcun valore (non è cioè necessario concluderla con un'istruzione

Il primo firmware

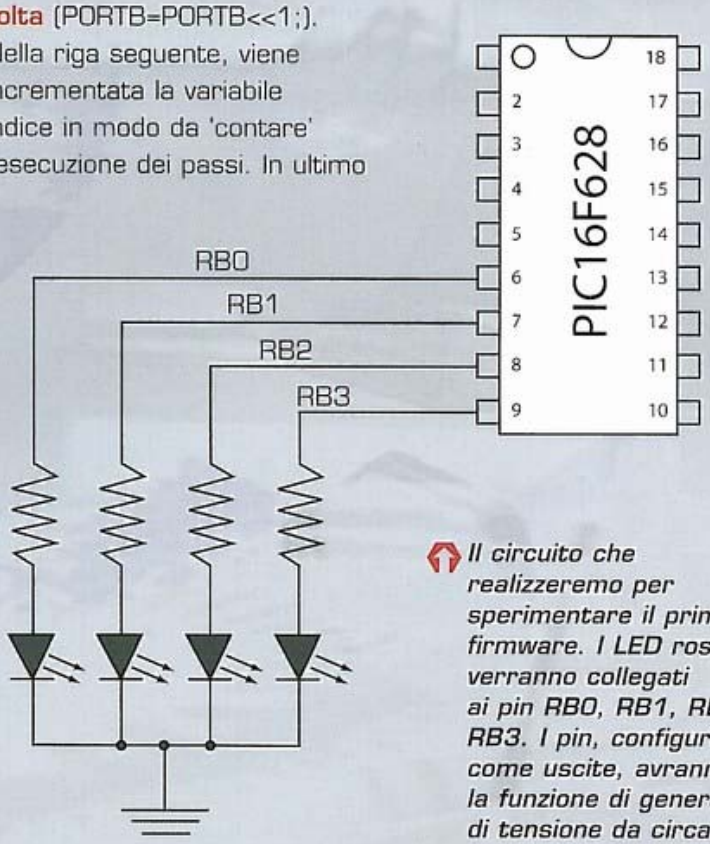
```
void main()
{
      int indice;

      TRISB = 0b00000000;  //tutti i pin della porta B in output
      PORTB = 0b00000001; //impostazione dello stato di uscita della porta B

      while(1)
      {
        indice = 0;
        //shift a sinistra
        while(indice<3)
        {
          PORTB = PORTB<<1;
          indice++;
          Delay_ms(300);
        }

        indice = 0;
        //shift a destra
        while(indice<3)
        {
          PORTB = PORTB>>1;
          indice++;
          Delay_ms(300);
        }
      }
}
```

'return'). L'unica variabile che dichiareremo è **'indice'** (di tipo **int**), che ci servirà come contatore per tener traccia dei passi eseguiti durante le sequenze di shift. Veniamo ora alle due prime vere istruzioni dedicate ai PIC. In esse puoi vedere utilizzate due delle variabili di cui abbiamo discusso nei paragrafi precedenti, ossia **TRISB** e **PORTB (tali 'variabili' non vanno dichiarate, ma sono già definite in modo nativo da mikroC sulla base dell'architettura del PIC utilizzato).** Nella prima di queste istruzioni **impostiamo tutti i bit del registro TRISB a 0** (per esprimere un numero in forma binaria è sufficiente farlo precedere dal prefisso '0b', come mostrato). In tale maniera, il PIC avrà **tutti i pin della porta B configurati come uscite digitali.** Nella riga successiva (istruzione PORTB = 0b00000001;), invece, **imposteremo lo stato di uscita dei pin della porta B, forzando il pin RB0 al valore logico 1 e a 0 gli altri sette piedini.** Inizia successivamente il **ciclo infinito di illuminazione dei quattro LED** (while(1), infatti è un ciclo infinito, in quanto la condizione di permanenza non potrà mai fallire), all'interno del quale sono implementate le **due sequenze di shift binario.** Come prima cosa **inizializziamo a 0 la variabile 'indice'.** Possiamo ora iniziare il **ciclo di shift del registro con un ciclo while il cui corpo verrà ripetuto per tre volte consecutive (condizione indice<3),** all'interno del quale **i bit della variabile di registro PORTB vengono fatti scorrere verso sinistra di una cella alla volta** (PORTB=PORTB<<1;). Nella riga seguente, viene incrementata la variabile indice in modo da 'contare' l'esecuzione dei passi. In ultimo viene chiamata la funzione **Delay_ms().** Tale funzione non

