Prima parte
La programmazione base

Seconda parte
Le variabili, le costanti e gli ingressi

Terza parte
Utilizzare al meglio le uscite del PIC

Mikrobasic per PICmicro

Finalmente un corso molto semplice, ma allo stesso tempo efficace, per imparare a programmare i microcontrollori PIC, prodotti dalla famosa casa Microchip. Questa volta utilizziamo il linguaggio ad alto livello Basic, molto semplice ed intuitivo ma estremamente potente, che permette tempi di sviluppo e di manutenzione molto bassi, con il vantaggio di realizzazioni veloci e progetti immediati.

Dobbiamo proprio ammetterlo, l'elettronica è un settore ormai inarrestabile che, giorno dopo giorno, raggiunge livelli sempre più elevati nel campo della precisione, della velocità e della miniaturizzazione. Se cinquant'anni fa, in un metro cubo riuscivano a stare mille valvole, oggi, in un solo centimetro quadrato entrano comodamente miliardi di transistor molto più efficienti e veloci. Un esempio pratico? Pensiamo ai comodissimi *Pendrive* da 512 Mb, utili accessori che si collegano alle porte USB del computer e fungono da memorie di massa: ebbene, questi piccoli oggetti che stanno comodamente in tasca contengono circa 5 miliardi di "interruttori" capaci di memorizzare tantissime informazioni.
Chi si occupa d'elettronica deve oggi adeguarsi all'era **informatica** e **digitale**. È impossibile considerare le due cose separate. Basti pensare ai telefoni cellulari, ai moderni televisori, ai lettori DVD, alle centraline delle automobili, ai decoder satellitari e videogiochi: tutte queste apparecchiature non contengono al loro interno i normali circuiti elettronici, bensì sofisticati congegni debitamente **programmati** con opportuni linguaggi di programmazione.

L'ABITO NON FA IL MONACO

Andiamo a conoscere dunque questo strano componente, dall'aspetto umile ma misterioso, che non mancherà davvero di stupirci. Cos'è innanzitutto un microcontrollore? È un circuito integrato, di piccole dimensioni, la cui funzione non è stabilita a priori dal produttore, ma è programmata dall'utente finale o progettista. Un esempio ci chiarirà meglio il concetto: se acquistiamo un circuito integrato CD40106, possiamo solamente utilizzarlo quale sestupla porta NOT e nulla più, mentre un integrato CD4026 espleterà la sua funzione solamente come contatore e decoder LED. Essi non eseguiranno mai altre funzioni diverse da quelle per cui sono stati "programmati".
Invece un microcontrollore, se pur fisicamente identico ai suddetti IC, può compiere i più disparati compiti, sia semplici che enormemente complessi: tutto dipende dalla capacità del programmatore umano di impartire i comandi corretti, ossia di scrivere con un opportuno linguaggio, il **programma.** Facile a dirsi, difficile a farsi.
Nel corso di queste puntate daremo i primi passi verso questa difficile "arte", anticipando però già da ora che la *perfezione* si raggiungerà solo dopo un lungo e attento studio e sperimentazione.

PRIMO APPROCCIO

I PIC sono dei microcontrollori, che opportuna-

di Giovanni Di Maria
gdm1967@libero.it



La programmazione base

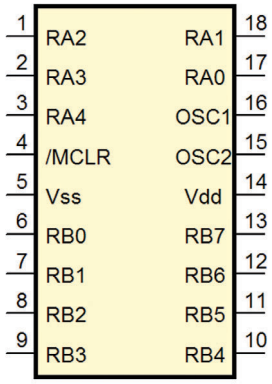


Figura 1 *Piedinatura del PIC16F84*

mente programmati, eseguono molto velocemente i compiti impartiti. In altri termini costituiscono un intero sistema programmabile digitale. Internamente sono costituiti da complessi dispositivi che somigliano molto, anche se in maniera ridotta, ad un personal computer.
Le parti fondamentali contenute sono: la **CPU** (Central Process Unit) che ha la funzione di eseguire una dopo l'altra le istruzioni programmate; la **PROM** (Progammable Read Only Memory) che contiene il programma scritto dall'uomo; la **EEPROM** (Electrical Erasable Programmable Read Only Memory) che può memorizzare i dati utente anche senza alimentazione elettrica; la memoria **RAM** (Random Access Memory) che contiene provvisoriamente i dati transitori e le variabili; le **PORTE** di Input e Output, con cui il PIC comunica con il mondo esterno e ad esse possono essere collegate qualunque tipo di apparecchiature, naturalmente con opportuni interfacciamenti elettronici.
Durante il nostro studio utilizzeremo il **PIC16F84** un microcontrollore molto popolare, costruito dalla **Microchip** (www.microchip.com), il quale presenta ottime caratteristiche di affidabilità, robustezza, anche alle inversioni di polarità, versatilità ed, oltretutto, è abbastanza economico. Inoltre può essere programmato praticamente un numero infinito di volte. Tale microcontrollore lavora con una tensione di 5 Volt CC. Non sono ammesse tolleranze e differenze elevate; in ogni caso è molto semplice ottenere questa tensione con opportuni regolatori.
Per un approfondimento sui microcontrollori PIC e la loro programmazione vi consiglio l'acquisto del libro *"Conoscere ed usare - PICmicro"* scritto da *Tiziano Galizia* e *Maurizio Del Corso* ed edito dalla *INWARE Edizioni.* Maggiori informazioni le trovate all'indirizzo www.farelettronica.com/libri.

LE FUNZIONI DEI 18 PIN

Il nostro PIC16F84 si presenta in contenitore DIL (dual in line) con 18 piedini, 9 per fila. Ogni pin ha un suo specifico compito, ma il fatto sorprendente è che, 13 di questi costituiscono le porte di ingresso e uscita: le funzioni "vitali" sono quindi svolte solamente dai rimanenti 5 piedini.
In figura 1 riportiamo la disposizione dei piedini (vista dall'alto dell'integrato).

Ogni piedino ha una sua funzione ben precisa:

- Il pin 14 (**VDD**) va collegato alla linea positiva dell'alimentazione (+5V).
- Il pin 5 (**VSS**) va collegata alla massa comune del circuito.
- Il pin 4 (**MCLR**) di norma va collegato al positivo dell'alimentazione; se riceve un breve impulso logico 0, viene eseguito il reset del chip, ed il programma inizia da capo (non preoccupatevi, non viene cancellato nessun dato).
- I pin 16 e 15 (**OSC1** e **OSC2**) servono per generare la cadenza del clock del micro. Approfondiremo bene questo concetto.
- I pin 6, 7, 8, 9, 10, 11, 12, 13 (**RB0-RB7**) costituiscono la PORTB, ossia 8 bit che si possono utilizzare per l'ingresso e l'uscita dei segnali logici.
- I pin 17, 18, 1, 2, 3 (**RA0-RA4**) costituiscono

la **PORTA**, ossia 5 bit che si possono utilizzare per l'ingresso e l'uscita dei segnali logici.

Tredici pin per la comunicazione con il mondo esterno sembrano moltissimi, ma vedrete che "l'appetito viene mangiando".

SCHEMA "BASE" APPLICATIVO

Il microcontrollore, per poter funzionare, deve essere collegato a pochissimi altri componenti esterni, che ne determinano le condizioni di funzionamento e soprattutto di velocità operativa. Sostanzialmente esistono due modi per fare ciò: una economica e "poco" precisa, l'altra meno economica ma molto precisa. Vediamoli entrambi.
Come detto prima, ai pin 15 e 16 vanno collegati alcuni componenti che determinano la velocità del clock e quindi di tutto il PIC. Scegliendo addirittura certi specifici valori, si può elevare il clock del micro (overclocking), ossia lo si può far funzionare a velocità maggiori di quelle previste dal chip stesso.
Chiaramente a nostro rischio e pericolo, in quanto l'intero circuito non garantirebbe quella precisione assicurata dalla casa produttrice e inoltre si avrebbe un consumo di corrente molto maggiore.

Collegamento con rete RC

L'assemblaggio più semplice è quello visualizzato in figura 2. Viene utilizzata una rete composta da una resistenza ed un condensatore per generare il segnale di clock. Si utilizza questa soluzione quando servono applicazioni dove la precisione non è fattore fondamentale.

Collegamento con rete QUARZATA

È sicuramente la configurazione preferita e precisa. Se si utilizza un quarzo da 4 Mhz, si otterrà dal software sviluppato, e quindi dal micro, una precisione dei tempi sbalorditiva, anche su quelli lunghi. In figura 3 è rappresentato lo schema base di funzionamento. Data la sua semplicità esso può essere assemblato tranquillamente su basetta mille fori. Si notino ai capi del quarzo i due condensatori ceramici da 22 pF.

Bene, abbiamo già abbastanza materiale per iniziare a lavorare. Entrambi gli schemi funzionano egregiamente, ma non sono capaci di visualizzare al mondo esterno i propri risultati e le elaborazioni, ne tanto meno, di compiere azioni meccaniche e simili. Più avanti vedremo come si fa. In ogni caso sappiate che questi schemi costituiscono già da soli una sorta di *Pic Test Board* con la quale si possono già effettuare i primi esperimenti e i test sul software sviluppato.

I LINGUAGGI DI PROGRAMMAZIONE

Per comunicare le proprie intenzioni ad un

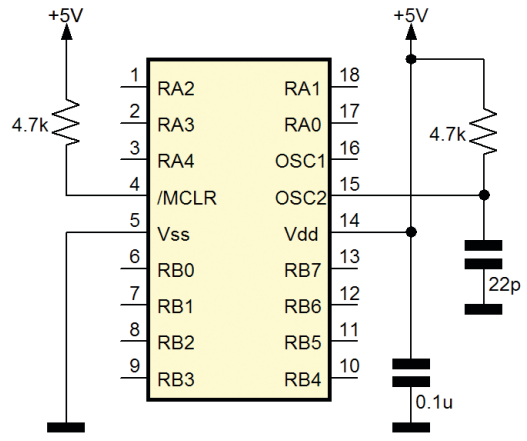


Figura 2 Schema BASE con rete RC

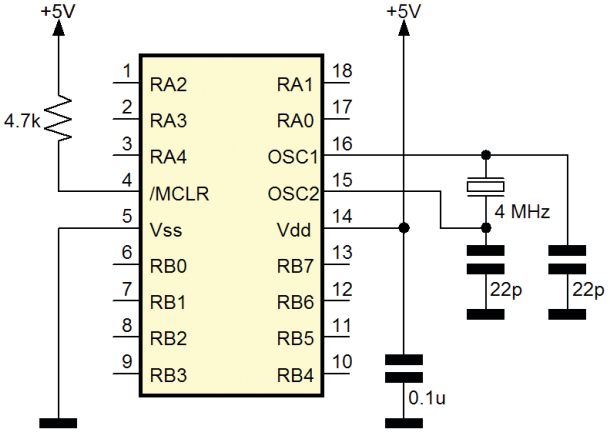


Figura 3 Schema BASE con quarzo

microcontrollore (o ad un computer) è necessario usare lo stesso linguaggio elettronico. Purtroppo esso è costituito solo da simboli elementari (0 e 1), davvero incomprensibili per gli esseri umani. Sono stati perciò ideati alcuni strumenti, in grado di tradurre i linguaggi con semantica umana in struttura binaria, che prendono il nome di interpreti e compilatori. Approfondiamo adesso il concetto di linguaggio di programmazione. Viene definito *programma* l'insieme sequenziale di comandi ed istruzioni lecite e corrette che la macchina esegue senza possibilità di errore al fine di risolvere un problema. A pensarci bene anche l'uomo, durante le sue azioni giornaliere, segue uno schema ordinato e sequenziale di compiti e doveri, predefiniti o meno che siano. Se vogliamo, in un esempio pratico, potremmo descrivere, con questo *pseudo-linguaggio*, le fasi salienti della sua giornata:

- Ore 07:00 ascolta la sveglia.
- Si alza dal letto.
- Va in bagno.
- Apre il rubinetto.
- **SE** l'acqua è fredda apre quella calda.
- Chiude il rubinetto.
- Va in camera da letto.
- Si veste.
- Va in cucina.
- Prepara il caffé.
- Versa il caffè nella tazza.
- Versa un cucchiaino di zucchero.
- **Ripete** l'azione dello zucchero per 3 volte.
- … e così via.

Come si può notare, anche nella nostra vita pratica, vengono seguite ed eseguite alcune azioni semplici, ma altre avvengono SOLO in certe situazioni (esempio dell'acqua calda), o altre ancora si ripetono per un certo numero di volte (esempio del cucchiaino di zucchero).
È proprio questo, il programma, con la "sola" differenza che per spiegare un comportamento ad un microcontrollore occorre la conoscenza di un linguaggio ad esso comprensibile.

Linguaggi di alto e basso livello

A seconda se i linguaggi utilizzati si avvicinano maggiormente al lessico umano o a quello elettronico, possiamo distinguerli in *alto* e *basso livello.* È linguaggio di alto livello quello che è simile alla lingua parlata umana (di solito inglese) con costrutti lessicali e semantici ad alto *livello*. È linguaggio a basso livello invece quello che somiglia alla codifica elettronica. In figura 4 vediamo questa suddivisione. Naturalmente entrambi presentano vantaggi e svantaggi. Vediamo quali. Il linguaggio ad alto livello è semplice da apprendere, è facilmente gestibile e la sua manutenzione non comporta problemi di sorta; è però lento da eseguirsi ed il codice generato occupa molta memoria elettronica. Ma il neo principale consiste nella perdita di tutte quelle finezze e quelle conoscenze di ciò che succede all'interno del controllore, in quanto sono già scontate e trasparenti. Il linguaggio a basso livello invece è molto difficile da imparare e l'apporto di modifiche al codice potrebbe risultare molto pesante; però la sua esecuzione è molto veloce ed il codice generato è piccolissimo. Per poter programmare con tale linguaggio è necessaria la **completa** conoscenza dell' hardware utilizzato. Se vogliamo fare un paragone, un po' spartano a dire la verità, è come se cucinassimo una busta di pasta già pronta (linguaggio ad alto livello) invece di prepararla completamente in casa (linguaggio a basso livello). La scelta dipende da come si vuole gestire il proprio tempo e denaro. I tempi di sviluppo del software, in un linguaggio ad alto livello, sono molto ridotti: in poco tempo un prototipo viene consegnato ed eventualmente modificato con pochissime conseguenze. Tra i linguaggi ad alto livello ricordiamo il Basic, il Pascal e il C mentre tra quelli a basso livello domina l'Assembler. In questo corso studieremo il semplice ma potente linguaggio **Basic.**

IL BASIC

È uno dei più "antichi" linguaggi di programmazione. È nato per dare ai principianti la possibilità di programmare gli elaboratori elettronici con facilità, senza il bisogno di conoscere a fondo la struttura dei processori. Infatti la parola Basic sta per **B**eginner's **A**ll Purposes **S**ymbolic **I**nstruction

Code (Codice simbolico di istruzioni per principianti adatto a tutte le applicazioni). È composto da un certo numero di comandi e istruzioni (in lingua inglese) che, nella loro sequenzialità, sono rigorosamente eseguiti dal microcontrollore. Tale linguaggio è una sorta di compendio, di sommario di tutto ciò che deve svolgere il processore. Nel corso delle puntate approfondiremo i vari concetti, considerando che, con l'elettronica di oggi, il tecnico deve assumere il duplice ruolo di **progettista** elettronico nonché di **programmatore** software. La capacità del programmatore consisterà pertanto nell'includere in maniera esatta tutte le funzioni che il processore dovrà svolgere, anche perché basterebbe un piccolissimo errore di codice a mandare in tilt l'intero sistema (bug). Come vedrete in seguito, anche il più complesso programma, avrà sempre e solo uno scopo: quello di far cambiare, secondo le necessità del progetto, lo stato logico di una o più porte del PIC (0 Volt – 5 Volt).

IL MIKROBASIC

Per i nostri progetti e i nostri esperimenti utilizzeremo l'innovativo e potente mikrobasic, prodotto dalla **mikroElektronika**, azienda iugoslava. La nostra scelta è caduta su tale soluzione per alcuni motivi che di seguito elenchiamo:

- È un completo ed efficiente ambiente IDE (Integrated Development Environment), ossia in un unico pacchetto software sono compresi l'editor del sorgente, il compilatore Basic, il debugger ed il programmatore per scrivere sul nostro PIC).
- Dispone di un convertitore in notazione HEX, BIN e DEC (utilissimo).
- Dispone di un Code Explorer avanzato.
- Presenta on ottimo assistente di codice.
- Permette la colorazione del codice sorgente contestualmente alla sintassi del codice.
- Possiede un esauriente Help on-line.
- Genera automaticamente il file HEX.
- Visualizza il codice Assembler del programma.
- Mostra le statistiche grafiche dell'uso delle risorse.
- Presenta molte utilità.
- …. e molto ancora.

In figura 5 vediamo come si presenta la schermata principale del compilatore, la cui versione demo (limitata a 2K di codice) si può scaricare dal sito di Fare Elettronica (www.farelettronica.com).

ACCENDIAMO IL NOSTRO PRIMO LED

Vediamo adesso come, con semplici operazioni, è possibile far illuminare un diodo led collegato

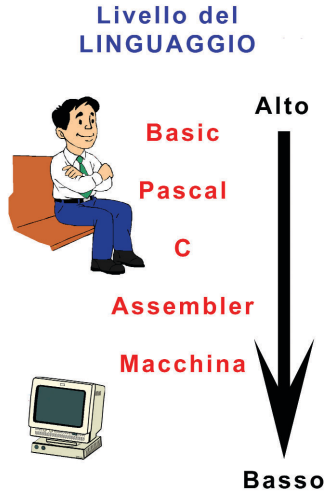


Figura 4 Linguaggi di Alto e Basso livello

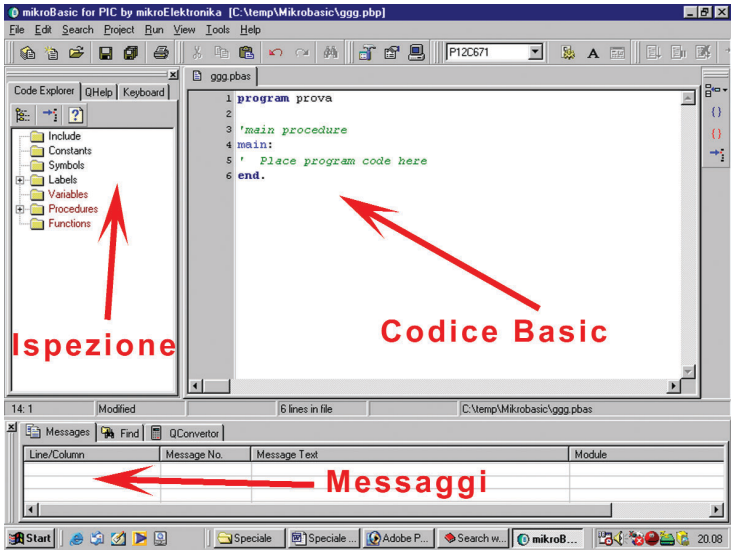


Figura 5 Schermata dell'ambiente IDE

al nostro PIC.
Approntiamo lo schema visibile in figura 6, collegando anche un diodo led in serie ad una resistenza di limitazione alla porta RB0 del PIC.
Creiamo in mikrobasic un nuovo progetto, definendone il nome a nostro piacimento, la cartella di destinazione ed il tipo di microcontrollore utilizzato (nel nostro caso PIC16F84A).
Abilitiamo anche i check box relativi al **WDT_OFF e XT_OSC** (vedi figura 8). Nell'area del codice scriviamo il seguente programma:

```
program led
main:
  trisb.0=0   'Definisce il pin
              'RB0 come Uscita

  portb.0=1   'Pone il pin RB0 a
              'livello logico alto 1 (5V)
end.
```

Se avete scritto correttamente il codice, potete tranquillamente compilare selezionando il menu *Run* e scegliendo la voce *Compile.*
Il mikrobasic genererà un file con estensione .HEX. Sarà proprio questo il file che dovrà essere copiato all'interno del nostro PIC attraverso un programmatore hardware connesso alla porta seriale o Usb del nostro computer, come la scheda *EasyPIC 2* che potete trovare sul sito di Fare Elettronica nella sezione *shop.*
Nel secondo caso potremmo usare lo stesso Mikrobasic per effettuare tale operazione, altrimenti potremmo utilizzare il programmatore presentato nel numero 222 (Dicembre 2003) di Fare Elettronica.
Alimentiamo il nostro circuito e vedremo con sorpresa che il diodo LED collegato alla porta RB0 si illumina.
Certo, utilizzare un microcontrollore per compiere solamente un'operazione così semplice non sarebbe certamente la soluzione più economica, però ci serve per capire come un programma, tramite istruzioni ordinate e sequenziali, riesca a far cambiare il livello logico, e quindi il voltaggio, di una porta del PIC.

Esaminiamo attentamente le istruzioni del nostro primo programma. La prima linea **program** informa il compilatore che da quel punto inizia un programma. La seconda linea **main:** serve a definire un punto di riferimento chiamato label o etichetta. Più avanti spiegheremo di cosa si tratta. La terza linea **trisb.0=0** comunica al PIC che il pin RB0 sarà considerato come uscita e non come ingresso. La quarta linea **portb.0=1** invece pone a 1 logico il pin RB0 del PIC. Con un tester noteremo che tra tale piedino e la massa si misurano esattamente 5 V. Infine la quinta linea **end.** comunica appunto la fine sequenziale delle istruzioni e quindi del programma.

INGRESSO E USCITA

Bene, spegniamo per un attimo il circuito e spieghiamo dettagliatamente come avviene la

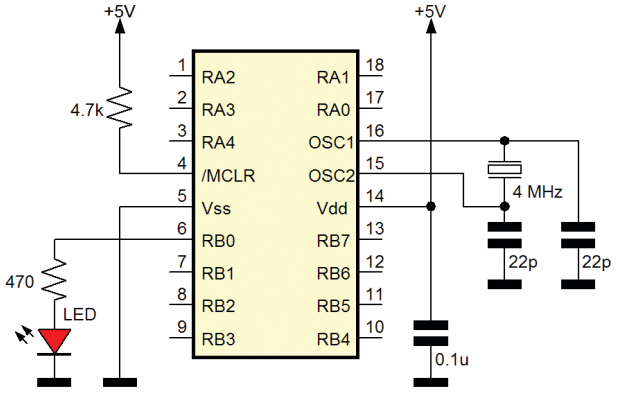


Figura 6 Schema di prototipo con LED

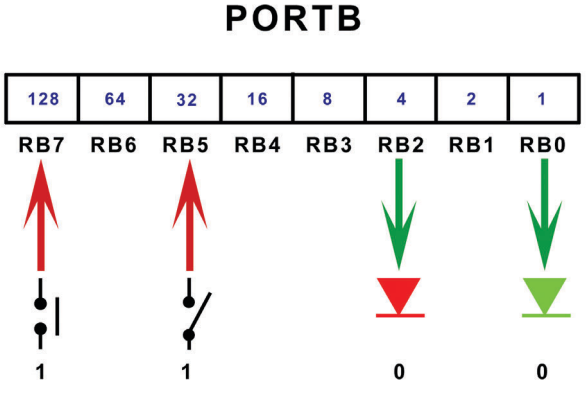


Figura 7 La porta B in ingresso e uscita

programmazione delle porte. Come detto in precedenza, il PIC dispone di 13 pin utilizzate per comunicare con il mondo esterno, cioè la PORTB formata da 8 bit (contrassegnati RB0, RB1, RB2, RB3, RB4, RB5, RB6, RB7) e la PORTA formata da 5 bit (contrassegnati RA0, RA1, RA2, RA3, RA4). Ognuno di questi bit e/o pin può essere configurato indipendentemente come ingresso (a cui possiamo collegare un pulsante, un interruttore, un sensore o altro) oppure come uscita (a cui possiamo collegare un led o un transistor di potenza per pilotare carichi maggiori). Per gestire al meglio queste porte si usano diversi metodi, alcuni peraltro molto semplici. Consigliamo vivamente di utilizzare sempre i registri per la configurazione di esse, vale a dire il registro **TRIS** ed il registro **PORT.**

Consideriamo per esempio la PORTB composta da 8 ingressi/uscite.

Il PIC non potrà mai sapere in anticipo se è nostra intenzione collegare su RB0 un LED verde, su RB2 un LED rosso, su RB5 un interruttore e su RB7 un pulsante: dobbiamo essere noi, poveri programmatori, a renderlo noto. Questa assegnazione di compiti è effettuata appunto grazie al registro **TRISA** (per la Porta A) e **TRISB** (per la Porta B).

Guardiamo per un attimo la figura 7. La porta B è utilizzata per illuminare alcuni diodi colorati e per accettare dall'esterno dei segnali. Facciamo anche riferimento alla figura 1. I terminali collegati ai diodi LED sono chiaramente delle **uscite,** mentre i terminali collegati a pulsanti ed interruttori sono **ingressi.** Ebbene, per comunicare al PIC questa nostra scelta, occorre assegnare al registro TRISB un opportuno valore. Ciò si può effettuare in due modi possibili:

1. Specificare per ogni pin la sua funzione di Input o Output (ed in particolare occorre dare il valore **0** se si tratta di uscita, **1** se si tratta di ingresso) con queste istruzioni:

```
trisb.0=0   'OUT
trisb.2=0   'OUT
trisb.5=1   'In
trisb.7=1   'In
```

In questo modo abbiamo definito la funzione specifica per ogni pin. Attenzione a non porre mai a livello logico 1 un ingresso (specialmente se è direttamente cortocircuitato a massa). Il numero dopo il punto indica quale pin si deve programmare (da 0 a 7).

2. Usando solo una istruzione "cumulativa" per tutti i pin (basta sommare il numero blu della figura 7 in corrispondenza degli ingressi (valore logico 1):

```
trisb=160 '(128+32)
```

In un'unica istruzione abbiamo inglobato tutte le funzioni della porta B. Come si nota, non è stato specificato il numero del singolo pin. Le stesse regole valgono per la PORTA A, con la sola differenza che essa ha solamente 5 pin contro gli 8 della porta B.

LA TEMPORIZZAZIONE

Il micro esegue in un secondo 1 milione di istruzioni macchina, numero di tutto rispetto per un chip così piccolo.

Ci sono però molte occasioni in cui questa velocità non serve. Un esempio concreto è rappresentato da un lampeggiatore. Se non mettessimo tra i vari lampeggi del led un' istruzione che in qualche modo ritardi gli impulsi, i lampeggi stessi avanzerebbero con una tale rapidità che l'occhio umano non si accorgerebbe di nulla.

Ecco quindi il nostro prossimo progetto: quello di realizzare un lampeggiatore a Led con cadenza di un secondo (1000 millisecondi). Utilizziamo ancora lo schema di figura 6. Creiamo un nuovo progetto, chiamandolo *lampeggio*, specifichiamo la cartella di lavoro, definiamo il PIC 16F84A quale microcontrollore e abilitiamo i flags WDT_OFF e XT_OSC. Scriviamo quindi il seguente programma:

```
program lampeggio
main:
```

```
  trisb.0=0   'RB0 come uscita
inizio:
  portb.0=1 'Accende LED
  Delay_ms(1000)'Wait 1 sec
  portb.0=0 'Spegne LED
  Delay_ms(1000)'Wait 1 sec
  goto inizio 'Ripete sempre
end.
```

In questo listato vi sono alcuni concetti nuovi da chiarire. Spieghiamo dettagliatamente cosa avviene in ogni linea di programma, partendo dalla prima:

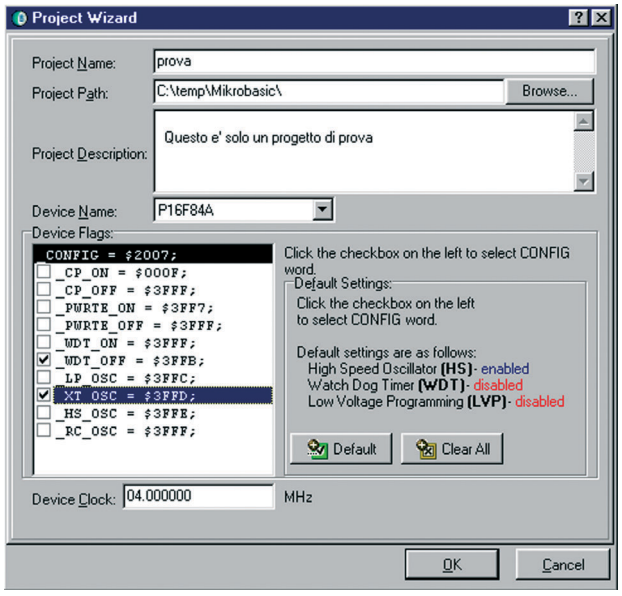


Figura 8 *Configurazione di un progetto*

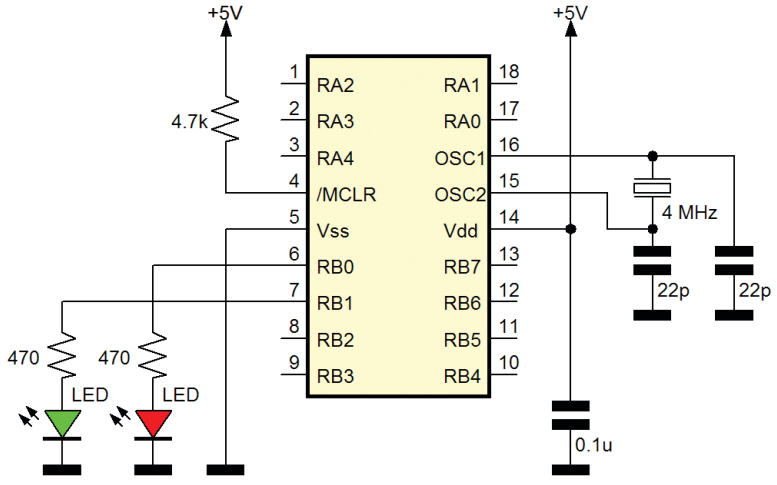


Figura 9 *Schema con due diodi led*

1. Inizia il programma dal nome *lampeggio*.
2. Definisce la label o punto di riferimento (main).
3. Definisce il pin RB0 come uscita.
4. Definisce la label inizio.
5. Mette a 1 logico il pin RB0.
6. Aspetta 1 secondo (1000 ms).
7. Mette a 0 logico il pin RB0.
8. Aspetta 1 secondo (1000 ms).
9. Ritorna a eseguire le istruzioni dalla label **inizio.**
10. Fine (non viene praticamente mai eseguita).

Compiliamo il programma e con il programmatore built-in o con IC-Prog (o Pic Prog) carichiamo il file *lampeggio.hex* sulla memoria del PIC.
Collegando l'alimentazione al nostro circuito vedremo che il diodo LED lampeggia esattamente alla cadenza di un secondo.

Come vedete, il metodo migliore per imparare l'utilizzo del PIC è proprio quello di provare e provare ancora, magari sbagliando. Male che vada mandiamo in fumo il microcontrollore ma, per il suo costo estremamente contenuto, vale proprio la pena di tentare. Dovete sapere anzi che i processori utilizzati circa 50 anni fa costavano milioni di dollari, erano grandi quanto un appartamento e a livello di prestazioni non si avvicinavano neppure lontanamente al nostro 16F84. Vedrete col tempo che le routines di temporizzazione saranno sicuramente le più utilizzate: in un progetto occorre sempre conciliare quelli che sono i tempi macchina ai ritmi più lenti dell'uomo.
Il nostro compilatore dispone di due utili istruzioni per mettere il PIC in pausa per un certo tempo: la prima è *delay_ms(t)* che ferma l'esecuzione dei comandi per t millesimi di secondo; la seconda è *delay_us(t)* che ferma l'esecuzione dei comandi per t milionesimi di secondo.

AUMENTIAMO IL NUMERO DELLE USCITE

Inseriamo adesso un secondo diodo LED al circuito, in modo da averne uno rosso su RB0 e uno verde su RB1. Tali porte vanno quindi considerate come delle uscite (output). Realizziamo lo schema presente in figura 9.

Il nostro scopo stavolta è quello di far lampeggiare alternativamente i due LED, al ritmo di mezzo secondo. Le cose cominciano pian piano a complicarsi. Per gestire meglio gli eventi proviamo a tracciare uno schema a blocchi. Creiamo allora la seguente flow-chart o diagramma di flusso, che spiega passo per passo cosa avviene nel PIC in maniera schematica, come da figura 10.

Creiamo con il compilatore un nuovo progetto, specificando sempre gli stessi parametri visti in precedenza, e scriviamo questo codice:

```
program lampeggio
main:
  trisb.0=0   'RB0 come uscita
  trisb.1=0   'RB1 come uscita
inizio:
  portb.0=1 'Accende LED 1
  portb.1=0 'Spegne LED 2
  Delay_ms(500)'Wait 0,5 sec
  portb.0=0 'Spegne LED 1
  portb.1=1 'Accende LED 2
  Delay_ms(500)'Wait 0,5 sec
  goto inizio 'Ripete sempre
end.
```

Dando tensione al circuito vedremo i due diodi LED lampeggiare alla frequenza di mezzo secondo.

DEDUZIONI

Tutto ciò che abbiamo fatto sino ad ora ci porta a trarre le seguenti conclusioni:

- Un software può essere scritto in tanti modi diversi, anche secondo il proprio stile di ragionamento.
- Il PIC segue ciecamente quanto programmato, quindi se certi risultati sono errati, il problema è da imputare all'uomo e non alla macchina.
- Non esiste problema che non può essere risolto con un linguaggio di programmazione.
- Scrivere in Basic è molto semplice, ma scrivere in Assembler dà più conoscenze dell' hardware utilizzato, anche sacrificando chiarezza di codice e tempi di sviluppo.
- La velocità di esecuzione del software scritto dipende anche dal quarzo esterno utilizzato.
- Le uscite del PIC possono solo "pilotare" piccolissimi carichi. Per ottenere potenze maggiori è necessario utilizzare alcuni componenti per amplificare il segnale, ma ciò sarà approfondito successivamente.
- In un programma occorre prevedere TUTTE le possibili evenienze, anche quelle più remote.
- Abituatevi a inserire molti commenti nel codice scritto: commentare vuol dire scrivere appunti e annotazioni nel codice stesso che spiegano le funzioni svolte. Ciò vuol dire massima leggibilità nostra o di altri programmatori, anche in fase di debug e modifiche varie.
- Le uscite o ingressi del PIC sono "solamente" 13. È possibile però con accorgimenti particolari aumentare questo numero.

IL NOSTRO PRIMO SEMAFORO

Bene, siamo già maturi per creare il prossimo progetto, quello di un semaforo stradale. Dobbiamo innanzitutto analizzare cosa avviene durante un ciclo semaforico completo:

1. Si accende il verde. Le automobili passano.
2. Il verde continua a restare acceso per 30 secondi.
3. Mentre il verde è acceso, si illumina il giallo per 5 secondi.
4. Si accende il rosso. Il verde e il giallo si spengono.
5. Il rosso resta acceso per 30 secondi.
6. Si spegne il rosso ed il ciclo ricomincia da capo.

Ricordate sempre che l'analisi del problema deve essere studiata in maniera perfetta e soprattutto

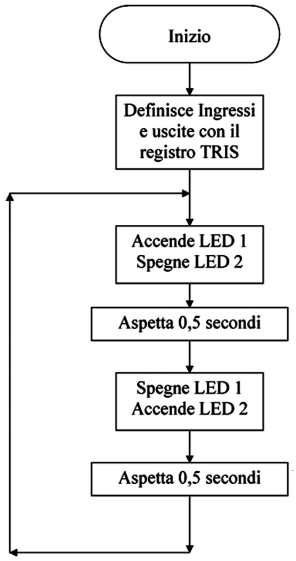


Figura 10 Flow chart per il lampeggio alternato di due led

deve essere esente da errori di valutazione. In definitiva il PIC è solo il "braccio" che esegue i comandi, e la "mente" è rappresentata dal programmatore.

Assembliamo lo schema visibile in figura 11, che prevede il LED rosso su RB0, il LED giallo su RB1 ed il LED verde su RB2.

Adesso scriviamo il codice Basic, tenendo sott'occhio l'analisi che abbiamo prima sviluppato:

```
program Semaforo

main:
  'Definiamo RB0 RB1 RB2
  'come output

  trisb.0=0 'RB0 OUT
  trisb.1=0 'RB1 OUT
  trisb.2=0 'RB2 OUT

  'Spegniamo inizialmente
  'tutte le tre porte
  'per sicurezza e per
  'evitare eventuali
  'shock iniziali alle
  'periferiche collegate
  portb.0=0 'RB0 OFF
  portb.1=0 'RB1 OFF
  portb.2=0 'RB2 OFF

  inizio:   'label del ciclo

  portb.2=1 'Verde ON
  Delay_ms(30000) 'Wait 30 sec.

  portb.1=1 'Anche giallo ON
  Delay_ms(5000) 'Wait 5 sec.

  portb.2=0 'Verde OFF
  portb.1=0 'Giallo OFF

  portb.0=1 'Rosso ON
  Delay_ms(30000) 'Wait 30 sec.
  portb.0=0 'Rosso OFF

  goto inizio 'Ricomincia tutto

end.
```

Compiliamo il programma, carichiamolo sul nostro PIC e vedremo con grande soddisfazione che il semaforo funziona egregiamente, rispettando i tempi programmati ed eseguendo infinite volte l'intero ciclo con incredibile affidabilità.
Prima dell'avvento dei microcontrollori tali progetti erano realizzabili solo utilizzando decine di circuiti integrati, flip-flop, contatori, convertitori, che rendevano complicato uno schema semplice come il nostro. C'è da dire anche che in passato, una modifica al progetto avrebbe implicato la completa ricostruzione dei prototipi mentre con il PIC basta modificare solamente il *programma* per rendere subito efficace la variazione. Senza intervenire minimamente sul circuito! Come detto prima, lo schema base può già da solo costituire una valida *Pic Test Board,* ossia una scheda per effettuare tutte le prove possibili ed immaginabili. Basta dotare i terminali del microcontrollore di alcuni morsetti per permettere il collegamento provvisorio a eventuali periferiche esterne, come Led, lampade, pulsanti e interruttori. In figura 12 potete notare un esempio di tutto questo. Per la sua notevole semplicità, la Test Board può essere approntata anche su basetta mille fori.

L'ALGORITMO

Dopo questi esempi si nota subito una grande differenza tra come "pensa" la macchina e come pensa l'uomo. La macchina ha necessità di avere specificate, passo dopo passo, le varie operazioni da eseguire, e, seppur semplici, que-

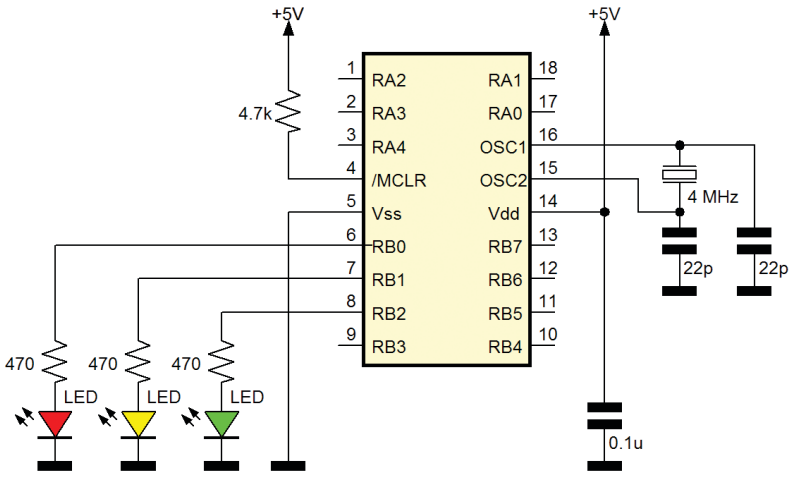
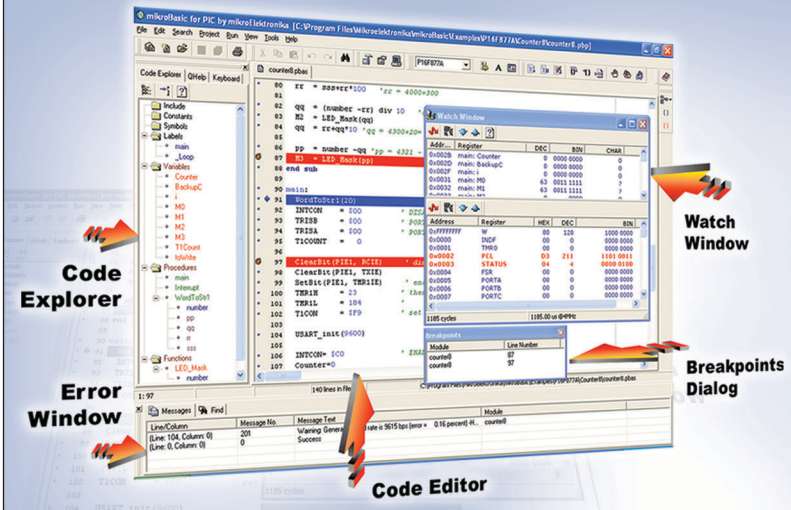


Figura 11 Schema elettrico del Semaforo

ste devono essere spiegate in modo molto dettagliato. L'uomo invece ha un approccio totalmente differente: il suo pensiero neurale deriva da confronti e deduzioni complesse che, forse, la macchina non riuscirà mai ad emulare.Il programmatore deve quindi risolvere un determinato problema sostituendosi alla macchina e pensando come essa. Deve trovare il metodo di risoluzione, codificarlo e insegnarlo al processore. Se egli non ha chiari questi concetti, difficilmente il software sviluppato funzionerà bene. Introduciamo allora un argomento importantissimo, che segna la teoria del software e che traccia i fondamentali passaggi del suo sviluppo: **l'Algoritmo**. Esso si può definire come una sequenza di compiti successivi eseguibili in modo meccanico per risolvere un

MIKROBASIC
UN POTENTE COMPILATORE BASIC PER PICmicro



MIKROBASIC:
✔ Code Editor
✔ Code Explorer
✔ Debugger
✔ Statistiche

Tutto in un ambiente
Windows facile ed intuitivo

Un set di strumenti veramente indispensabili
per sviluppare applicazioni con i PICmicro

Speciale
riservato ai lettori di Fare Elettronica
aggiungi il codice FE237MB alle note del tuo ordine ed avrai diritto ad uno sconto del 20%*

*promozione valida fino al 01/04/2005

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

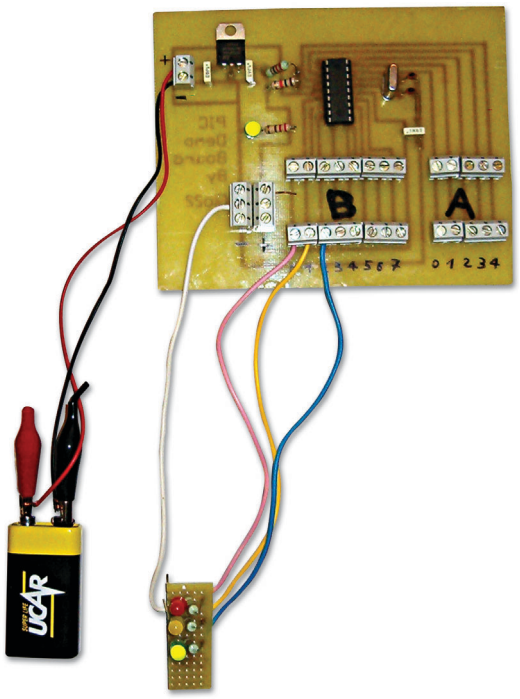


Figura 12 La Pic Test Board con il semaforo

problema. Per facilitare la comprensione e la schematizzazione, esso può venir rappresentato tramite un *diagramma di flusso*, ossia uno schema a blocchi, dove ogni forma rappresenta un'azione che il calcolatore deve eseguire. Nel corso degli eventi utilizzeremo molto i diagrammi di flusso (o Flow chart), ma vedrete che con la pratica se ne può anche farne a meno.