NUMERI BINARI ED ESADECIMALI IN C

Nelle applicazioni di basso livello e in particolar modo nel mondo dell'elettronica è molto comodo poter utilizzare rappresentazioni binarie o esadecimali al posto della comune base decimale. Il linguaggio C offre anche la possibilità di scrivere i numeri in questi formati, semplicemente aggiungendo dei prefissi di due caratteri. In particolare, per comunicare al compilatore che il formato numerico è in base binaria, è sufficiente anteporre alla sequenza di 1 e 0 il prefisso '0b'. Per la rappresentazione esadecimale (ossia in base 16 con simboli che vanno da 0 a F) si impiega, invece, il prefisso '0x' seguito dal numero esadecimale desiderato. Di seguito puoi osservare come inizializzare una variabile intera con il valore decimale '21', riespresso poi in formato binario ed esadecimale (tutte e tre le istruzioni effettuano di fatto la stessa operazione, cambia solo il formato di rappresentazione del valore numerico considerato).

```
int variabile = 21;        /*assegnamento del numero 21 (base 10)
                           in base 10*/
int variabile = 0b10101;   /*assegnamento del numero 21
                           (base 10) espresso in binario (10101)*/
int variabile = 0x15;      /*assegnamento del numero 21
                           (base 10) espresso in esadecimale (15)*/
```



Il circuito che realizzeremo per sperimentare il primo firmware. I LED rossi verranno collegati ai pin RB0, RB1, RB2, RB3. I pin, configurati come uscite, avranno la funzione di generatori di tensione da circa 5 V.

è una funzione standard C, ma è definita e implementata all'interno dell'ambiente mikroC. Il suo scopo è quello di **generare un ritardo nell'esecuzione** (implementata mediante una condizione di stallo controllato) **di una quantità di millisecondi pari al valore passato come parametro**. Nel nostro caso, quindi, prima di passare al passo successivo dell'effetto luminoso **introdurremo un ritardo di 300 ms** (che potrai modificare per accelerare o rallentare l'effetto). Terminati i tre passi di questo ciclo viene eseguito il secondo ciclo while, identico a quello appena osservato, ma con **direzione di shift verso destra** (PORTB = PORTB>>1;), che riporterà il sistema allo stato iniziale. Nel prossimo fascicolo vedremo come creare e compilare il progetto del firmware nell'ambiente mikroC.

PER GRANDI 'CARICHI'›››

Nel corso dell'articolo (e in particolare a pagina 3) abbiamo visto che anche il microcontrollore 16F628, così come tutti i dispositivi integrati, è soggetto a vincoli elettrici molto rigorosi che riguardano i limiti delle correnti assorbite (correnti di sink) ed emesse (correnti di source). Se nel caso preso in considerazione in questo articolo è stato sufficiente utilizzare gli output digitali del PIC come generatori di tensione a bassa corrente, inserendo semplici resistori in serie ai diodi utilizzati per poterne controllare l'accensione, ciò non è vero in termini più generali. Nel caso dei robot e dei sistemi di automazione, infatti, vi è spesso la necessità di impiegare componenti elettromeccanici che richiedono correnti notevolmente superiori a quelle erogabili dai comuni microcontrollori digitali (non possiamo pretendere, ad esempio, di alimentare un motore elettrico sfruttando le correnti di source dei PIC). Come in altre situazioni osservate in passato, anche ora possiamo far ricorso all'uso dei transistor. Collegando, ad esempio, una serie di MOSFET tramite i loro gate ai pin del microcontrollore (configurati come uscite digitali), possiamo pilotare carichi (come motori o LED ad alta luminosità) agendo direttamente sui valori logici delle uscite. Nell'esempio sottostante è mostrato un semplice schema elettrico nel quale un generico PIC a 18 pin (analogo per ipotesi al 16F628) viene utilizzato come sistema di controllo digitale per un motorino, proprio grazie all'uso di due transistor MOSFET a canale N. Nell'esempio in questione, il pin RB5 viene utilizzato per pilotare l'accensione del diodo a emissione luminosa, mentre il pin RB4 viene utilizzato per attivare il motore elettrico M. Da osservare che, in questo caso, il motore può operare esclusivamente in modo ON/OFF monodirezionale. La programmabilità e la versatilità dei PIC, tuttavia, permetterebbero di pilotare il motore interfacciandosi direttamente con un ponte ad H (identico a quello che hai potuto sperimentare nel corso di questo Workshop). Se si ipotizza, poi, di sfruttare alcuni pin del PIC come ingressi (con segnali provenienti da sensori esterni), si può sviluppare un sistema in grado di comandare attuatori e luci in funzione dei dati ambientali, proprio come avviene nei sistemi di controllo dei robot.