PRIMO APPROCCIO AL CICLO ITERATIVO

I programmi fin'ora visti sono molto semplici e fanno uso di poche risorse software e hardware. Quando le esigenze operative aumentano, anche il linguaggio di programmazione ci viene sempre più d'aiuto, mettendoci a disposizione ulteriori mezzi di sviluppo. Supponiamo di risolvere il seguente problema: si hanno a disposizione due lampadine (o LED), una rossa e una verde. Inizialmente la verde è spenta. La rossa deve invece lampeggiare per 5 (cinque) volte alla frequenza di 1 Hz (una volta al secondo). Al termine del lampeggio si deve accendere permanentemente il LED verde. Realizziamo nuovamente lo schema proposto in figura 9. Con le conoscenze attuali, il programma (peraltro *funzionante* ma *NON funzionale*) è il seguente:

```
program Lampeggio2

main:
   'definisce funzione pin
   trisb.0=0 'RB0 out
   trisb.1=0 'RB1 out

   'spegne tutto
   portb.0=0 'RB0 OFF
   portb.1=0 'RB1 OFF

   '--Primo lampeggio--
   portb.0=1 'rosso ON
   Delay_ms(500)
   portb.0=0 'rosso OFF
   Delay_ms(500)

   '--Secondo lampeggio--
   portb.0=1 'rosso ON
   Delay_ms(500)
   portb.0=0 'rosso OFF
   Delay_ms(500)

   '--Terzo lampeggio--
   portb.0=1 'rosso ON
   Delay_ms(500)
   portb.0=0 'rosso OFF
   Delay_ms(500)

   '--Quarto lampeggio--
   portb.0=1 'rosso ON
   Delay_ms(500)
   portb.0=0 'rosso OFF
   Delay_ms(500)

   '--Quinto lampeggio--
   portb.0=1 'rosso ON
   Delay_ms(500)
   portb.0=0 'rosso OFF
   Delay_ms(500)

   '--Finalmente accende verde
   portb.1=1 'Verde ON

end.
```

Compilate il programma e dopo il suo scaricamento sul PIC vedrete cinque lampeggi del LED rosso e poi la definitiva accensione del LED verde. Questa soluzione potrebbe andare bene, ma se per ipotesi il numero dei lampeggi dovesse aumentare da *cinque* a *cento* inizierebbero i problemi. Dovremmo riscrivere per altre 95 volte il blocco di codice relativo alla sequenza; ne verrebbe un programma molto lungo, sia per quanto riguarda il codice che per l'eseguibile. Probabilmente il PIC stesso lo rifiuterebbe.
Ecco allora che il linguaggio Basic ci mette a disposizione una funzione estremamente potente, che ci permette di ripetere *tante volte* un insieme di istruzioni. Tale soluzione prende il nome di **ciclo**. Ritornando al nostro problema, guardiamo bene il nostro listato: vediamo che c'è una parte di codice che, ripetendosi per cinque volte, è sempre la stessa.
Ci riferiamo all'accensione e allo spegnimento del diodo rosso, con i relativi tempi di attesa. Modifichiamo il sorgente con l'uso dei cicli, poi commenteremo:

```
program Lampeggio2

dim k as byte

main:

  trisb.0=0 'RB0 out
  trisb.1=0 'RB1 out

  portb.0=0 'RB0 OFF
  portb.1=0 'RB1 OFF

  for k=1 to 5
    portb.0=1 'rosso ON
    Delay_ms(500)
    portb.0=0 'rosso OFF
    Delay_ms(500)
  next k

  '--Finalmente accende verde
  portb.1=1 'Verde ON

end.
```

Compilando ed eseguendo sul PIC il programma noteremo che il risultato finale è lo stesso. Guardando però il sorgente esso è passato dalle iniziali 36 righe di codice a sole 16 righe, ed anche il file eseguibile risulta molto più snello e *leggero*.
Quali sono le novità? Innanzitutto non ci sono più i 5 blocchi di codice che servivano ad accendere e spegnere il LED rosso, ma ne è rimasto uno solo. Tale blocco è "chiuso" all'interno di un *ciclo di ripetizione*, cioè una sorta di scatola che viene ripetuta per un numero indefinito. Inizia con il comando **for k=1 to 5** e termina con il comando **next k**.
Letteralmente vuol dire "considera per K tutti i valori numerici da 1 a 5"; in altre parole, tutto ciò che è racchiuso tra la parola chiave **for** e la parola **next** sarà ripetuto per cinque volte.
Dato che k rappresenta un valore numerico (avremmo potuto usare qualsiasi altra lettera), lo abbiamo inserito all'inizio del programma usando la definizione di variabile (dim k as byte). Un ulteriore esempio chiarirà meglio le idee. Riferendoci stavolta allo schema di cui in figura 11, abbiamo la necessità di far lampeggiare alla cadenza di 2 Hz il LED rosso per 5 volte, il LED giallo per 10 volte ed il LED verde per 7 volte. Studiamo attentamente il listato:

```
program Lampeggio3

dim ripeti as byte

main:
  trisb.0=0 'RB0 Out
  trisb.1=0 'RB1 Out
  trisb.2=0 'RB2 Out
  '--Spegne TUTTA la PORTB
  portb=0 '00000000 (8 bit a zero)

inizio:
  '-Ciclo per diodo ROSSO su RB0-
  for ripeti=1 to 5
    portb.0=1 'RB0 ON
    Delay_ms(250)
    portb.0=0 'RB0 OFF
```

```
    Delay_ms(250)
  next ripeti

  '-Ciclo per diodo GIALLO su RB1-
  for ripeti=1 to 10
    portb.1=1 'RB1 ON
    Delay_ms(250)
    portb.1=0 'RB1 OFF
    Delay_ms(250)
  next ripeti

  '-Ciclo per diodo VERDE su RB2-
  for ripeti=1 to 7
    portb.2=1 'RB2 ON
    Delay_ms(250)
    portb.2=0 'RB2 OFF
    Delay_ms(250)
  next ripeti

  goto inizio  'Ripeti TUTTO

end.
```

Stavolta abbiamo utilizzato la variabile **ripeti**, che ha il compito di contare il numero lampeggi dei vari LED. Il nome della variabile può essere qualsiasi, purché non infranga le regole del mikrobasic.
Dedicheremo il prossimo capitolo allo studio delle variabili.
Grazie ai cicli, il codice risulta essere *enormemente* più corto e ottimizzato, più leggibile e facilmente modificabile, in quanto la parte da reiterare è inserita tra i comandi FOR e NEXT.
Riprendiamo un po' di fiato provando un piccolo esercizio.
Colleghiamo a TUTTI i pin della PORTB otto diodi LED (uno per ogni pin) con la relativa resistenza di limitazione da 470 ohm.
Dopo la definizione di output della PORTB, occorre programmare i seguenti comportamenti dei LED:

1. Spegnimento iniziale di tutti i LED.
2. Si accende il primo LED per 3 secondi.
3. Si accende il secondo LED per 3 secondi (il primo resta acceso).
4. Si spengono i due LED restando spenti per 2 secondi.
5. Si accendono i LED 3, 4, 5 per 5 secondi.
6. Si spengono quindi tali LED e si accendono i LED 6, 7, 8 per 5 secondi.
7. Si spengono tutti i LED.
8. I LED 1, 3, 5, 7 lampeggiano assieme (ON-OFF) alla cadenza di 1 secondo, per 10 volte.
9. Si accendono TUTTI i LED per 10 secondi(il PIC inizia a sudare …).
10. Finalmente tutti i LED si spengono (con una sola istruzione).

Mamma mia, stiamo iniziando a fare sul serio. Potete inviare le risposte alla e-mail di intestazione dell'articolo, nonché quesiti su eventuali problemi e malfunzionamenti del circuito.

Avete mai visto le centraline di allarmi e antifurto? Se avete notato, all'accensione del circuito, vi sono alcuni LED che lampeggiano per un numero definito di volte, facendo intendere che l'antifurto è in fase di "riscaldamento" o di preparazione al funzionamento. Ebbene, questi cicli di lampeggio sono proprio effettuati tramite microcontrollore, che coordina pazientemente tutte queste operazioni.
Pian piano vedrete che alcuni concetti saranno sempre più chiari: basta solamente molta costanza e studio.
Un consiglio che diamo è quello di rileggere tante volte l'articolo e di studiarlo, magari sforzandovi di afferrare un concetto inizialmente non troppo chiaro.

UN PICCOLO GIOCO DI LUCI

Ricordate il telefilm Supercar? Il protagonista aveva sulla sua vettura un fila di lucine che si illuminavano da destra verso sinistra e viceversa. Ebbene, simuliamo questo gioco di luci con il microcontrollore.
Per il nostro scopo proviamo tre sorgenti Basic, tutti funzionanti, ma uno più "perfezionato" dell'altro.

Realizziamo per questo progetto lo schema che vedete in figura 13. Su tutte le otto uscite della PORTB sono collegati altrettanti diodi LED con la relativa resistenza di limitazione.

Il programma deve illuminare il primo LED; dopo 100 ms. (un decimo di secondo) deve accendere il secondo LED, il terzo e, così via, sino all'ottavo LED, poi il settimo, il sesto sino al secondo LED.
Il ciclo così continua all'infinito.
In figura 14 vediamo l'ordine di accensione dei LED nelle 14 sequenze cicliche (ogni sequenza è numerata progressivamente).

Primo Metodo

Il primo metodo prevede l'accensione di un *singolo* LED e lo spegnimento contemporaneo del precedente.
In questa maniera il codice Basic risultante potrebbe essere molto lungo:

```
program SuperCar
main:
  trisb=0 'Definisce la
          'PORTB tutta in
          'Uscita (%00000000)

portb=0 'Spegniamo tutti i Pin
        'Equivale a %00000000

ciclo:
  portb.0=1 'Led 1 ON
  delay_ms(100)

  portb.0=0 'Led 1 OFF
  portb.1=1 'Led 2 ON
  delay_ms(100)

  portb.1=0 'Led 2 OFF
  portb.2=1 'Led 3 ON
  delay_ms(100)

  portb.2=0 'Led 3 OFF
  portb.3=1 'Led 4 ON
  delay_ms(100)

  portb.3=0 'Led 4 OFF
  portb.4=1 'Led 5 ON
  delay_ms(100)

  portb.4=0 'Led 5 OFF
```

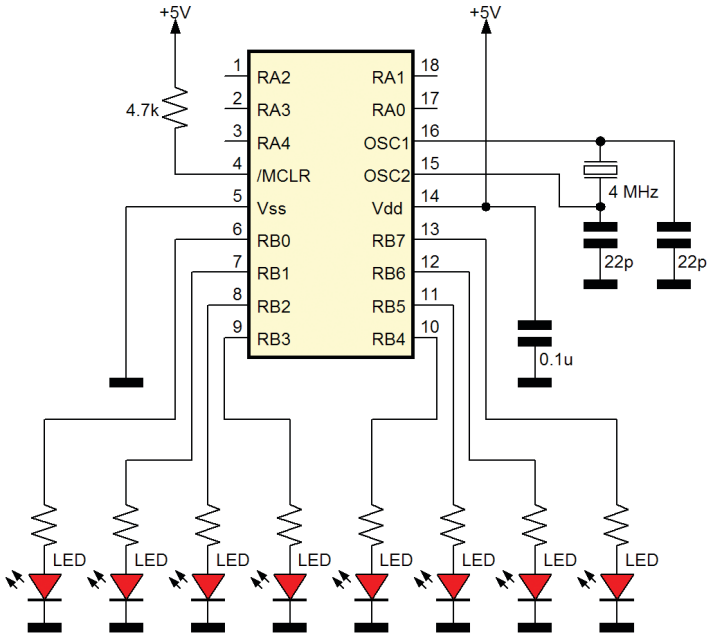


Figura 13 Schema con 8 LED su PORTB

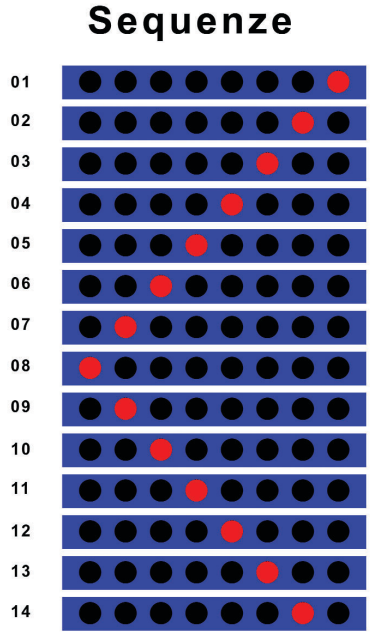


Figura 14 La sequenza di accensione

```
    portb.5=1 'Led 6 ON
    delay_ms(100)

    portb.5=0 'Led 6 OFF
    portb.6=1 'Led 7 ON
    delay_ms(100)

    portb.6=0 'Led 7 OFF
    portb.7=1 'Led 8 ON
    delay_ms(100)

    portb.7=0 'Led 8 OFF
    portb.6=1 'Led 7 ON
    delay_ms(100)

    portb.6=0 'Led 7 OFF
    portb.5=1 'Led 6 ON
    delay_ms(100)

    portb.5=0 'Led 6 OFF
    portb.4=1 'Led 5 ON
    delay_ms(100)

    portb.4=0 'Led 5 OFF
    portb.3=1 'Led 4 ON
    delay_ms(100)

    portb.3=0 'Led 4 OFF
    portb.2=1 'Led 3 ON
    delay_ms(100)

    portb.2=0 'Led 3 OFF
    portb.1=1 'Led 2 ON
    delay_ms(100)

    portb.1=0 'Led 2 OFF

    goto ciclo

end.
```

Secondo Metodo

Con il secondo metodo, paragoniamo gli 8 LED della PORTB a 8 bit di un byte. Pertanto, se i vari pin della porta rappresentano il numero binario della porta stessa, per abilitare le varie uscite dovremo fornire l'esatto valore corrispondente, secondo questa tabella di sequenza (il simbolo 1 indica il Led acceso mentre il simbolo 0 indica il Led spento):

01) 00000001 = 1
02) 00000010 = 2
03) 00000100 = 4
04) 00001000 = 8
05) 00010000 = 16
06) 00100000 = 32
07) 01000000 = 64
08) 10000000 = 128
09) 01000000 = 64
10) 00100000 = 32
11) 00010000 = 16
12) 00001000 = 8
13) 00000100 = 4
14) 00000010 = 2

Bene, adesso che disponiamo dei vari "pesi" numerici, possiamo scrivere un codice sicuramente più corto e più compatto:

```
program SuperCar2

main:
  trisb=0 'Definisce la
          'PORTB tutta in
          'Uscita (%00000000)

  portb=0 'Spegniamo tutti i Pin
          'Equivale a %00000000

  ciclo:

    portb=1
    delay_ms(100)
    portb=2
    delay_ms(100)
    portb=4
    delay_ms(100)
    portb=8
    delay_ms(100)
    portb=16
    delay_ms(100)
    portb=32
    delay_ms(100)
```

```
    portb=64
    delay_ms(100)
    portb=128
    delay_ms(100)
    portb=64
    delay_ms(100)
    portb=32
    delay_ms(100)
    portb=16
    delay_ms(100)
    portb=8
    delay_ms(100)
    portb=4
    delay_ms(100)
    portb=2
    delay_ms(100)

    goto ciclo

end.
```

Terzo Metodo

Il terzo metodo risulta essere il più corto ed ottimizzato di tutti. Utilizza variabili ed array, un concetto che approfondiremo nella prossima sezione. Ecco il listato, poi passiamo ai commenti:

```
program SuperCar3

const sequenza as
byte[14]=(1,2,4,8,16,32,64,128,64,32,16,8,4,
2)
dim k as byte

main:
  trisb=0 'PORTB OUTPUT
  portb=0 'Spegne portb
  ciclo:
    for k=0 to 13
      portb=sequenza[k]
      Delay_ms(100)
    next k
    goto ciclo
end.
```

Fantastico, lo stesso programma è divenuto cortissimo, sia in termini di codice sorgente, sia in termini di spazio eseguibile. Con l'esperienza si riuscirà a scrivere programmi sempre più ottimizzati. Si potrebbe fare ancora di meglio: comunque meglio non esagerare... almeno per il momento.
In breve diciamo che il metodo utilizza un nuovo concetto: l'uso di *variabili* e di *array* che approfondiremo nel prossimo articolo.
Cosa abbiamo fatto? In pratica abbiamo memorizzato nella memoria del PIC i 14 passi corrispondenti alla combinazione di LED che vogliamo illuminare. Il programma riprodurrà tali valori sugli 8 LED con tempo di attesa di 100 millisecondi.
Eseguite stavolta questo programma:

```
program SuperCar4

const sequenza as
byte[16]=(1,128,2,64,4,32,8,16,0,129,0,129,0
,255,0,255)
dim k as byte

main:
  trisb=0 'PORTB OUTPUT
  portb=0 'Spegne portb
  ciclo:
    for k=0 to 15
      portb=sequenza[k]
      Delay_ms(100)
    next k
    goto ciclo
end.
```

Il programma è lo stesso del precedente. Cambiano solo le sequenze delle luci da accendere. Provate il listato ed otterrete un simpatico gioco di luci.

CONCLUSIONI

Nella prossima lezione approfondiremo l'utilizzo delle *variabili* e delle *costanti* in Basic. Vedremo come gestire gli *array* e come interagire con il microcontrollore attraverso i suoi ingressi.
Naturalmente realizzeremo nuove ed interessanti esercitazioni.

Prima parte
La programmazione base

Seconda parte
Le variabili, le costanti e gli ingressi

Terza parte
Utilizzare al meglio le uscite del PIC

Mikrobasic per PICmicro

Questo mese approfondiremo l'utilizzo delle variabili e delle costanti in Basic. Vedremo come gestire gli array e come interagire con il microcontrollore attraverso i suoi ingressi. Un altro passo fondamentale nella gestione dei PIC con un linguaggio ad alto livello come il Basic.

LE VARIABILI E LE COSTANTI

Senza le informazioni e i dati da elaborare i programmi non esisterebbero, o meglio non potrebbero produrre i risultati sbalordivi che invece normalmente forniscono. Questi dati risiedono normalmente in alcune celle di memoria. Possono venire gestiti attraverso le variabili e le costanti.

Le variabili

La variabile è semplicemente uno spazio in memoria che può contenere un dato. Come dice il nome, tale dato può essere soggetto a modificazioni e cancellazioni durante l'esecuzione del programma. Possiamo immaginare le variabili come una sorta di scatole, all'interno delle quali si possono inserire dei valori numerici. Siccome solitamente vi sono diverse scatole, è necessario differenziarle tra di loro, semplicemente assegnando un nome che possa distinguerle.
In figura 1 abbiamo due scatole, una piccola

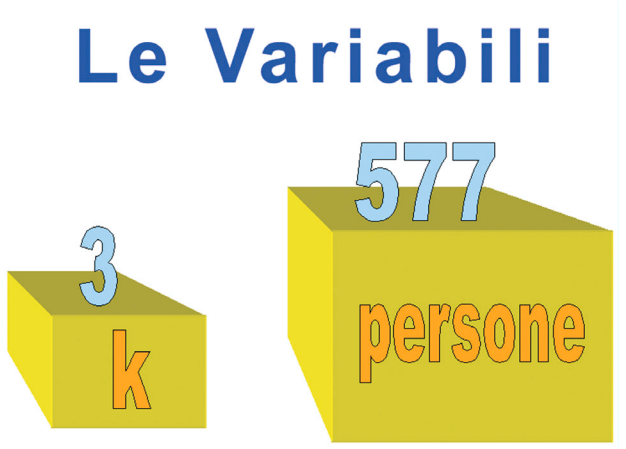


Figura 1 Concetto di variabile

ed una grande. La scatola piccola è stata arbitrariamente chiamata ***k***. In realtà possiamo dare alle variabili qualsiasi nome lecito, ma è sempre meglio sceglierne uno appropriato, in base al contenuto.
Dal momento che la variabile k deve "contenere" un numero piccolo (nel nostro caso il numero 3), la relativa scatola deve essere *piccola*. A destra vediamo invece un grande scatolone, cioè la variabile ***persone***, che contiene un grosso numero (il 577). Si deduce quindi che lo spazio preposto a contenere i valori deve essere accuratamente "dimensionato" in base al massimo valore previsto.
Non conviene usare quindi un grosso scatolone se dobbiamo inserirvi solamente un piccolissimo numero.
Non può invece verificarsi il caso contrario, ossia un grosso numero non può entrare in una scatola piccola. Adesso vedremo come si programmano le variabili e come si sceglie la relativa dimensione.
Il Mikrobasic ci mette a disposizione un comando per dimensionare queste "scatole" in base al loro massimo contenuto previsto. Il

Le variabili, le costanti e gli ingressi

di Giovanni Di Maria
g.dimaria@farelettronica.com



Tipi di dato		
Tipo	**Spazio occupato in memoria**	**Intervallo di valori**
byte	8 bit	0 ÷ 255
char	8 bit	0 ÷ 255
word	16 bit	0 ÷ 65535
short	8 bit	-128 ÷ 127
integer	16 bit	-32768 ÷ 32767
longint	32 bit	-2147483648 ÷ 2147483647

contenuto, come vedremo meglio in seguito, è sempre numerico, anzi in ogni locazione di memoria è sempre contenuto un valore massimo di 255 (ossia un byte). Ecco i tipi di dato che gestisce il compilatore:
Come si vede dalla tabella, a seconda del valore massimo che una variabile assumerà durante l'esecuzione di un programma, occorre usare un tipo dato appropriato. Così ad esempio, se prevediamo che l'indice di un contatore non supererà MAI il valore di 100, utilizzeremo il tipo di dato **byte**. Potremmo usare anche il tipo word o altro più capiente, ma consumeremo inutilmente memoria RAM che, credetelo, è veramente molto limitata. Approntiamo lo schema di cui alla figura 13 presentato nella puntata precedente (FE 237 - Marzo 2005): 8 LED collegati in PORTB.
Esaminiamo il seguente programma che, a dire il vero, potrebbe essere un po' ostico da capire:

```
program scanner

dim k as byte
dim inizio as byte
dim fine as byte

main:
  trisb=0 'PORTB OUTPUT
  portb=0 'Spegne portb
  inizio=30 'Stabilisce punto iniziale
  fine=200 'Stabilisce punti finale
  ciclo:
    for k=inizio to fine
      portb=k 'Visualizza il byte
      Delay_ms(100)  'Aspetta
    next k
end.
```

Si inizia ad informare il compilatore che verranno utilizzate 3 variabili di tipo byte (poiché il loro contenuto, durante l'esecuzione, non supererà il valore 255) tramite l'istruzione **dim**. Dopo aver definito la portb completamente in output (trisb) ed azzerato la stessa, stabiliamo che il conteggio deve iniziare da 30 (variabile *inizio*) e terminare a 200 (variabile *fine*). Il ciclo for farà iterare la variabile k affinché, di volta in volta, assuma il valore sequenziale vincolato da tali limiti. Durante questo lavoro, verrà assegnato alla portb il valore reale della variabile k, comportandosi quindi da monitor di visualizzazione del contenuto della variabile k. Con il termine di assegnazione si intende quella operazione mediante la quale un valore numerico viene "memorizzato" nella variabile.
Un esempio di assegnazione è questo:

```
k = 200
limite = 33000
tempo = 60
gradi = 200
secondi = 30
ore = minuti * 60
```

Come si vede, il valore posto alla destra del simbolo uguale (=) viene assegnato alla variabile di sinistra. Si realizza in altre parole l'immissione di un valore numerico all'interno di una casella di memoria. Le variabili vengono rilasciate, ossia perse, non appena si toglie l'alimentazione al PIC.
La dichiarazione delle variabile va sempre eseguita prima della label ***main***.

Le costanti

A differenza delle variabili, le costanti sono entità il cui contenuto non cambia durante l'esecuzione di un programma.
È quindi utile definirle tali in quanto il compilatore ottimizza il loro utilizzo in termini di spazio occupato e velocità. Quando un'informazione resta immutata nel corso della sua vita (informatica), questa può tranquillamente essere considerata come costante. Nella codifica solitamente è buona norma scriverle in MAIUSCOLO.
Ecco nell'esempio come vanno dichiarate le costanti:

```
program costanti

const NESSUNO      = 0     'Tutti Led spenti
const TUTTI        = 255   'Tutti Led accesi
const PRIMOULTIMO = 129   'Primo e Ultimo ON
const CENTRALI     = 24    'Led centrali ON

main:

  trisb=0 'PORTB in OUTPUT
  portb=0 'Spegne portb

  portb=TUTTI 'Accende TUTTI i LED
  Delay_ms(1000)
  portb=NESSUNO 'Spegne tutti i LED
  Delay_ms(1000)
  portb=PRIMOULTIMO 'Accende primo e ultimo
LED
  Delay_ms(1000)
  portb=CENTRALI 'Accende i due LED centra-
li
  Delay_ms(1000)
  portb=0 'Spegne TUTTO

end.
```

Esaminiamo il software. Dopo la dichiarazione della quattro costanti, e dopo l'inizializzazione della PORTB, tutti i LED si illumineranno per un secondo. Perché?
Perché la costante TUTTI (che vale 255) è assegnata direttamente alla PORTB, e se si converte il numero decimale 255 in notazione binaria, il risultato sarà 11111111, cioè otto cifre abilitate, (otto LED accesi). Effettuiamo adesso la conversione di queste quattro cifre decimali in cifre binarie e vediamo il risultato:

$(255)_{10} = (11111111)_2$
$(0)_{10} = (00000000)_2$
$(129)_{10} = (10000001)_2$
$(24)_{10} = (00011000)_2$

Come si può facilmente notare, le otto cifre rappresentano i nostri otto diodi Led. Se sono tutti accesi è ovvio che stanno visualizzando il byte 255, vale a dire 11111111.
Tutto questo è il *digitale*. L'1 rappresenta il segnale presente, la tensione, la presenza di informazione.
Lo 0 rappresenta il silenzio, assenza di tensione o meglio tensione nulla quindi il Led spento.
Se volete accendere un numero di Led predefiniti dovete convertire sempre la relativa rappresentazione binaria in decimale. Oppure potete abilitare individualmente ogni singola porta, come abbiamo visto nei primi esempi.

Gli array o vettori

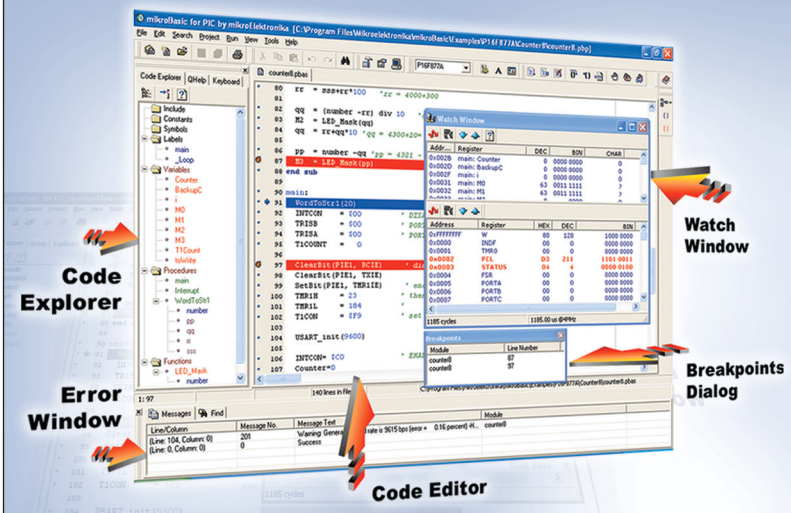
Le variabili e le costanti sono oggetti indipendenti l'una dall'altra. Può essere però a volte utile (se non indispensabile) trattarle come membri di una stessa famiglia, e quindi con lo stesso nome, differenziandole solo da un numero progressivo che le accompagni.
Nel precedente programma SuperCar3 abbiamo memorizzato i valori dei Led da illuminare in un vettore di costanti. Esaminiamo attentamente la riga di dimensionamento:

```
const sequenza as
byte[14]=(1,2,4,8,16,32,64,128,64,32,16,8,4
,2)
```

Questo comando crea quattordici costanti di nome ***sequenza***.
Non si tratta di una costante semplice, ma di una famiglia di costanti, tutte chiamate "*sequenza*" ma ognuna "codificata" con un numero progressivo da 0 a 13.
Pertanto l'assegnazione riserverà in memoria ben 14 celle, ognuna delle quali ospiteranno rispettivamente i valori di 1, 2, 4, 8 e così via.
Perché si usano gli array? Non sarebbe più semplice usare le normali variabili o costanti? Ovviamente in questo caso no, poiché avere a disposizione una lista organizzata permette al programma di esaminare i dati più facilmente e sequenzialmente; basta infatti inserire la parte di codice da elaborare all'interno di un efficiente ciclo *FOR-NEXT*.
In questo modo la lunghezza del codice programma è sempre la stessa, indipendentemente dal numero di elementi che compongono il vettore stesso.
Con l'assegnazione di costanti a vettore, il programma si comporta come se ogni elemento singolo fosse così inizializzato:

```
sequenza[0] = 1
sequenza[1] = 2
sequenza[2] = 4
sequenza[3] = 8
sequenza[4] = 16
sequenza[5] = 32
sequenza[6] = 64
sequenza[7] = 128
```


MIKROBASIC
UN POTENTE COMPILATORE BASIC PER PICmicro



✔ Code Editor
✔ Code Explorer
✔ Debugger
✔ Statistiche

Tutto in un ambiente
Windows facile ed intuitivo

Un set di strumenti veramente indispensabili
per sviluppare applicazioni con i PICmicro

Speciale
riservato agli abbonati di Fare Elettronica
aggiungi il codice FE239MB alle note del tuo ordine ed avrai diritto ad uno sconto del 10%*

*promozione valida fino al 31/05/2005, non cumulabile con altre promozioni in corso

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

```
sequenza[8] = 64
sequenza[9] = 32
sequenza[10] = 16
sequenza[11] = 8
sequenza[12] = 4
sequenza[13] = 2
```

Con una sola riga di comando abbiamo effettuato tutte le 14 assegnazioni. Non possiamo però modificare il contenuto in quanto si tratta pur sempre di una costante.
Per definire un vettore di tipo "variabile" si usa invece questa codifica:

```
dim vett as byte[4]
```

dove *vett* rappresenta un vettore di quattro elementi di tipo byte, numerati da 0 a 3. In altri termini è come se avessimo quattro variabili: invece di avere nomi diversi hanno un numero di riconoscimento, ossia un *indice*. Come si vede in figura 2, il vettore è un insieme di cassettini, indipendenti l'uno dall'altro, che hanno la possibilità di contenere un valore.
Si usa di solito quando si ha a che fare con molti valori appartenenti alla stessa tipologia di oggetti.

Il precedente programma del Semaforo con le costanti

Vedremo adesso che con l'uso prudente delle costanti la realizzazione del semaforo visto in precedenza risulterà molto più semplice dal punto di vista del software.
Approntiamo nuovamente lo schema della figura 11 presentato nella puntata precedente (FE 237 - Marzo 2005). Scriviamo il programma necessario e poi commentiamo i punti salienti:

```
program Semaforo2

const VERDE =  4 '00000100
const GIALLO = 6 '00000110
const ROSSO  = 1 '00000001

main:
  trisb=0 'Portb OUT
  portb=0 'Spegniamo tutto

  inizio:   'label del ciclo
    portb=VERDE 'Accende verde
    Delay_ms(30000) 'Wait 30 sec.
    portb=GIALLO 'Verde+giallo
    Delay_ms(5000) 'Wait 5 sec.
    portb=ROSSO 'Accende rosso
    Delay_ms(30000) 'Wait 30 sec.
  goto inizio 'Ricomincia tutto
end.
```

Abbiamo utilizzato tre costanti, VERDE, GIALLO, ROSSO. Rispettivamente contengono la notazione binaria della visualizzazione dei led, cioè:

- VERDE vale 4 ed in binario 100 (Verde ON, Giallo OFF, Rosso OFF).
- GIALLO vale 6 ed in binario 110 (Verde ON, Giallo ON, Rosso OFF).
- ROSSO vale 1 ed in binario 001 (Verde OFF, Giallo OFF, Rosso ON).

Le tre costanti vengono quindi rappresentate direttamente dalla PORTB (ricordiamo che questa è un byte a tutti gli effetti), opportunamente ritardate dal comando delay_ms. Il ciclo si ripete all'infinito grazie all'istruzione *goto inizio*.
Bene, proponiamo un ultimo esempio, prima di approfondire altri argomenti.
Analizziamo il seguente problema: abbiamo due diodi LED, rispettivamente di colore rosso e verde, collegati indipendentemente a due

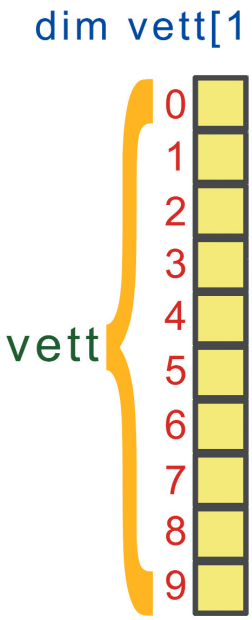


Figura 2 Struttura di un vettore

uscite della PORTB. Alla cadenza di 2 Hz deve lampeggiare il diodo rosso **per dieci volte**, poi alla cadenza di 1 Hz deve lampeggiare il diodo verde **per due volte**. Il ciclo si ripete all'infinito. Realizzate nuovamente il circuito di figura 9 presentato nella puntata precedente (FE 237 - Marzo 2005) rispettando i colori dei diodi LED. Il programma (una delle tante soluzioni possibili) sarà il seguente:

```
program RossoVerde

dim conta as byte

main:
  'Un altro modo di
  'esprimere un valore in
  'codice binario
  'digit per digit

  trisb=%00000000 'PORTB in OUTPUT
  portb=%00000000 'PORTB tutta spenta

  inizio:  'Etichetta di riferimento

  for conta=1 to 10
    portb.0=1 'Rosso ON
    Delay_ms(250)
    portb.0=0 'Rosso OFF
    Delay_ms(250)
  next conta

  for conta=1 to 2
    portb.1=1 'Verde ON
    Delay_ms(500)
    portb.1=0 'Verde OFF
    Delay_ms(500)
  next conta

  goto inizio

end.
```

Nel listato c'è una novità: nell'assegnare il valore 0 ai registri TRIS e PORT si è utilizzato l'operatore %. Collocato all'inizio di un valore, permette la sua codifica in formato binario.

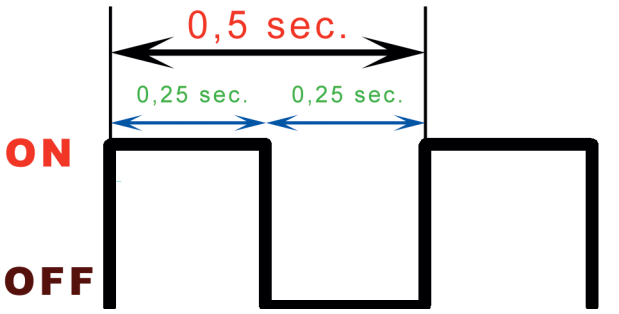


Figura 3 Temporizzazione di un lampeggio

Certe volte è usato per abilitare in un solo colpo più uscite. Se ad esempio volessimo accendere (partendo da RB0) la prima, la seconda, la terza e l'ultima porta di PORTB, basterà programmare la porta con la semplice istruzione:

```
PORTB = %10000111
```

RB0 è posto sempre alla destra della cifra binaria (bit meno significativo) mentre RB7 è posto alla sinistra della cifra (bit più significativo).
Analizziamo adesso il problema della temporizzazione. Se il diodo LED rosso deve lampeggiare alla cadenza di 2 Hz, perché abbiamo usato un tempo di attesa di 250 ms. e se il diodo LED verde deve lampeggiare alla cadenza di 1 Hz, perché abbiamo usato un tempo di attesa di 500 ms? Guardiamo la figura 3.
Quando la frequenza, o cadenza, di un lampeggio è, ad esempio, 2 Hz vuol dire che esso si ripete 2 volte al secondo. Pertanto la durata del suo periodo, cioè il tempo che intercorre tra l'inizio dell'evento e l'inizio del successivo evento ammonta a 0,5 secondi. La relazione è data dalla formula $t = 1/f$, dove t è il tempo in secondi ed f è la frequenza. Dal momento che l'intero ciclo dura 0,5 secondi è ovvio che un semiciclo (rappresentato dal led acceso) dura 0,25 secondi (cioè 250 ms.) e l'altro semiciclo (rappresentato dal led spento) dura anche 0,25 secondi.

E ADESSO USIAMO GLI INGRESSI

Sino ad ora abbiamo esaminato solamente il modo in cui il PIC comunica con il mondo

esterno. Adesso vedremo il contrario, cioè come è il mondo esterno a comunicare con il PIC. Si considerano esterni tutti quegli eventi provenienti da fonti non appartenenti alla circuiteria del microcontrollore stesso, come ad esempio l'impulso proveniente da un interruttore o da un pulsante, il comando proveniente da un sensore, le cifre generate da una tastiera esterna, e così via.
Occorre prima di tutto puntualizzare un concetto importantissimo: dal momento che il PIC lavora con tensioni logiche positive (0 Volt e 5 Volt rispetto massa), occorre che anche i segnali provenienti dall'esterno siano adeguati a questo range. Se da un pulsante preleviamo una tensione di 15 Volt, non dovremmo assolutamente applicarla al controllore, pena la sua distruzione.
Occorre quindi che tutti i livelli superiori a 5 Volt siano "accuratamente" adeguati al limite massimo sopportato dal PIC (+/- 10% di tolleranza massima). Sarà utile a tale scopo usare i partitori resistivi affinché la tensione di ingresso si abbassi al valore consentito.
Per il resto si segue la solita storia: un livello in entrata messo a 0 volt rispetto massa sarà considerato dal PIC come un segnale *basso* o *falso* (0), viceversa un segnale di 5 Volt sarà interpretato come un segnale logico *alto* o *vero*.
Consideriamo un altro fatto importante. Tutti gli ingressi devono essere comunicati al PIC in quanto tali. Occorre a tale scopo programmare in anticipo la funzione di ogni pin, ossia se esso fungerà da input oppure da ouput.
Se ad esempio su RB0 colleghiamo un LED e su RB1 colleghiamo un pulsante, in fase di programmazione dobbiamo comunicare al compilatore (attraverso il registro TRIS) che il pin RB0 è un'uscita e quindi un Output (valore **0**) ed il pin RB1 un ingresso e quindi un Input (valore **1**).
Passiamo subito alla pratica.

Occorre creare il seguente prototipo: su RB0 c'è collegato un LED rosso, su RB1 un pulsante normalmente aperto e su RB2 un altro pulsante normalmente aperto. Il circuito deve comportarsi nel modo seguente:

- All'accensione del circuito entrambi i LED sono spenti.
- Se si preme il primo pulsante (su RB1) il LED dovrà illuminarsi e dovrà mantenere tale stato anche al rilascio del pulsante stesso.
- Se si preme il secondo pulsante (su RB2) il LED dovrà spegnersi e dovrà mantenere tale stato anche al rilascio del pulsante stesso.

Non preoccupatevi, è più difficile da leggere che da realizzare. Realizzate lo schema della figura 4, poi passiamo ai commenti elettronici ed informatici. Guardando lo schema si nota che il Led è collegato, tramite la solita resistenza di limitazione di corrente, al pin RB0; quest'ultimo sarà quindi considerato come Output. Sui pin RB1 e RB2 sono collegati invece due pulsanti normalmente aperti.
Ecco il funzionamento del collegamento: se un pulsante è aperto (cioè non premuto) esso non si collega ai 5 Volt dell'alimentazione, pertanto RB1 (o RB2) sarà connesso a massa tramite la resistenza di 1K (resistenza di *pull-down*). In altre parole tale resistenza forza lo stato logico basso (0V) all'ingresso RB1 (o RB2). Viceversa, quando il pulsante è premuto, il pin RB1 (o RB2) si connetterà direttamente ai 5 Volt dell'alimentazione. Il pin passerà pertanto a livello 1 logico e la

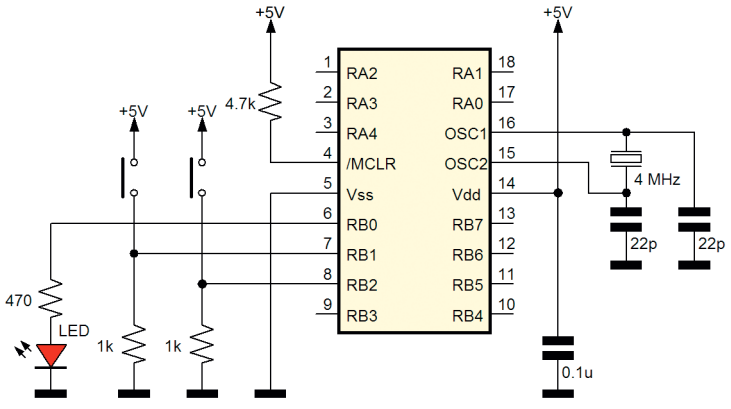


Figura 4 *Schema con un LED e due pulsanti (I/O)*

resistenza di pull-down non costituirà minimamente un problema, in quanto su essa scorrerà una corrente di soli 5 mA (essendo l'ingresso stesso ad alta impedenza). Volendo "consumare" meno corrente, potremo benissimo "alzare" il valore di queste resistenze fino a 47 Kohm senza problemi. L'importante è che ci siano, altrimenti, data l'alta impedenza dei suoi terminali, il PIC potrebbe leggere e recepire il segnale logico in maniera inaspettata e ambigua (ossia presenza di 1 e 0 senza apparenti spiegazioni "logiche"). Ecco come il PIC riconosce gli eventi esterni: con segnali logici 0-1 che corrispondono a due livelli analogici 0 Volt e 5 Volt. Il problema sarà adesso quello di istruire il microcontrollore a ricevere questi segnali, ad elaborarli e a comportarsi di conseguenza.
Studiamo il relativo programma, leggendo attentamente anche i commenti:

```
program Input1

main:
   'definiamo RB7-6-5-4-3
   'come uscite, RB2 e RB1
   'come entrate e RB0 come
   'uscita. Comunque le porte
   'da RB7 a RB3 in questo
   'esempio restano inutilizzate

   trisb=%00000110

   'In maniera equivalente
   'potremo scrivere:
   'trisb.0=0 (output)
   'trisb.1=1 (input)
   'trisb.2=1 (input)

   'Azzeriamo adesso la SOLA
   'porta in OUTPUT (RB0)
   'Non settiamo MAI da
   'software i pin configurati
   'come input. Potrebbe essere
   'pericoloso (per il PIC).

   portb.0=0 'Spegniamo LED
   ciclo:    'Label di iterazione

     if portb.1=1 then 'Se premo pulsante 1
       portb.0=1       'accende LED
     end if

     if portb.2=1 then 'Se premo pulsante 2
       portb.0=0       'spegne LED
     end if

   goto ciclo

end.
```

Come si vede, il listato è molto breve, abbiamo prolungato un po' il commento, cioè la parte descrittiva del programma che il compilatore non considera durante la traduzione in linguaggio macchina.
Vediamo come si comporta il software: dopo le definizioni di rito che comandano il funzionamento della porta e il suo azzeramento, si passa direttamente ad esaminare lo **stato logico** delle due porte-ingresso.
Questa operazione viene svolta dal comando IF-THEN che in italiano vuol dire SE-ALLORA. Il micro inizialmente si domanda SE lo stato della RB1 (dove è montato il primo pulsante) si trova a livello logico alto (1). Se così è, viene eseguita subito la successiva istruzione (portb.0=1) cioè effettua l'accensione del led su RB0 (o portb.0, che poi è la stessa cosa). Se invece il livello su RB1 è basso (pulsante non premuto) la successiva istruzione (sempre portb.0=1) non viene minimamente considerata e quindi non è eseguita.
Continuando, il micro si pone usa seconda domanda, cioè se lo stato logico presente su portb.2 (RB2) è alto (pulsante premuto). Se così è, viene eseguita la successiva istruzione (portb.0=0) e il LED si spegne sino ad altra comunicazione. Con il comando *goto ciclo*, l'intero processo si ripete all'infinito.
Abbiamo realizzato una sorta di switch digitale che prevede l'accensione e lo spegnimento mediante due pulsanti indipendenti, adibiti esclusivamente a questo scopo.

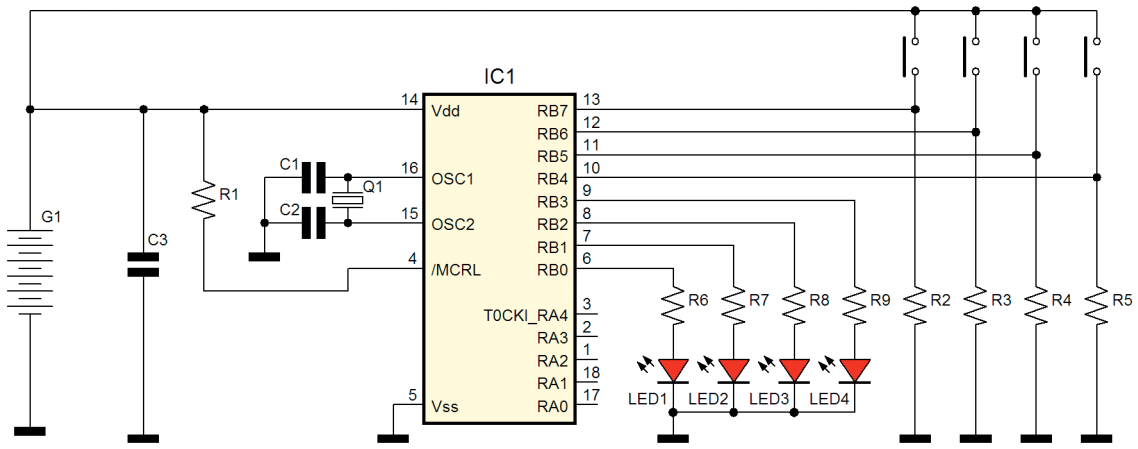


Figura 5 Schema elettrico del progetto con 4 pulsanti e 4 LED

Il solo modo di imparare a programmare un microcontrollore è quello di fare pratica su pratica. Inventate quindi da voi gli esempi e gli esperimenti più disparati, non abbiate paura di sbagliare: nel caso peggiore brucereste qualche componente elettronico di pochi euro. Solo così acquisirete l'esperienza necessaria. Non limitatevi in ogni caso ai soli esempi riportati in questi articoli, ma liberate la fantasia alla ricerca di altri espedienti o addirittura ad altre soluzioni.
Per proporre molti esempi occorrerebbero migliaia di pagine e di illustrazioni, e risulterebbero sempre insufficienti.
Ecco un altro esempio da provare, questa volta un po' più complicato.
Ecco il problema da risolvere:

- Si hanno quattro pulsanti e 4 diodi LED.
- I 4 LED saranno collegati su RB0, RB1, RB2, RB3 (Output).
- I 4 pulsanti (normalmente aperti) saranno collegati su RB4, RB5, RB6, RB7 (Input).
- Inizialmente i 4 LED saranno spenti.
- Se l'utente non preme alcun tasto non succede nulla.
- Se si preme il primo pulsante (su RB4) i quattro LED lampeggeranno per cinque volte (5) alla frequenza di 1 Hz, poi si spegneranno.
- Se si preme il secondo pulsante (su RB5) i quattro LED si accenderanno in sequenza per un secondo (1 secondo il primo, un secondo il secondo, un secondo il terzo e un secondo il quarto).
- Se si preme il terzo pulsante (su RB6) i quattro LED staranno accesi per tre secondi, poi si spegneranno.
- Se si preme infine il quarto pulsante (su RB7) i quattro LED lampeggeranno per cinquanta volte (50) alla frequenza di 100 millisecondi.
- Il software deve essere ciclico ed infinito, cioè una volta terminato un evento, aspetterà il verificarsi del successivo.

Abbiamo complicato veramente le cose: il programma in questione è proprio contorto, richiede una visione completa del problema, ma non preoccupatevi. Cercate di compenetrarvi nell'analisi della questione e vedrete che pian piano troverete anche voi la soluzione. Gli articoli di questa rubrica non vanno letti come un romanzo o un libro giallo. Al contrario essi vanno studiati attentamente. Ma con la passione e la costanza si disporrà sicuramente di una marcia in più.

Bisogna realizzare un prototipo costituito da 4 ingressi e 4 uscite. Si può tranquillamente usare una basetta millefori, per minimizzare le operazioni di realizzazione. Seguiamo lo schema di figura 5.
Scorgiamo subito i quattro Led collegati alla

RB0-1-2-3 e i pulsanti collegati alla RB4-5-6-7. Si notino le resistenze di pull-down che forzano gli ingressi del Pic a livello logico basso per i pulsanti non premuti, mentre in caso contrario gli stessi assumeranno il valore logico alto, proprio perché collegati direttamente all'alimentazione. Le resistenze di limitazione dei diodi Led sono di 220 ohm, che consentono di alimentare i Led stessi con una corrente di (5-2)/220 = 13,6 mA circa, corrente che garantirà una corretta illuminazione.

Andiamo ora ad esaminare il diagramma di flusso proposto in figura 6. L'insieme dei compiti che svolge il microcontrollore ed il metodo usato per risolvere i vari problemi è definito **algoritmo**.

Digitiamo il programma cercando di non commettere errori, in seguito lo commenteremo:

```
program QuattroQuattro

dim k as byte

main:

  portb=0 'Azzera PORTB prima

  'Definisce RB7,RB6,RB5,RB4 come INPUT e
  'RB3,RB2,RB1,RB0 come OUTPUT
  trisb=%11110000  'Definizione PORTB

  while true        'Inizio Ciclo INFINITO
    '-----Se si preme tasto 1----
    if portb.4=1 then
      for k=1 to 5
        portb.0=1 'Led 1 ON
        portb.1=1 'Led 2 ON
        portb.2=1 'Led 3 ON
        portb.3=1 'Led 4 ON
        delay_ms(500)
        portb.0=0 'Led 1 OFF
        portb.1=0 'Led 2 OFF
        portb.2=0 'Led 3 OFF
        portb.3=0 'Led 4 OFF
        delay_ms(500)
      next k
    end if
    '-----Se si preme tasto 2----
    if portb.5=1 then
      portb.0=1 'Led 1 ON
      delay_ms(1000)
      portb.0=0 'Led 1 OFF
      portb.1=1 'Led 2 ON
      delay_ms(1000)
      portb.1=0 'Led 2 OFF
      portb.2=1 'Led 3 ON
      delay_ms(1000)
      portb.2=0 'Led 3 OFF
      portb.3=1 'Led 4 ON
      delay_ms(1000)
      portb.3=0 'Led 4 OFF
    end if
    '-----Se si preme tasto 3----
    if portb.6=1 then
      portb.0=1 'Led 1 ON
      portb.1=1 'Led 2 ON
      portb.2=1 'Led 3 ON
      portb.3=1 'Led 4 ON
      delay_ms(3000)
      portb.0=0 'Led 1 OFF
      portb.1=0 'Led 2 OFF
      portb.2=0 'Led 3 OFF
      portb.3=0 'Led 4 OFF
    end if
    '-----Se si preme tasto 4----
    if portb.7=1 then
      for k=1 to 50
        portb.0=1 'Led 1 ON
        portb.1=1 'Led 2 ON
        portb.2=1 'Led 3 ON
        portb.3=1 'Led 4 ON
        delay_ms(50)
        portb.0=0 'Led 1 OFF
        portb.1=0 'Led 2 OFF
        portb.2=0 'Led 3 OFF
        portb.3=0 'Led 4 OFF
        delay_ms(50)
      next k
    end if
    '-------------------------
  wend  'Ritorna al ciclo infinito
end.
```

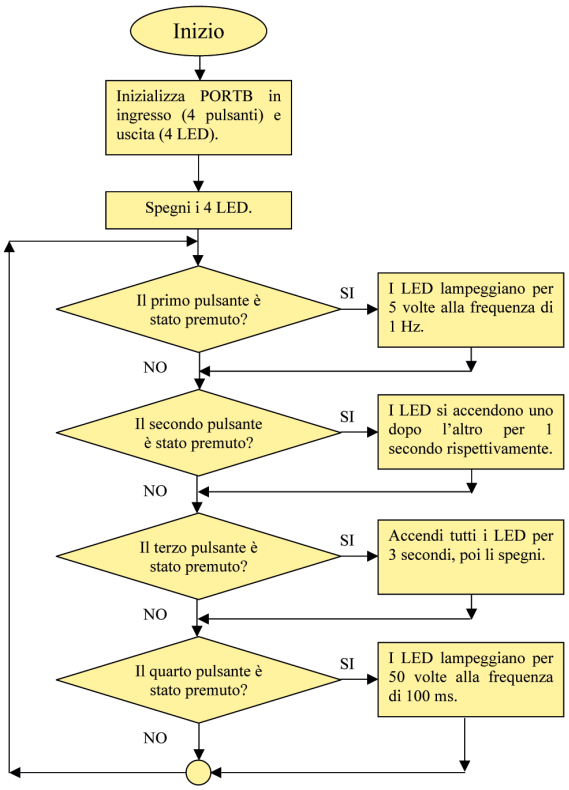


Figura 6 *Flow Chart del progetto con 4 pulsanti e 4 LED*

Il listato è molto chiaro. È stato inserito un nuovo comando, o meglio, una nuova struttura: il blocco di codice *while - wend*. Esso consente di migliorare la leggibilità del programma e aggiunge eleganza alla struttura stessa. In pratica, tutto il codice che è racchiuso all'interno di queste due istruzioni, viene eseguito sino a quando la condizione specificata dopo il comando *while* risulta vera. Dal momento che, come condizione, abbiamo messo la parola riservata *true* (vero), il ciclo si ripeterà all'infinito. Il funzionamento è simile all'uso della label con l'istruzione goto;
Qui la label è sostituita dal *while*, mentre il *goto* è sostituito dal wend. Prendete però con le pinze questo paragone: è solo per raffrontare i due metodi, ma in realtà si tratta di comandi completamente diversi.
Con questo esempio abbiamo realizzato un completo sistema automatizzato: un insieme di periferiche in uscita che è costituito dai diodi Led, un insieme di periferiche indipendenti in entrata, costituito dai pulsanti, un sistema di lettura eventi esterni, un blocco di decisione ed un blocco di esecuzione.
Anche i software più complessi (lo vedremo nelle prossime puntate) saranno basati sempre su questi principi basilari che governano i sistemi automatizzati.
A questo punto provate tranquillamente il prototipo creato, e se avete seguito attentamente gli appunti, avendo trascritto correttamente anche il codice, noterete, con grande soddisfazione, che il programma si comporta egregiamente eseguendo ciecamente i comandi impartiti dall'uomo. I comandi da pulsante possono essere ripetuti diverse volte. Il PIC eseguirà il suo lavoro nel migliore dei modi.
Una nota particolare: durante l'esecuzione di un evento dei quattro previsti (parliamo dei vari lampeggi) i tasti non risponderanno ai comandi. Mentre cioè sta verificandosi un lampeggio, anche se si preme uno dei quattro pulsanti, non accadrà nulla.
Questo avviene perché il controllo del software è vincolato all'esecuzione dei comandi in un ciclo "bloccato" e, in teoria, mentre si esegue un processo non ne può avvenire un altro (multitasking).

PROGETTO ILLUMINAZIONE SCALA

Il prossimo progetto è quello che prevede l'illuminazione di un garage o di una scala di un condominio. Esso è realizzato in versione ridotta e a 12 Volt.
Tuttavia è estremamente didattico poiché ripropone tutte le fasi teoriche, pratiche e realizzative che devono essere seguite.
Iniziamo subito dall'analisi del problema da risolvere, e quindi all'ideazione dell'algoritmo.

Analisi del problema

In un scala condominiale solitamente le luci sono spente, per un motivo prettamente economico. In prossimità degli interruttori restano però illuminate delle piccole lampadine che anche al buio danno la possibilità di orientarsi e di trovare subito il pulsante.
Se a questo punto quest'ultimo viene premuto,

si accendono immediatamente le luci della scala, per un periodo di tempo deciso in sede condominiale. Ma contemporaneamente le piccole lucine di orientamento si spengono, essendo ormai del tutto inutile il loro funzionamento. Dopo questo breve tempo di "attività" del circuito, le luci scala si spengono nuovamente e si riaccendono le piccole lampadine di orientamento; il sistema resta quindi in attesa di un nuovo impulso del pulsante. Questo è un classico esempio di un programma a "ciclo infinito".
In una delle prossime puntate proporremo la versione più sofisticata ed "intelligente" del progetto, che prevede anche lo spegnersi delle lampade scala in maniera graduale.

Hardware necessario

Il materiale necessario è ridotto ai minimi termini, almeno in questa versione didattica. Al circuito di pilotaggio vengono aggiunte le seguenti componenti aggiuntive:

- Un pulsante n/a collegato a RB0.
- Un diodo LED collegato a RB1.
- Una lampada di potenza a 12 Volt collegata a RB2 (e pilotata naturalmente da un transistor di potenza).

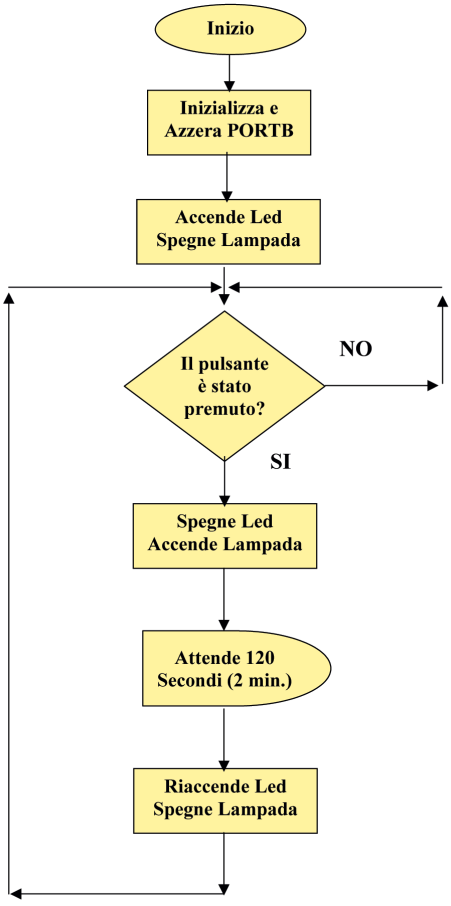


Figura 7 Flow Chart del progetto Illuminazione Scala

Algoritmo generale

Le fasi che il microcontrollore dovrà gestire per eseguire il software programmato sono le seguenti:

1. Inizializza ed azzera le porte.
2. Accende piccola lampadina di orientamento e spegne luci scala.
3. Se l'uomo preme il pulsante:
   a. Accende luci scala di potenza
   b. Spegne lampadina di orientamento (rappresentata dal LED)
   c. Mantiene tale stato per 2 minuti
   d. Spegne luci scala di potenza
   e. Accende lampadina di orientamento
4. Ritorna al punto 3 (ciclo infinito).

Rappresentazione mediante Flow Chart

Rappresentiamo in figura 7 lo schema dell'algoritmo in formato grafico, utilizzando i blocchi standard per i diagrammi di flusso.
Lo scopo di questo diagramma è quello di visualizzare schematicamente ed immediatamente il "ragionamento" che seguirà il PIC.

Il listato Basic

Il seguente listato Basic è molto semplice da capire. Eccolo di seguito:

```
program Scala

dim k as byte

main:
  portb=0 'Azzera PORTB

  trisb.0=1 'RB0 in Input
  trisb.1=0 'RB1 in Output
```

```
  trisb.2=0 'RB2 in Output

  portb.1=1 'Accende LED
  portb.2=0 'Spegne lampada

  while true
    if portb.0=1 then  'Se premo pulsante
      portb.2=1 'Accende Lampada
      portb.1=0 'Spegne Led
      '---Attesa---
      for k=1 to 120 'Secondi
        delay_ms(1000) 'Attende un secondo
      next k
      '---------------
      portb.2=0 'Spegne lampada
      portb.1=1 'Riaccende Led
    end if
  wend

end.
```

I commenti del software si spiegano da soli. Si deve approfondire il discorso della temporizzazione dei 120 secondi. Potremmo in teoria usare l'istruzione *delay_ms*(120000) per attendere 120 secondi. Purtroppo tale funzione built-in accetta solo valori in formato *word* (2 byte). Accetterebbe, almeno in teoria, un numero massimo di 65535 millisecondi (65 secondi).
Superiamo allora l'ostacolo facendo ripetere al programma, per 120 volte (for-next), l'attesa di un secondo, e in definitiva otteniamo in maniera precisa i due minuti necessari.
Usando tale metodo si possono effettuare temporizzazioni lunghissime, anche di centinaia di anni...

Schema elettrico

In figura 8 possiamo vedere lo schema elettrico, peraltro molto semplice. Vi sono questa volta due sorgenti di alimentazione: i 5V per alimentare il microcontrollore e i 12V per alimentare la lampada di potenza. Ecco il funzionamento. Se il pulsante (normalmente aperto) non viene premuto, la porta RB0 assumerà valore logico basso in quanto collegata a massa attraverso la resistenza di pull-down da 1 Kohm.
Non appena viene premuto il pulsante, il piedino RB0 assume subito il valore logico alto, poiché avviene il collegamento diretto con i 5 Volt di alimentazione. Il programma si accorgerà subito di questo cambio di stato e, attraverso la nostra routine appositamente creata, si comporterà di conseguenza.
Ricordiamo sempre che all'ingresso devono essere sempre applicati i 5 volt per il livello logico alto e 0 Volt per il livello logico basso. Non bisogna MAI superare questa soglia pena la distruzione del PIC.
Il software, accorgendosi del pulsante premuto, porrà a livello logico alto il piedino RB2, in modo che, attraverso la resistenza di polarizza-

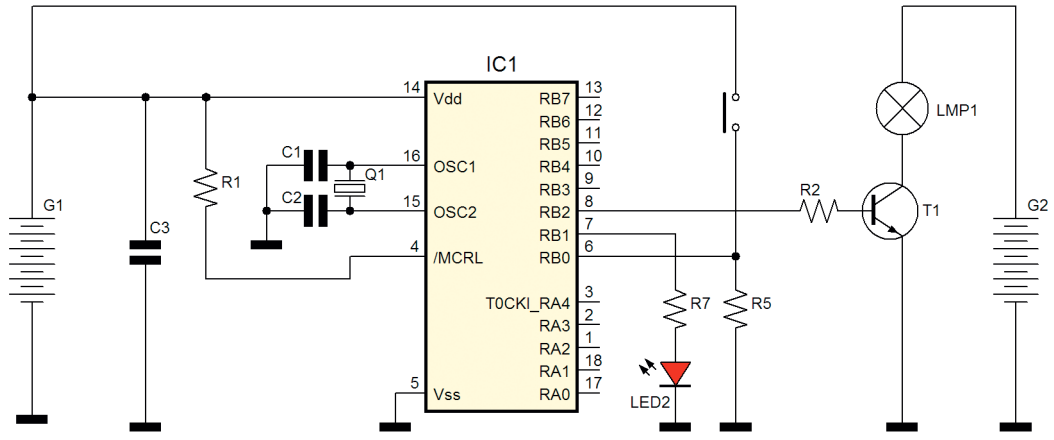


Figura 8 Schema elettrico del progetto Illuminazione Scala

zione di base del transistor di potenza BD243, scorra una corrente di circa 20 mA. Tale corrente è più che sufficiente a saturare il transistor (con Beta prossimo a 100) e porlo in conduzione diretta, quindi ad illuminare la lampada.
Occorre dire che il BD243 sopporta bene fino a 6 A di corrente, ma è opportuno dotarlo di apposita aletta di raffreddamento poiché, in ogni caso, scalda parecchio.
Chi volesse può facilmente cambiare la durata di accensione della lampada.
Basta semplicemente sostituire il valore 120 (nel comando *for k=1 to 120*) con altro valore desiderato (ricordiamo però che la variabile k è definita come tipo ***byte***, quindi accetta al massimo un valore di 255).

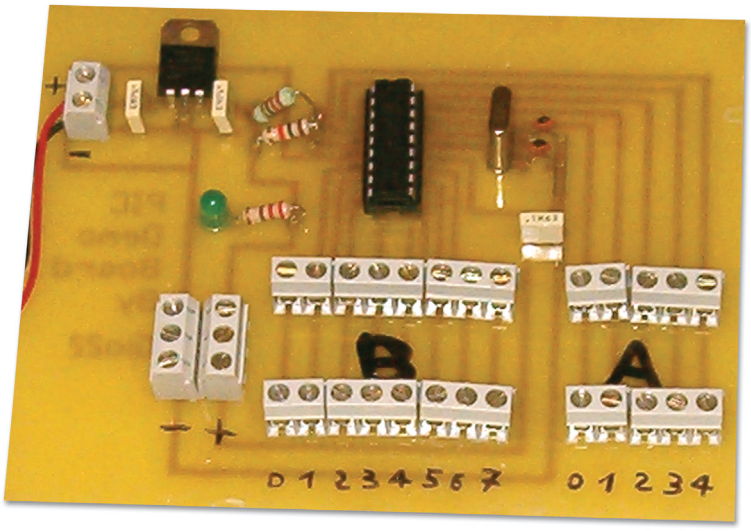


Figura 9 *L'utilissimo prototipo del Pic Test Board*

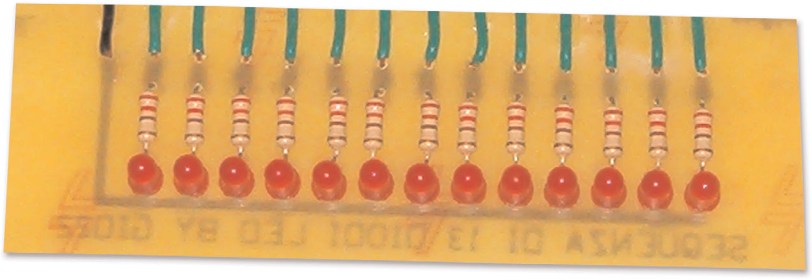


Figura 10 *La barra dei LED per testare i programmi*

MIKROBASIC SUL WEB

Trovate il compilatore mikrobasic sul sito http://www.mikroelektronika.co.yu, naturalmente in versione limitata (il limite consiste nella massima estensione del codice generato). Potete scaricarlo liberamente ed installarlo subito sul vostro PC.
Consigliamo anche di scaricare e di stampare il manuale utente che, anche se in lingua inglese, dà moltissime informazioni utili sul suo utilizzo. Per ciò che concerne l'hardware di trasferimento del codice eseguibile (Hex) sul PIC è necessario un programmatore USB, seriale o parallelo connesso al PC.
Sui precedenti numeri trovate anche molti progetti a riguardo.
Per quanto riguarda il software di trasferimento consigliamo l'utilizzo del famoso *IC-Prog*, scaricabile liberamente da Internet e rintracciabile un po' ovunque sui siti.

ESERCITAZIONI

Dal momento che la pratica è la migliore maestra, perché non provare a realizzare "da soli" qualche piccola esercitazione?

Semaforo di un incrocio stradale: provate a realizzare un semaforo che preveda il controllo del traffico in un incrocio stradale. Pertanto i semafori devono essere due: uno per ciascuna strada. Il funzionamento deve essere il seguente:

- Semaforo A rosso, semaforo B verde.
- Semaforo A rosso, semaforo B verde + giallo.
- Semaforo A verde, semaforo B rosso.
- Semaforo A verde + giallo, semaforo B rosso.
- Ripete il ciclo all'infinito.

Usate una temporizzazione sufficiente per gestire i due eventi.

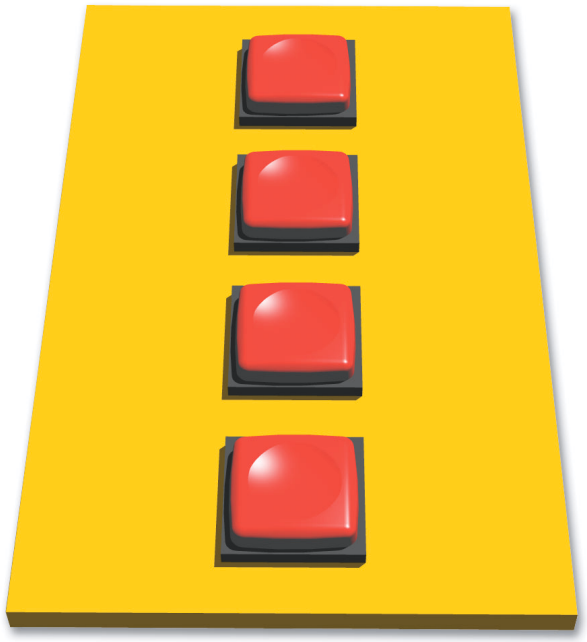


Figura 11 La pulsantiera per testare i programmi

COSA BOLLE IN PENTOLA

Non perdete assolutamente i prossimi numeri della rivista. Se questo articolo, unito al precedente, costituisce solo l'assaggio, il bello deve ancora arrivare.
Nelle parti successive saranno approfondite le seguenti problematiche: utilizzare al meglio le uscite del PIC, utilizzare al meglio gli ingressi del PIC, la teoria del PWM, facciamo musica col microcontrollore, utilizzo dei motori passo-passo, la memorizzazione dei dati, le conversioni analogico digitali, realizzazioni pratiche e divertenti per tutti.

CONCLUSIONI

Il PIC non serve "solamente" per accendere o spegnere led e lampade, come abbiamo visto fino ad ora. In questi primi due articoli abbiamo soltanto mosso i primi passi nella programmazione dei microcontrollori.
Gli esperimenti fin qui condotti hanno avuto il carattere della dimostrazione e della prova, abbiamo trattato la materia con semplicità senza spingerci in ragionamenti troppo tecnici, almeno per il momento.
Nei prossimi numeri saranno trattati, con il dovuto approfondimento, gli argomenti che permetteranno di allargare il proprio bagaglio culturale sulla conoscenza dei PIC, ma soprattutto verranno presi in considerazione gli aspetti pratici di utilizzo che quotidianamente si ha l'esigenza di soddisfare.

Alle prossime puntate.

Voltage on V_{DD} with respect to V_{SS}	-0.3 to +7.5V
Voltage on $\overline{MCLR}$ with respect to V_{SS} (1)	-0.3 to +14V
Voltage on RA4 with respect to V_{SS}	-0.3 to +8.5V
Total power dissipation (2)	800 mW
Maximum current out of V_{SS} pin	150 mA
Maximum current into V_{DD} pin	100 mA
Input clamp current, I_{IK} (V_I < 0 or V_I > V_{DD})	± 20 mA
Output clamp current, I_{OK} (V_O < 0 or V_O > V_{DD})	± 20 mA
Maximum output current sunk by any I/O pin	25 mA
Maximum output current sourced by any I/O pin	25 mA
Maximum current sunk by PORTA	80 mA
Maximum current sourced by PORTA	50 mA
Maximum current sunk by PORTB	150 mA
Maximum current sourced by PORTB	100 mA

Figura 12 Valori massimi elettrici sopportati dal PIC (dal Datasheet della Microchip)

Seconda parte
Le variabili, le costanti e gli ingressi

Terza parte
Utilizzare al meglio le uscite del PIC

Quarta parte
Utilizzare al meglio gli ingressi del PIC

Mikrobasic per PICmicro

Proseguiamo lo studio del Mikrobasic approfondendo l'utilizzo delle uscite dei PICmico. Tramite apposite routine in linguaggio Basic si possono pilotare vari dispositivi collegati alle porte del PIC16F84, configurate stavolta come unità di output. Ma soprattutto capiremo come, la maggior parte delle applicazioni, non sia solamente frutto di acrobazie elettroniche ma di capacità logiche ed organizzative per la creazione di un codice efficiente e funzionale.

Il microcontrollore PIC16F84 (come abbiamo visto nella puntata precedente) dispone di ben 13 terminali (pin) adibiti a porte di comunicazione di ingresso e uscita. Sappiamo anche che ogni terminale (da RB0 a RB7 e da RA0 a RA4) ha un funzionamento digitale, nel senso che il suo stato logico può essere solamente vero (1) o falso (0). Parlando in termini di segnale analogico, si può affermare che ogni pin configurato come porta in uscita può erogare una tensione di 5 Volt (rispetto a massa) se posto a livello alto e 0 Volt (sempre rispetto a massa) se posto a livello basso.
Abbiamo studiato che la scelta della funzione di ogni pin è effettuata tramite l'uso del registro TRIS mentre la scelta dei livelli di uscita viene svolta attraverso il registro PORT. Invitiamo pertanto a rileggere il precedente articolo, in quanto molto importante come primo approccio all'uso dei microcontrollori.
Leggendo il datasheet del 16F84, possiamo annotare i parametri massimi assoluti che il programmatore deve scrupolosamente seguire, pena la distruzione, anche immediata, del microcontrollore. Tali parametri riguardano soprattutto la corrente massima che può essere fatta scorrere da ogni pin in uscita e, quindi, dall'intero integrato.
La dissipazione totale non DEVE superare gli 800mW di potenza, ogni pin non deve erogare più di 25 mA e, in ogni caso, la completa PORT-A deve erogare al massimo 50 mA mentre la PORT-B deve poter fornire 100 mA massimi.
Ci rendiamo subito conto di queste piccole "limitazioni", peraltro facilmente superabili. Collegando alla PORT-B ben 8 diodi LED (dal consumo di circa 20 mA), riusciamo tranquillamente a restare nei limiti della tolleranza per quel che riguarda la corrente assorbita da ogni singolo pin (25 mA massimi previsti contro 20 mA effettivamente richiesti dal LED), ma oltrepasseremo il limite massimo imposto dalla PORT-B (100 mA previsti contro 160 mA "usati" dai diodi LED).
In ogni modo non dobbiamo mai raggiungere questi livelli di soglia, ma cerchiamo di far

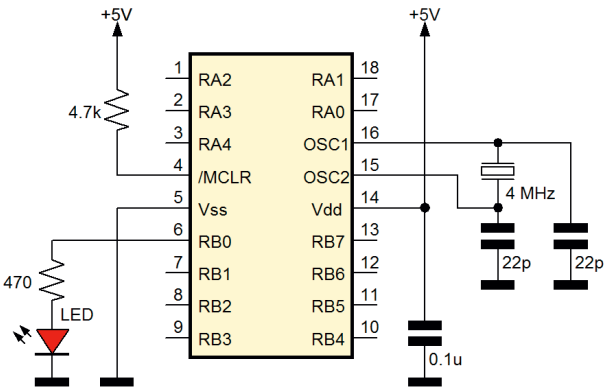


Figura 1 Schema elettrico del lampeggiatore

Utilizzare al meglio le uscite del PIC

di Giovanni Di Maria
g.dimaria@farelettronica.com



lavorare con tranquillità il nostro PIC, senza farlo eccessivamente "sudare". Per fortuna esistono i transistor che si caricano di compiti ben più pesanti: come dire, il PIC è la mente e il transistor il braccio.

PRIMO LAMPEGGIO

Iniziamo subito la puntata con una rielaborazione del lampeggiatore a diodo LED collegato ad una uscita del microcontrollore. Realizziamo per lo scopo lo schema proposto in figura 1.

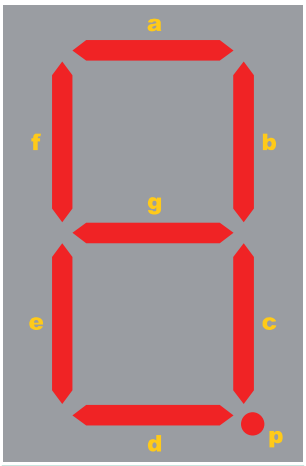


Figura 2 *I 7 segmenti di un display*

Scriviamo con attenzione il seguente programma:

```
program lampeggio
main:
    trisb.0=0   'RB0 come uscita
inizio:
    portb.0=1  'Accende LED
    Delay_ms(1000)'Wait 1 sec
    portb.0=0  'Spegne LED
    Delay_ms(1000)'Wait 1 sec
    goto inizio 'Ripete sempre
end.
```

Compiliamo il software e scarichiamo sul PIC il file HEX prodotto. Dando l'alimentazione al circuito si può notare che il diodo Led collegato su RB0 lampeggia senza sosta. Ogni stato logico (acceso e spento) ha la durata di 1 secondo.
Questo è naturalmente il minimo che si può pretendere da un microcontrollore. Il modo migliore per conoscere il PIC è quello di provare a scrivere da soli i programmi e le routine.

FINALMENTE IL DISPLAY A 7 SEGMENTI

Bene, vediamo adesso come pilotare un display a Led a 7 segmenti, utile per rappresentare una cifra decimale ed alcune lettere dell'alfabeto. Dal momento che un display a 7 segmenti è dotato di sette ingressi separati (ognuno dei quali è collegato ad un segmento luminoso), è ovvio che abbiamo bisogno di sette porte configurate come uscita per poter illuminare la barra desiderata.
In realtà il display dispone anche di un ottavo ingresso che è rappresentato dal punto decimale luminoso, ma questo è raramente utilizzato. Con due display la situazione diventa ancora più drammatica, poiché necessitano 14 porte; immaginate quindi se in un progetto occorrerebbero quattro o sei display: il caso sembrerebbe irrisolvibile.
Questo ostacolo è brillantemente superato attraverso l'artifizio di alcune tecniche utilizzate.

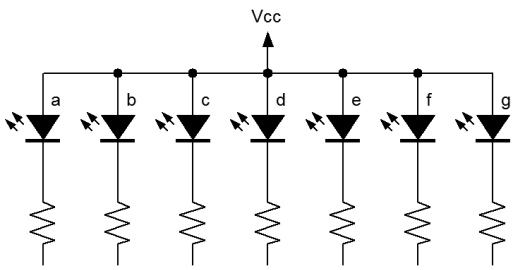


Figura 3 *Display ad Anodo Comune*

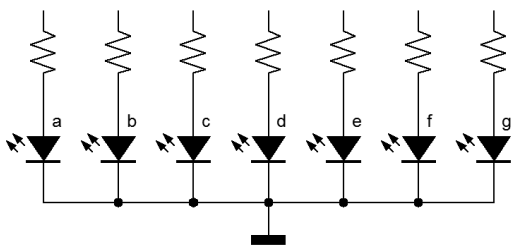


Figura 4 *Display a Catodo Comune*

Struttura

Un display è internamente composto da 7 diodi led, a forma di piccole barrette, disposti in modo da visualizzare un numero o una cifra.
Ogni segmento è "codificato" da una lettera dell'alfabeto, che ne determina anche il nome. In figura 2 vediamo la disposizione "ordinata" e standard dei 7 segmenti.

Elettricamente, il display altro non è che un insieme di 7 diodi Led. Se i Led hanno in comune l'anodo si dicono display ad *anodo comune*, viceversa vengono definiti display a *catodo comune*. Nelle figure 3 e 4 è rappresentato il relativo schema dei due tipi del display. Ogni segmento (che è un vero e proprio diodo led) deve essere attraversato da una corrente di circa 10-15 mA, per ottenerne la corretta illuminazione. La differenza di potenziale che troviamo attorno ai suoi capi ammonta a circa 1,5 Volt ed è da considerare nel calcolo della resistenza limitatrice di corrente.

Collegamento al PIC

Collegando un display a 7 segmenti al PIC, vengono utilizzate 7 porte del microcontrollore. Vediamo allora come si deve programmare il "micro" affinché piloti e visualizzi correttamente le cifre, secondo le nostre esigenze. Realizziamo lo schema di cui alla figura 5, poi passiamo alla codifica. Allo scopo si utilizzeranno i display a catodo comune, che sono i più usati.
La piedinatura è molto semplice, come si desume dalla disposizione dei pin, in figura 6 (vista dall'alto). I terminali centrali (contrassegnati dallo 0) sono i catodi e quindi vanno collegati alla massa comune del circuito.

Le resistenze di limitazione non sono critiche, l'importante è che siano attraversate da una corrente di circa 10-15 mA.
Calcolatene il valore con la seguente formula: **R = (5-1,5) / Ampere**. Nell'esempio abbiamo utilizzato un valore di 390 ohm.

L'obiettivo

Dobbiamo realizzare un contatore che visualizza, alla cadenza di un secondo, le cifre da 0 a 9. Alla fine il display si spegne.

Codifica dei 7 segmenti

Per visualizzare correttamente i numeri sul display, occorre "illuminare" i corrispondenti segmenti, precedentemente codificati con le lettere dalla A alla G.
Poiché ogni segmento è direttamente collegato alla PORT-B del Pic, il compito del software sarà anche quello di porre a livello logico alto il corrispondente piedino di uscita del micro.
Ecco in figura 7 una tabella riassuntiva che mostra la sequenza dei segmenti per ogni cifra, nonché il relativo valore binario corrispondente

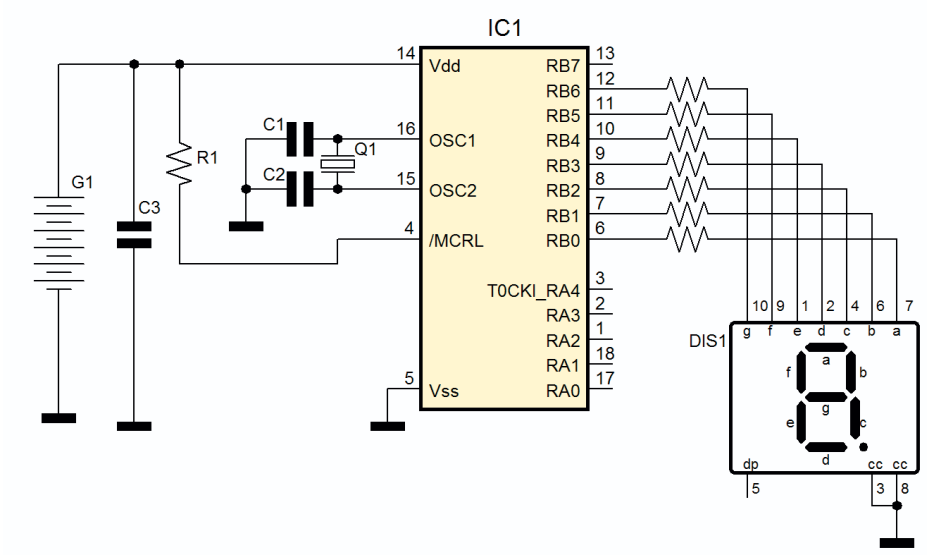


Figura 5 Schema con display a 7 segmenti

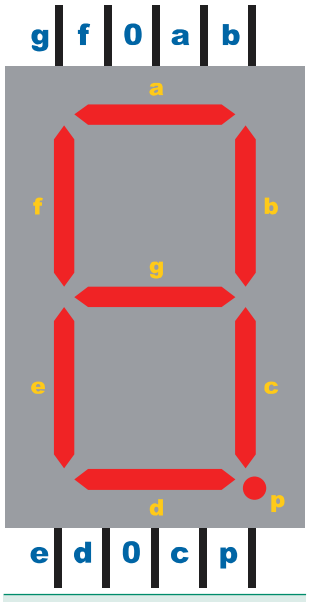


Figura 6 Piedinatura del display (vista dall'alto)

alla PORT-B. Nel compilatore Mikrobasic esiste una funzione di utilità che fornisce il valore binario e decimale della combinazione dei segmenti illuminati.

Display	Numero	Segmenti accesi	Porte del PIC a livello logico ALTO	Valore decimale PORTB
	0	A+B+C+ D+E+F	RB0, RB1, RB2, RB3, RB4, RB5	63
	1	B+C	RB1, RB2	6
	2	A+B+G+E+D	RB0, RB1, RB6, RB4, RB3	91
	3	A+B+G+C+D	RB0, RB1, RB6, RB2, RB3	79
	4	F+G+B+C	RB5, RB6, RB1, RB2	102
	5	A+F+G+C+D	RB0, RB5, RB6, RB2, RB3	109
	6	A+F+G+ C+D+E	RB0, RB5, RB6, RB2, RB3, RB4	125
	7	A+B+C	RB0, RB1, RB2	7
	8	A+B+C+D+ E+F+G	RB0, RB1, RB2, RB3, RB4, RB5, RB6	127
	9	A+B+F+ G+C+D	RB0, RB1, RB5, RB6, RB2, RB3	111

Figura 7 Tabella dei valori per la formazione delle cifre

Il software

Ecco il listato Basic (peraltro molto semplice) che commentiamo *[Listato 1]*:

Il programma inizia a memorizzare i dieci codici corrispondenti alle cifre da visualizzare (come da tabella 7). Si usano a tale scopo dieci costanti, chiamate *display[10]*, che contengono il corretto set di segmenti da visualizzare.
Dopo gli azzeramenti di rito della PORT-B e la sua definizione come porta di output, inizia un ciclo di conteggio da 0 a 9 (che evita di dover riscrivere per dieci volte la stessa routine).
Per ogni valore "sequenziale" contenuto nel vettore, e contrassegnato dall'indice k, si ha la corrispondente assegnazione alla PORT-B del relativo valore che, quindi, illumina i segmenti desiderati.
Tale accensione è effettuata alla cadenza di un secondo, grazie alla funzione *delay_ms*. La routine termina quindi con l'azzeramento totale della port-b ed il conseguente spegnimento di tutti i segmenti del display.

Commenti al progetto

Il prototipo funziona egregiamente. Abbiamo però dovuto sacrificare ben 7 (sette) pin della port-b (e non abbiamo utilizzato il punto decimale…).
Se necessitano 2, 3, 4 o più display in teoria occorrerebbero 14, 21, 28 o più terminali.
Vedremo nelle prossime sezioni come si risolve tale problema, usando naturalmente alcuni stratagemmi…

[Listato 1]

```
program seven_segment01
    const display as byte[10]= (63,6,91,79,102,109,125,7,127,111)
    dim k as byte
 main:
    portb=0 'Azzera portb
    trisb=0 'definisce PORTB in output
    portb=0 'Azzera portb
    for k=0 to 9
       portb=display[k] 'visualizza numero
       Delay_ms(1000)    'aspetta un secondo
    next k
    portb=0 'alla fine spegne Display
 end.
```

NUMERI PARI E DISPARI

L'obiettivo

Nel prossimo esperimento dobbiamo realizzare quanto segue: un display a 7 segmenti è inizialmente spento per 2 secondi.
Quindi, al ritmo di 1 secondo visualizza tutti i numeri PARI da 0 a 8. Dopo due secondi, durante i quali il display resta spento, vengono visualizzati tutti i numeri DISPARI da 1 a 9, sempre alla cadenza di un secondo. Infine il display si spegne definitivamente. Facciamo riferimento allo schema della figura 5.

Algoritmo

Tralasciando lo schema a blocchi, ecco di seguito una delle possibili soluzioni del problema, espressa in uno pseudo linguaggio:

- Inizio
- Memorizza codici LED
- Azzera PORTB
- Definisci come output la PORTB
- Attesa 2 secondi (OFF)
- Conta da 0 a 8 (a 2 a 2)
- Visualizza Numero
- Esegui fino a quando numero = 8
- Spegni Display
- Attesa 2 secondi (OFF)
- Conta da 1 a 9 (a 2 a 2)
- Visualizza Numero
- Esegui fino a quando numero = 9
- Spegni Display
- Fine

Il software

Di seguito il listato Basic. Leggetelo e studiatelo molto attentamente *[Listato 2]*.

Escludendo commenti superflui, salta subito all'occhio una grande novità: il comando **step 2** posto subito dopo la clausola *for*.
Questa aggiunta al ciclo di conteggio permette alla variabile *k* di incrementare il proprio valore di due (invece di 1) dando la possibilità di processare tutti i numeri 0, 2, 4, 6, 8 e quindi 1, 3, 5, 7, 9. Si possono utilizzare tutti gli step che si desidera, l'importante è non superare i valori massimi, pena errori imprevisti del software.

TEST DI UN DISPLAY A 7 SEGMENTI

Esistono apparecchiature che, venendo azionate, illuminano ciclicamente e singolarmente ogni segmento, per testare appunto la sua

[Listato 2]

```
program seven_segment02
    const display as byte[10]= (63,6,91,79,102,109,125,7,127,111)
    dim k as byte
main:
    portb=0 'Azzera portb
    trisb=0 'definisce PORTB in output
    portb=0 'Azzera portb
    delay_ms(2000) 'Attesa 2 sec.
    for k=0 to 8 step 2
        portb=display[k] 'visualizza numero
        Delay_ms(1000)    'aspetta un secondo
    next k
    portb=0 'alla fine spegne Display
    delay_ms(2000)
        for k=1 to 9 step 2
        portb=display[k] 'visualizza numero
        Delay_ms(1000)    'aspetta un secondo
    next k
    portb=0 'alla fine spegne Display
end.
```

funzionalità o meno.
Tale processo è ripetuto abbastanza velocemente per un numero elevato di volte, dando anche una suggestiva idea della complessità tecnologica del congegno stesso.
Questo è ciò che noi realizzeremo: un visualizzatore per ogni singolo segmento che ci permette di controllare la bontà dell'intero display.

L'idea

A tale scopo occorre illuminare, ad uno ad uno, i sette segmenti (a, b, c, d, e, f, g).
Sapendo che ogni segmento è collegato indipendentemente ad ogni pin della PORT-B, e che questi formano praticamente un byte, potremmo scrivere la sequenza che il micro deve generare per l'accensione sequenziale:

a) 0000001 = 1
b) 0000010 = 2
c) 0000100 = 4
d) 0001000 = 8
e) 0010000 = 16
f) 0100000 = 32
g) 1000000 = 64

L'intero ciclo avviene alla cadenza di 100 ms per segmento e ripetuto per 10 volte. Infine l'intero display si spegne. Lo schema è quello mostrato in figura 5.

Il software

Di seguito il listato Basic. Come al solito leggetelo e studiatelo molto attentamente *[Listato 3]*.

Stavolta abbiamo memorizzato nel vettore i valori corrispondenti ai "singoli" segmenti da visualizzare. A tale scopo provate l'utilissimo **7 segment decoder** fornito nel compilatore mikrobasic.
Anche qui incontriamo una "grossa" novità rispetto ai precedenti programmi: il ciclo predisposto a visualizzare i sette segmenti è rappresentato dalla variabile *k*, che passa appunto in rasse-

easyPIC2



La rivoluzionaria scheda di sviluppo per PICmicro

- Programmatore USB on-board
- Tastiera a 32 tasti
- 32 LED per il monitoraggio degli I/O
- 4 cifre LED a 7 segmenti
- Predisposizione per moduli LCD alfanumerici
- Predisposizione per comunicazione RS232
- Alimentazione esterna o tramite bus USB
- Fornita con PIC16F877
- Supporta tutti i PIC a 8, 14, 18, 28 e 40 pin

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

[Listato 3]

```
program seven_segment03
    const display as byte[7]= (1,2,4,8,16,32,64)
    dim k as byte
    dim i as byte
main:
    portb=0 'Azzera portb
    trisb=0 'definisce PORTB in output
    portb=0 'Azzera portb
    for i=1 to 10
        for k=0 to 6
            portb=display[k] 'visualizza segmento
            Delay_ms(100)    'aspetta
        next k
    next i
    portb=0 'alla fine spegne Display
end.
```


gna i 7 valori possibili (da 0 a 6) del vettore. Questo ciclo è a sua volta "racchiuso" in un altro ciclo più esterno, rappresentato dalla variabile *i*.
Tale configurazione prende il nome di **loop nidificato** ed è utilissimo quando un insieme ripetitivo di operazioni deve essere a sua volta ripetuto per un numero definito di volte. State sempre attenti però ad inserire correttamente un ciclo dentro un altro.
Non esiste limite al numero dei loop nidificabili, anche se è buona norma tenere basso questo numero, onde evitare un programma difficile da seguire.

PILOTIAMO 4 DISPLAY: IL MULTIPLEXING

Se volessimo pilotare 4 display a 7 segmenti (per la realizzazione di un orologio o un di cronometro) occorrerebbero in teoria ben 28 porte da adibire solamente alla visualizzazione delle cifre.
Esiste però un metodo molto ingegnoso che permette di ottenere lo stesso risultato utilizzando solamente le 7 uscite!
Come è possibile ciò? Il problema viene risolto grazie ad una tecnica chiamata **multiplexing**, che opera nel seguente modo: i 4 display sono collegati tra loro in parallelo in modo che, nell'invio del codice della cifra da visualizzare, vengano coinvolti tutti i display. Il trucco però sta nell'abilitare sequenzialmente gli stessi display in modo da accenderne uno per volta.
In pratica il micro si assume il compito (oltre a quello di codificare il numero da visualizzare) di abilitare in sequenza ogni display (uno dopo l'altro). Per non far accorgere questo espediente all'occhio umano, tale cadenza deve avvenire molto velocemente, almeno 500 volte al secondo, in modo da dare l'illusione che tutti i display siano sempre accesi.
Naturalmente non si può avere la botte piena e la moglie ubriaca: dobbiamo sottostare a due svantaggi che rendono la vita più complicata:

1) Il codice Basic è molto più complicato e lungo da scrivere, con la maggiore occupazione della RAM del micro;
2) Dal momento che, in un dato istante, solamente un display è acceso e gli altri tre sono spenti, la luminosità dell'intero numero mostrato subirà una drastica riduzione del 75%.

Il secondo svantaggio può essere in parte risolto. Esistono sul mercato comunque alcuni circuiti integrati che sono adibiti proprio a questo compito, ma, per motivi didattici, ne facciano a meno, dal momento che implementiamo tale funzionalità nel nostro micro.
Realizziamo adesso un contatore a quattro cifre da 0 a 9999 con incremento unitario alla

cadenza di 1 secondo. Avvertiamo sin d'ora che tale tecnica non è per niente semplice, pertanto la piena comprensione del listato Basic è necessaria.

Realizziamo un contatore

Lo scopo dell'esercizio è quello di realizzare un semplice contatore unitario, di passo 1 con incremento progressivo di cadenza 1 Hz.
Il risultato finale sarà la visualizzazione di un numero sequenziale, apparentemente stabile, che mostrerà il conteggio da 0000 e terminerà a 9999.

Schema elettrico

Per lo scopo realizziamo lo schema elettrico proposto in figura 8.
Si tratta , come si vede, di una normale applicazione del PIC. I quattro display sono collegati in parallelo.
La codifica del numero è affidata alla PORT-B, mentre il pilotaggio del singolo display è affidato alla PORT-A. Potrebbe risultare alquanto difficoltoso il collegamento tra le parti, poiché sono molti i segmenti di segnale da considerare nel circuito.

Il software

Esaminiamo attentamente il software del prototipo *[Listato 4]*.

Come si nota, i quattro cicli nidificati scandiscono rispettivamente le singole cifre da visualizzare. Dal momento che ogni ciclo conta dieci volte, e che ogni ciclo si trova contenuto all'interno di un altro, vengono "processati" 10x10x10x10 (10.000) numeri.
Ogni cifra (migliaia, centinaia, decina, unità) è visualizzata sul relativo display, in quanto abilitato dal suo transistor di pilotaggio.
In un determinato istante, solo un transistor si trova in saturazione mentre gli altri tre si trovano in interdizione.
Dato che ogni cifra resta "accesa" per 1 ms, e

[Listato 4]

```
program seven_segment04
const display as byte[10]=(63,6,91,79,102,109,125,7,127,111)
dim migliaia as byte
dim centinaia as byte
dim decine as byte
dim unita as byte
dim k as byte
main:

   porta=0 'Azzera porta
   portb=0 'Azzera portb
   trisa=0 'definisce PORTA in output
   trisb=0 'definisce PORTB in output
   for migliaia=0 to 9
     for centinaia=0 to 9
       for decine=0 to 9
         for unita=0 to 9
            for k=1 to 250
               porta=1 'Attiva display 1
               portb=display[migliaia] 'Visualizza
               delay_ms(1)
               porta=2 'Attiva display 2
               portb=display[centinaia] 'Visualizza
               delay_ms(1)
```

```
                porta=4 'Attiva display 3
                portb=display[decine] 'Visualizza
                delay_ms(1)

                porta=8 'Attiva display 4
                portb=display[unita] 'Visualizza
                delay_ms(1)
            next k
          next unita
        next decine
      next centinaia
    next migliaia
porta=0
portb=0

end.
```

quattro cifre impiegano quindi 4 ms. è ovvio che, se si vuol dare al circuito la cadenza di 1 secondo, occorre ripetere questo iter per 250 volte. Da qui la funzione del ciclo più interno rappresentato dalla variabile *k*.

Nota importante: se volete effettivamente "vedere" ciò che succede dietro le quinte del programma, apportate le seguenti modifiche: nello statement for *k=1 to 250* sostituite il valore 250 con 1; i quattro statement *delay_ms(1)* vanno sostituiti con i rispettivi *delay_ms(250)*, per permettervi di osservare più tranquillamente il processo di multiplexing.
Consigliamo vivamente di provare tutti i programmi riportati nella rivista, proprio per applicare in pratica i concetti illustrati negli articoli.

Divisione dei compiti

Nel progetto appena realizzato le due porte del PIC sono utilizzate per due differenti compiti. Il più oneroso è affidato alla PORT-B, che ha la mansione di trasmettere ai display (collegati in parallelo) il numero da visualizzare.
La PORT-A invece ha la funzione di abilitare "sequenzialmente" i quattro display, in rapida successione.
Le resistenze da 2,2 Kohm collegate alle quattro uscite garantiscono una erogazione di corrente estremamente bassa, nell'ordine di appena 2 mA.

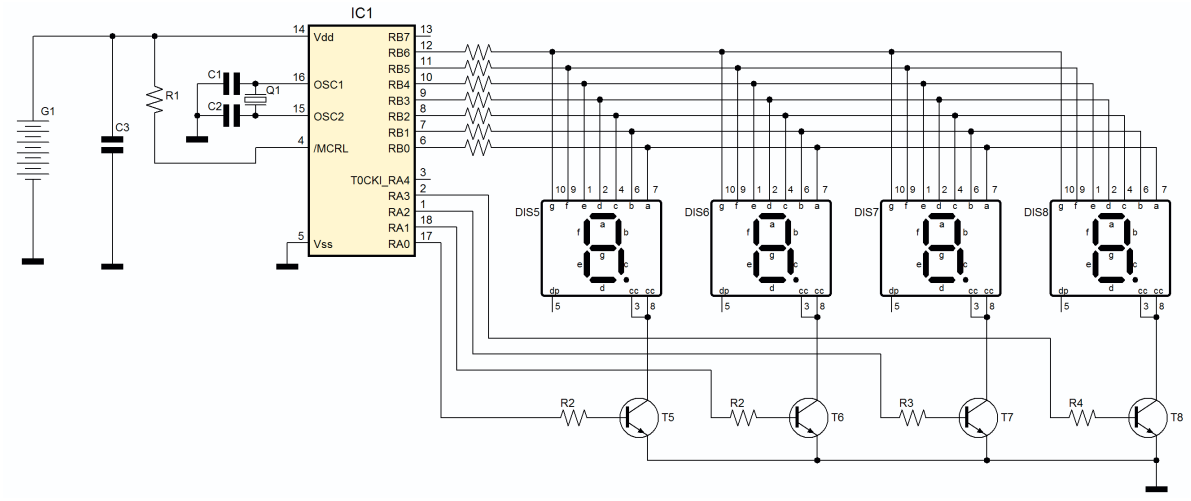


Figura 8 *Schema elettrico del contatore a 4 display*

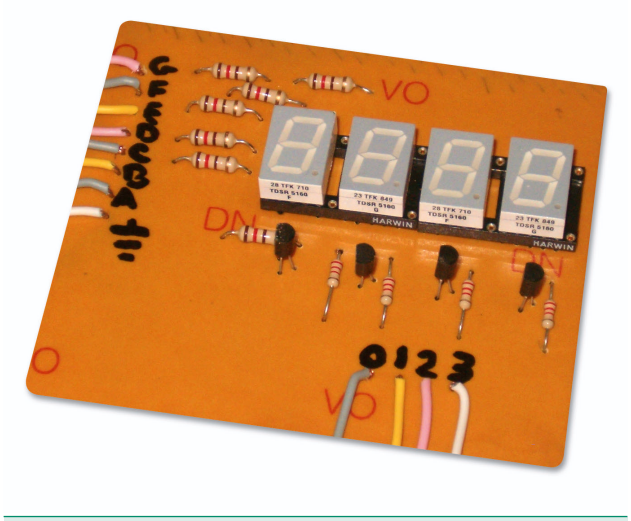


Figura 9 Montaggio dei display a 7 segmenti

AZIONARE CARICHI ROBUSTI

Il PIC non è un dispositivo di potenza, anzi è proprio "deboluccio" considerando la sua massima corrente di erogazione. È però possibile comandare carichi di potenza semplicemente corredando le sue uscite di adatti transistor o MOSPOWER (questi ultimi fortemente consigliati).

Un motore temporizzato

Si debba realizzare il seguente progetto: su una fontanella (da cui inizialmente non esce acqua) è presente un pulsante.
Se questo viene premuto, si aziona un motorino in CC che permetterà la fuoriuscita del prezioso liquido, per un periodo di 15 secondi. Terminati i quali il motore si arresta e la fontanella ritorna allo stato d'attesa iniziale.

Schema elettrico

Lo schema elettrico è molto semplice. Su RB0 è collegato un pulsante normalmente aperto, con resistenza di pull down che assicura lo stato logico basso in condizioni di riposo. Su RB1 è collegato un Mosfet di potenza, il classico IRF540 che è capace di sopportare senza dolore parecchie decine di Ampere di corrente. Una vera forza. Questo Mospower comanda un motore a corrente continua di 12 Volt, che può essere una pompa d'aspirazione d'acqua a basso voltaggio. Si noti in parallelo al motore il diodo di protezione, che interviene per smorzare le extratensioni generate dagli indotti del motore stesso, che potrebbero danneggiare il Mosfet. In effetti esso è gia dotato internamente di un diodo di protezione, ma abbondare non fa mai male.
Ricordate di maneggiare con cura il mosfet, in quanto una scarica di elettricità statica lo potrebbe rovinare.

Analisi

Le fasi che il microcontrollore dovrà gestire per eseguire il software programmato sono le seguenti:

1. Inizializza ed azzera le porte.
2. Se l'uomo preme il pulsante (su RB0):
   a) Avvia motore (su RB1);
   b) Mantiene tale stato per 15 secondi;
   c) Spegne motore.
3. Ritorna al punto 2 (ciclo infinito).

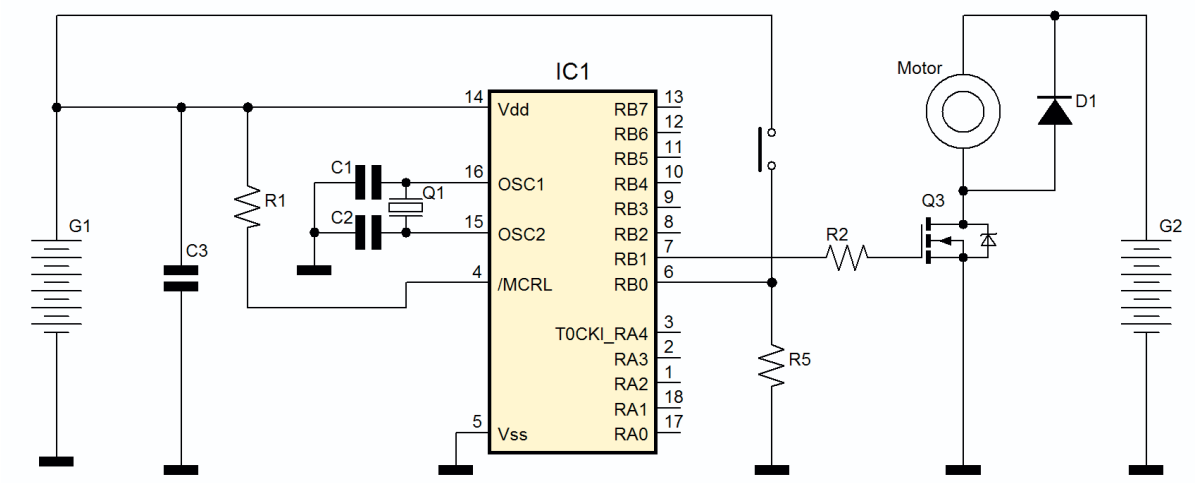


Figura 10 Schema del controllore di motore

Con un microcontrollore si può realizzare in effetti qualsiasi progetto di automazione, dal più semplice al più complicato. È il software a decidere le fasi da seguire secondo le periferiche collegate al PIC.

Il software

Leggete al solito i commenti che spiegano ulteriormente le fasi svolte dal programma seguente:

```
program Fontana

main:
    portb=0 'Azzera portb
    trisb=%00000001 'RB0 Input
                    'RB1 Output
                    'Altri Output

inizio:
    if portb.0 = 1 then ' Se premo pulsante
       portb.1 = 1 ' Avvia motore
       delay_ms(15000) ' aspetta 15 secondi
       portb.1 = 0 ' Spegne motore
    end if
    goto inizio     ' Loop Infinito

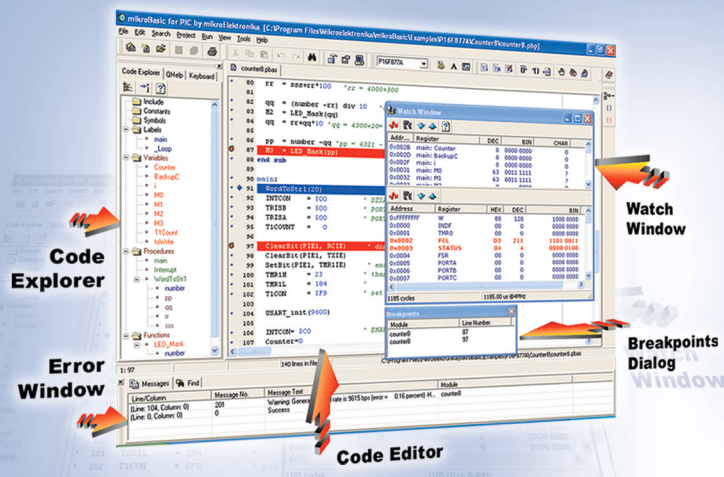
end.
```

Il programma stavolta è davvero semplice.
Da notare che, quando il motore è acceso, la pressione del tasto non ha effetto, essendo il programma "bloccato" sulla istruzione *delay_ms(15000)*.
Potete tranquillamente provare altri esempi, utilizzando magari altre porte, in input e output e usare altri tipi di carichi di potenza, come lampade, cicalini, relè, e molti altri. Sono comunque a disposizione di chiunque volesse chiedere consigli o altro per la realizzazione dei prototipi.

USO DEL DISPLAY A CRISTALLI LIQUIDI (LCD)

Finalmente la ciliegina sulla torta, l'applicazione che si attendeva per completare l'argomento

MIKROBASIC



Un potente compilatore basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

output del microcontrollore.
Un display LCD (intelligente) è un sofisticato circuito che permette di visualizzare messaggi e caratteri in genere su uno spazio suddiviso in righe e colonne.
Si tratta di un display dotato di un microprocessore interno che ha il compito di decodificare i comandi ricevuti dall'utente e di visualizzare uno o più caratteri suddivisi su una o più righe.
Ragion per cui deve essere obbligatoriamente pilotato da un microcontrollore, poiché il protocollo di comunicazione utilizzato è alquanto complesso.
Il Basic facilita enormemente tale processo, in quanto permette di ridurre a pochi comandi tutto il lavoro necessario a visualizzare un dato, mentre scrivendo le routines in Assembler occorrerebbero centinaia di comandi e istruzioni.

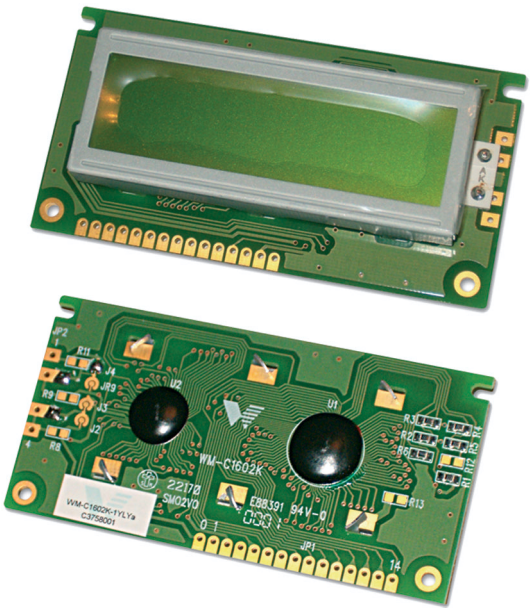


Figura 11-12 Il display LCD visto da entrambi i lati

Il modulo utilizzato

Per i nostri esperimenti abbiamo pensato di utilizzare un modulo economico, funzionale e semplice da utilizzare.
Ecco le principali caratteristiche:

- 16 caratteri x 2 linee.
- Carattere di 5 x 7 pixel.
- Dimensioni: 6,5cm.x1,5 cm.
- Contrasto regolabile.
- Condizioni di temperatura non critiche.
- Bassissimo consumo.

Campi di utilizzo

Si usa il display LCD quando occorre visualizzare messaggi o elaborazioni da parte del circuito utilizzato, creando una sorta di "colloquio" interattivo tra uomo e apparecchiatura.

Caratteri

La ROM del display contiene un set formato da molti caratteri, appartenenti al diffuso codice ASCII.
L'insieme dei caratteri prevede numeri, lettere maiuscole e minuscole, segni di punteggiatura e simboli grafici vari.

Schema applicativo

In figura 13 è mostrato il collegamento del display alle porte di un PIC. Per default è conveniente utilizzare le porte mostrate, per evitare di modificare l'impostazione iniziale del compilatore.
Il pin 1 del display e il pin 2 rappresentano l'alimentazione del nostro LCD.
Agli stessi è collegato in parallelo un potenziometro che funge da partitore variabile per attingere una tensione regolabile atta a stabilire il contrasto ottimale. Il perno centrale del potenziometro è connesso al pin 3. Il pin 4 (RS) è collegato a RB2.
Il pin 5 (R/W) è collegato a massa e comanda l'azione di scrittura sul display anziché quella di lettura. Il pin 6 (E) è collegato a RB3. I pin 11, 12, 13, 14 (che rappresentano il dato vero e proprio da scrivere in formato doppio nibble) sono collegati rispettivamente a RB4, RB5, RB6, RB7.
Tale configurazione è ottimale per la maggior parte delle applicazioni, in quanto lascia liberi 2 pin della PORT-B e non viene minimamente utilizzata la PORT-A.

Il nostro primo programma

Esaminiamo il brevissimo listato per accorgerci subito di quanto sia semplice la scrittura e l'utilizzo di un display LCD alquanto complesso *[Listato 5]*.

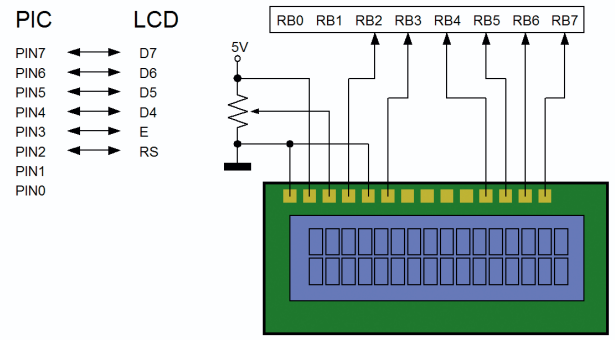


Figura 13 *Connessioni di un display LCD al PIC*



Figura 14 *Il display in funzione*

Più semplice di così non si può.
Dopo aver azzerato la PORT-B e definita la stessa come uscita, il primo comando lcd_init predispone la preparazione del display, affinché esso possa ricevere i comandi per il funzionamento.
Seguono una serie di comandi, quali la cancellazione del display, la disattivazione del cursore e soprattutto la visualizzazione di due messaggi, il primo dei quali in riga 1 (sopra) ed il secondo in riga 2 (sotto). In figura 14 possiamo vedere il display "al lavoro".

In tabella 1 elenchiamo invece tutti i comandi, con la relativa funzione svolta, sopportati dalla funzione di libreria lcd_cmd.

UN SEMPLICE CRONOMETRO

Grazie all'uso di un microcontrollore, anche la creazione di un cronometro digitale, funzione abbastanza complicata, è resa relativamente semplice e soprattutto realizzabile.

Un cronometro serve in molte applicazioni pratiche: in cucina o in una gara di velocità, per la temporizzazione di uno sviluppo in camera oscura o per i classici giochi tra amici, ecc.
Scopo di quest'ultima lezione è quello di dare al lettore una panoramica sulla realizzazione concreta di un preciso conta tempo, semplice da utilizzare e affidabile, ma soprattutto auto-costruito.

Caratteristiche

Il nostro cronometro deve poter contare, con accurata precisione, le ore, i minuti, i secondi e i decimi di secondo La visualizzazione avviene su un display LCD con la seguente notazione: oo:mm:ss:d. Esso è dotato di due pulsanti: uno è adibito allo start, ossia all'avvio del ciclo di conteggio, l'altro allo stop, cioè alla fermata del cronometro.
Non abbiamo, per il momento, voluto aggiungere altri comandi o funzioni, in quanto ci preme maggiormente l'aspetto didattico del progetto.

[Listato 5]

```
program LCD
main:
  portb=0 'Azzera portb
  trisb=0 'La definisce in Output
  Lcd_Init(PORTB) 'Inizializza DISPLAY
  lcd_cmd(LCD_CLEAR) 'Invia comando di cancellazione
  lcd_cmd(LCD_CURSOR_OFF) 'Spegne il cursore
  lcd_out(1,1,"Leggete") 'Stampa a riga 1 colonna 1
  lcd_out(2,1,"FARE ELETTRONICA") 'Stampa a riga 2 colonna 1
end.
```

Comando	Descrizione
LCD_First_Row	Muove il cursore alla prima riga
LCD_Second_Row	Muove il cursore alla seconda riga
LCD_Third_Row	Muove il cursore alla terza riga
LCD_Fourth_Rov	Muove il cursore alla quarta riga
LCD_Clear	Cancella il display
LCD_Return_Home	Riporta il cursore nella posizione iniziale (la Data Ram non viene modificata)
LCD_Cursor_Off	Spegne il cursore
LCD_Underline_On	Accende il cursore sottolineato
LCD_Blink_Cursor_On	Accende il lampeggio del cursore
LCD_Move_Cursor_Left	Muove il cursore a sinistra (la Data Ram non viene modificata)
LCD_Move_Cursor_Right	Muove il cursore a destra (la Data Ram non viene modificata)
LCD_Turn_On	Accende il display
LCD_Turn_Otf	Spegne il display
LCD_ShiftLeft	Shift del cursore a sinistra (la Data Ram non viene modificata)
LCD_Shift_Right	Shift del cursore a destra (la Data Ram non viene modificata)

Tabella 1 *I comandi di LCD_CMD*

Analisi Hardware

Analizziamo la parte elettrica ed elettronica del progetto. Allo schema base del microcontrollore è collegato il display intelligente sulla PORT-B (RB2-3-4-5-6-7).
I due pulsanti che permettono l'avvio e lo stop del conteggio sono invece collegati alle porte RB0 e RB1 (sempre sulla PORT-B) e quindi queste ultime

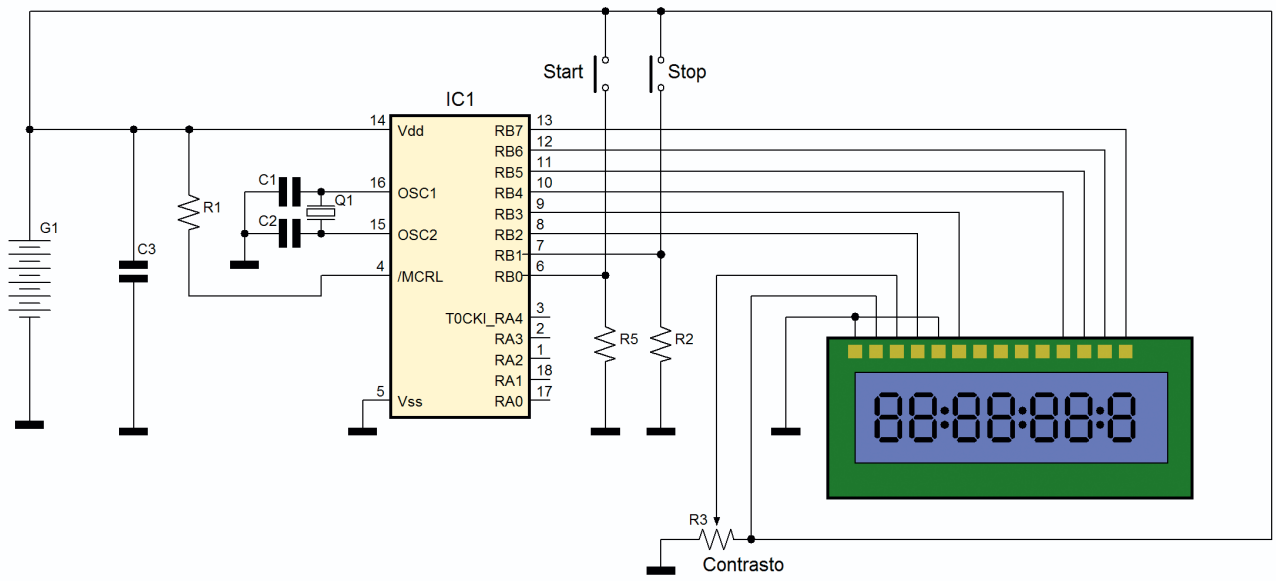


Figura 15 *Schema elettrico del cronometro*

vanno configurate come ingressi. Il potenziometro per il contrasto LCD è di 47 Kohm.
In figura 15 riportiamo lo schema completo, da realizzare con cura, utilizzando in alternativa anche delle connessioni volanti.

Analisi Software

Il funzionamento del cronometro è ridotto proprio all'osso. Non essendo addirittura previsto il tasto per il reset, lo si azzera togliendo e rimettendo l'alimentazione.
Ecco in sintesi le parti salienti dell'algoritmo:

- All'accensione mostra 00:00:00:0.
- Attende la pressione del tasto Start.
- Alla pressione di questo inizia il conteggio.

[Listato 6]

```
program Cronometro

dim ore    as byte 'Definisce ore
dim min    as byte 'Definisce minuti
dim sec    as byte 'Definisce secondi
dim decimi as byte 'Definisce decimi

dim ore_dec as byte 'Decine ore
dim ore_uni as byte 'unita ore
dim min_dec as byte 'Decine minuti
dim min_uni as byte 'unita minuti
dim sec_dec as byte 'Decine secondi
dim sec_uni as byte 'unita secondi

dim conta as byte 'Flag di controllo

main:

   portb=0 'Evita shock PORTB

   'Definisce RB0 e RB1
   'come input, gli altri
   'come output. Avremmo
   'potuto anche scrivere
   'trisb=3 (in decimale)
   trisb=%00000011

   lcd_init(portb) 'Inizializza LCD
   lcd_cmd(LCD_Clear) 'Cancella display
   lcd_cmd(LCD_CURSOR_OFF) 'Cursore OFF

   conta=0 'Inizialmente il conteggio è bloccato

   '------Inizia CICLO--------

   for ore=0 to 23 'Processa le ore
     for min=0 to 59 'Processa i minuti
```

```
        for sec=0 to 59 'Processa i secondi
          for decimi=0 to 9 'Processa i decimi
               '---Calcola decine e unita ore---
               ore_dec=ore/10  'Decina
               ore_uni=ore-(ore_dec*10)
               '---Calcola decine e unita minuti---
               min_dec=min/10  'Decina
               min_uni=min-(min_dec*10)
               '---Calcola decine e unita secondi---
               sec_dec=sec/10  'Decina
               sec_uni=sec-(sec_dec*10)

               '---Visualizza su LCD-------
              lcd_chr(1,1,ore_dec+48)
              lcd_chr(1,2,ore_uni+48)
              lcd_chr(1,3,":")
              lcd_chr(1,4,min_dec+48)
              lcd_chr(1,5,min_uni+48)
              lcd_chr(1,6,":")
              lcd_chr(1,7,sec_dec+48)
              lcd_chr(1,8,sec_uni+48)
              lcd_chr(1,9,":")
              lcd_chr(1,10,decimi+48)

              delay_us(42100) 'Attesa CALIBRATA

              '---Processa tasti START e STOP--
              ' Il programma resta "inchiodato"
              ' sino a quando si preme start

              if portb.1=1 then 'Se premo STOP
                 conta=0    'Disabilita conteggio
              end if
              while conta=0 'Conteggio Congelato
                 if portb.0=1 then 'Se premo START
                    conta=1 'Abilita il conteggio
                            'ed esce dal LOOP
                 end if
              wend
              '-------------------------

          next decimi
        next sec
      next min

    next ore

end.
```

- Alla pressione del tasto Stop si ferma il tempo.
- Se si preme nuovamente il tasto Start il conteggio riprende.
- E cosi via…

In figura 16 rappresentiamo il display LCD durante il suo funzionamento da cronometro.

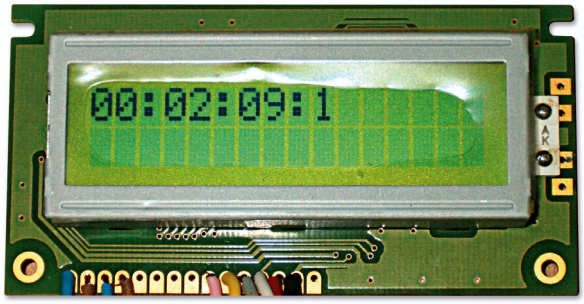


Figura 16 *Il cronometro in Funzione*

Il software

Qui sotto è riportato il listato Basic del cronometro digitale. Diamogli un'attenta occhiata, poi passiamo ai commenti *[Listato 6]*.

Bene, veramente un bel programma, non c'è che dire. Esaminiamone i punti salienti.
Le quattro variabili di tipo byte, *ore, min, sec* e *decimi* hanno lo scopo di contabilizzare il tempo da visualizzare. Tengono cioè il conto del tempo passato. Discorso diverso per quanto riguarda le altre sei variabili (decine e unità). Per poter visualizzare correttamente un numero, potremmo usare le normali funzioni dedicate a questo scopo.
Sorge però un problema: se il numero da visualizzare è composto da due cifre (es. 57) esso comparirà normalmente, ma se invece è composto da una sola cifra (es. 3) non comparirà nel formato necessario all'orologio (cioè 03). Se per esempio le quattro variabili hanno in un dato momento i valori rispettivamente di 16, 3, 8, 3 il display stamperà il valore 16:3:8:3 e non il classico 16:03:08:3.
Proprio a questo scopo serve la routine del calcolo delle decine e delle unità all'interno dei cicli. Dopo l'inizializzazione della PORT-B e suo azzeramento, si provvede anche a inizializzare il display LCD attraverso i comandi dedicati.
Il ciclo inizia con quattro loop nidificati: al termine di un ciclo più interno avviene l'incremento della variabile più esterna. La parte della visualizzazione su LCD è molto interessante: i numeri calcolati non possono essere subito stampati, ma vanno prima "aumentati" di 48 poiché, nella tabella ASCII, il relativo valore è proprio più alto di 48. Per esempio, se una variabile contiene il numero 3, per stamparlo su display dovremmo aumentarlo di 48 (3+48=51) poiché il codice ASCII del carattere 3 è proprio 51 (il nostro display accetta solo codici ASCII).
Degno di nota è l'algoritmo che si occupa della gestione tasti di Start e Stop. Tale routine è governata dalla variabile *conta*, utilizzata come **flag**, ossia come contrassegno. Il suo valore è posto a 0 se il conteggio deve essere disabilitato, mentre a 1 se il conteggio può proseguire. I vari cicli condizionali e iterativi svolgono proprio lo scopo di controllo e di decisione dei due tasti, permettendo così di azionare o interrompere il conteggio come desiderato.
Tutto questo avverrebbe in tempi molto ridotti, se non fosse rallentato dalla pausa dello statement *delay_us(42100)* che introduce un ritardo "calibrato" per far rispettare al cronometro la corretta cadenza.
Questo valore è già di per se abbastanza accettabile, ma se, nei tempi lunghi, si riscontrasse un rallentamento eccessivo nel cronometraggio o addirittura una piccola accelerazione, basterà ritoccare di poco questo valore fino al raggiungimento della massima precisione possibile.

CONCLUSIONI

Bene, siamo arrivati alla fine della puntata. In effetti non basterebbero nemmeno mille pagine per illustrare tutti gli aspetti di questo fantastico compilatore Basic.
Vi assicuriamo invece che, con piccoli spunti iniziali, con vostre idee e soprattutto con tanta pazienza e passione, nonché molta sperimentazione, arriverete certamente a conoscerne a fondo i segreti. Anche in questo campo è l'esperienza a far da padrona.
Alle prossime puntate quindi con tanti interessanti esperimenti e studi per utilizzare al meglio il Mikrobasic. Vi aspettiamo.

Terza parte
n° 240 - Giugno 2005
Utilizzare al meglio le uscite del PIC

Quarta parte
n° 241 - Luglio/Agosto 2005
Utilizzare al meglio gli ingressi del PIC

Quinta parte
n° 242 - Settembre 2005
PWM con il PIC

Mikrobasic per PICmicro

Dopo l'ampia panoramica della scorsa puntata, rivolta alle uscite del PIC, eccoci stavolta a parlare degli ingressi del nostro micro. Impariamo a collegare pulsanti, interruttori e sensori per poter ottenere il massimo da questo straordinario dispositivo.

Il PIC, si sa, è stato progettato per poter risolvere ed eseguire, in maniera del tutto indipendente, una serie di compiti impartiti dal programmatore. Esso dunque è capace di prendere autonomamente le decisioni e di comportarsi di conseguenza, riuscendo a pilotare carichi esterni di qualunque natura. Questo suo funzionamento è affiancato, nella maggior parte dei casi, da eventi esterni, ossia da impulsi o dati provenienti da altre apparecchiature collocate non necessariamente nell'ambito del circuito principale. Sono proprio queste informazioni, trasmesse dall'uomo o da un altro tipo di periferica, che possono condizionare il normale evolversi del programma, o addirittura determinare un particolare risultato del software sviluppato. Esistono alcune applicazioni che, senza l'immissione esterna di informazioni, non potrebbero operare correttamente. Studieremo in questo articolo alcune modalità di comunicazione tra il nostro piccolo microcontrollore e il mondo che lo circonda, in modo da interfacciarlo al resto della circuiteria, discreta o integrata.

L'INGRESSO

Con il termine *ingresso* si intende uno o più terminali ai quali possono venir applicati sia la corrente che la tensione. Spesso è indicato con l'equivalente parola inglese **input**. Un ingresso può essere *analogico* se la grandezza applicata è appunto di natura analogica, e che quindi il suo range di applicazione può variare con continuità e linearità da un minimo ad un massimo consentito. Può essere inoltre *digitale* se la grandezza applicata corrisponde *solamente* a due valori possibili. Se, *digitalmente* parlando, questi valori sono il vero e il *falso*, in senso analogico possono rivestire due valori ben determinati che rappresentano appunto le due possibilità.
Potremmo avere quindi un segnale di 0 Volt e di 5 Volt per rappresentare rispettivamente un valore di falso e vero, ma potremmo avere anche un segnale che può assumere il valore di 0 Volt e di 12 Volt, sempre per rappresentare il falso e il vero. Si potrebbe addirittura avere anche una tensione analogica di –12 Volt (negativi) e di 0 Volt per rappresentare una condizione di Vero e di Falso (quest'ultimo caso utilizzato nella codifica a logica negativa). Insomma si usa sempre una coppia di valori relativi che devono riprodurre la condizione di due stati logici ben precisi. Esiste anche la logica *three states* (logica a tre stati), ma non fa parte della nostra argomentazione.

GLI INGRESSI DEL PIC

Gli ingressi del nostro micro 16F84 funzionano tutti con logica digitale positiva. Questo vuol dire che se applichiamo un tensione di 5 Volt (rispetto a massa) ad un pin (naturalmente configurato come input), il suo stato logico sarà *alto*, mentre se lo colleghiamo a massa, direttamente o attraverso una rete, il suo stato logico sarà *basso*. Il software, sarà in poi grado di riconoscere facilmente lo stato delle porte attraverso dei registri speciali e si comporta di conseguenza, in base alle specifiche del progetto da realizzare. Si tratta di ingressi ad elevata impedenza, quindi, la maggior parte delle fonti di segnale esterno, può essere direttamente collegato senza dover preventivamente adattare

Utilizzare al meglio gli ingressi del PIC



di Giovanni Di Maria
g.dimaria@farelettronica.com

l'impedenza stessa. L'importante che si rispetti la tensione di 5 volt per il livello logico alto.

Valori digitali e analogici per gli ingressi del PIC16F84	
Valore digitale	Valore analogico
Alto (1, Vero, ON, True, H)	+ 5 Volt (rispetto a massa)
Basso (0, Falso, OFF, False, L)	0 Volt (rispetto a massa)

SEGNALI MAGGIORI DI 5 VOLT

In linea di principio, quando un segnale, proveniente dall'esterno del PIC, supera i 5 Volt, deve essere assolutamente abbassato, ossia adeguato in modo da assumere in ogni caso il valore di 5 volt, pena il possibile danno del microcontrollore. Normalmente questa operazione si effettua adottando un partitore di tensione che abbassa il livello esterno alla soglia desiderata. Per esempio: si debba processare un segnale proveniente da un sensore logico che fornisce le tensioni di +20 Volt (cc) e 0 Volt per esprimere rispettivamente i valori logici di vero e di falso. Preso così com'è, tale segnale non è assolutamente utilizzabile dal PIC, ma tramite una opportuna riduzione, attraverso un partitore, otteniamo lo scopo.
N.B. Si assume che il sensore sia capace di fornire la tensione su bassa impedenza, in modo che le resistenze di partizionamento non influiscano con il segnale generato. Ricordate dunque, il PIC lavora solo con segnali di 5 Volt e 0 Volt.
In figura 1 viene mostrato il tipo di collegamento che si deve realizzare per abbassare il livello di tensione a valori accettabili. Calcolate le resistenze utilizzando la legge di Ohm.

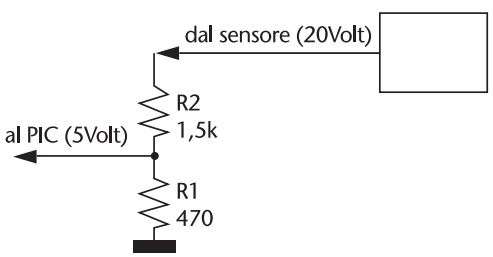


Figura 1 Partizionamento del segnale di ingresso

DUE TASTI, UN LED

Riproponiamo adesso il progetto dei due pulsanti e di un diodo led. Si debba creare il seguente prototipo: su RB0 c'è collegato un diodo LED rosso, su RB1 un pulsante normalmente aperto e su RB2 un altro pulsante normalmente aperto. Il circuito deve comportarsi nel modo seguente:

- All'accensione del circuito entrambi i LED sono spenti.
- Se si preme il primo pulsante (su RB1) il LED dovrà illuminarsi e dovrà mantenere tale stato anche al rilascio del pulsante stesso.
- Se si preme il secondo pulsante (su RB2) il LED dovrà spegnersi e dovrà mantenere tale stato anche al rilascio del pulsante stesso.

Realizzate lo schema della figura 2. Il prototipo può anche essere montato su basetta mille fori. Guardando lo schema si nota che il Led è collegato, tramite la solita resistenza di limitazione di corrente, al pin RB0; quest'ultimo sarà quindi considerato come Output. Sui pin RB1 e RB2

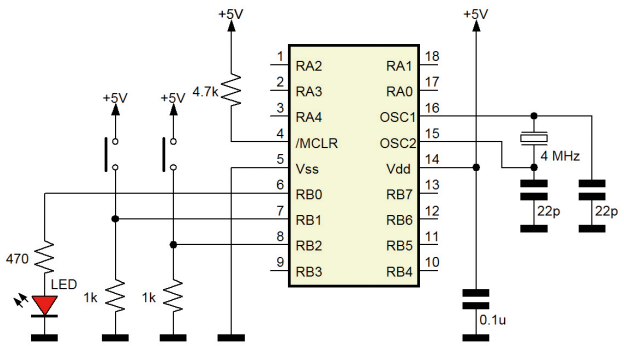


Figura 2 Schema con un LED e due pulsanti

sono collegati invece due pulsanti normalmente aperti. Ecco il funzionamento del circuito: se un pulsante è aperto (cioè non premuto) esso non si collega ai 5 Volt dell'alimentazione, pertanto RB1 (o RB2) sarà connesso a massa tramite la resistenza di 1K (resistenza di *pull-down*). In altre parole tale resistenza "forza" lo stato logico basso (0V) dell'ingresso RB1 (o RB2). Viceversa, quando il pulsante è premuto, il pin RB1 (o RB2) si connetterà direttamente ai 5 Volt dell'alimentazione.
Il pin passerà pertanto a livello logico 1 e la resistenza di pull-down non costituirà assolutamente un ostacolo, in quanto su di essa scorrerà una corrente di soli 5 mA (essendo l'ingresso stesso ad alta impedenza). Volendo "consumare" meno corrente, potremo benissimo "alzare" il valore di tali resistenze fino a 47 Kohm senza problemi. L'importante è che siano presenti, altrimenti, data l'alta impedenza dei suoi terminali, il PIC potrebbe recepire il segnale logico in maniera ambigua (ossia vi sarebbe la presenza di alternati valori 1 e 0 senza apparenti spiegazioni "logiche"). Ecco come il PIC riconosce gli eventi esterni: solamente con segnali logici 0-1 che corrispondono a due livelli analogici 0 Volt e 5 Volt. Studiamo adesso il relativo programma, leggendo attentamente anche i commenti:

```
program Input1

main:
    'definiamo RB2 e RB1 come entrate e RB0
    'come uscita.
    trisb=%00000110

    'Azzeriamo adesso la SOLA porta in
    'OUTPUT (RB0). Non settiamo MAI da
    'software i pin configurati come input.
    'Potrebbe essere pericoloso (per il PIC).
    portb.0=0 'Spegniamo LED

    ciclo:     'Label di iterazione
       if portb.1=1 then 'Se premo pulsante 1
          portb.0=1      'accende LED
       end if
       if portb.2=1 then 'Se premo pulsante 2
          portb.0=0      'spegne LED
       end if
    goto ciclo
end.
```

Dopo le definizioni di rito che comandano il funzionamento della porta e il suo azzeramento, si passa direttamente ad esaminare lo **stato logico** delle due porte-ingresso. Questa operazione viene svolta dal comando IF-THEN che in italiano vuol dire SE-ALLORA. Il micro inizialmente si domanda SE lo stato della RB1 (dove è montato il primo pulsante) si trova a livello logico alto (1). Se così è, viene subito eseguita la successiva istruzione (portb.0=1) cioè effettua l'accensione del led su RB0 (o portb.0, che poi è la stessa cosa). Se invece il livello su RB1 è basso (pulsante non premuto) la successiva istruzione (sempre portb.0=1) è ignorata e quindi non eseguita.
Continuando l'esecuzione, il micro si pone usa seconda domanda, cioè se lo stato logico presente su portb.2 (RB2) è alto (pulsante premuto). Se così è, viene eseguita la successiva istruzione (portb.0=0) e il LED si spegne sino ad altra comunicazione.
Con il comando *goto ciclo*, l'intero processo si ripete all'infinito.
Abbiamo realizzato una sorta di switch digitale che prevede l'accensione e lo spegnimento mediante due pulsanti indipendenti, adibiti esclusivamente a questo scopo.
Il diodo Led può essere idealmente sostituito da altro componente di potenza, previo adattamento del circuito.

IL "RUMORE" DEI CONTATTI

Quando azioniamo un interruttore o premiamo un pulsante, siamo convinti che il segnale, da un certo livello, passi immediatamente ad un altro livello. In teoria accade proprio questo. Facciamo riferimento per un attimo allo schema della figura 3. Si tratta di un semplicissimo circuito composto da un generatore di corrente, da un interruttore e da una lampadina che funge da carico utilizzatore.

Esaminiamo adesso la figura 4: il grafico indica la corrente che attraversa il circuito in vari momenti, simulando l'azione di un interruttore azionato molte volte in un certo intervallo di tempo.
La figura indica solamente il flusso *teorico*, in quanto presuppone la presenza di componenti *ideali*, e quindi "perfetti" sia nelle caratteristiche meccaniche che quelle elettroniche.

Ma nella realtà non è così: durante l'azionamento dell'interruttore, il cambiamento dello stato analogico del circuito non avviene così nettamente, come dice la teoria, ma la corrente subisce una serie di variazioni repentine e casuali, che ne fanno risultare un segnale elettrico tutt'altro che pulito. Guardiamo allora la figura 5 per rendercene subito conto.

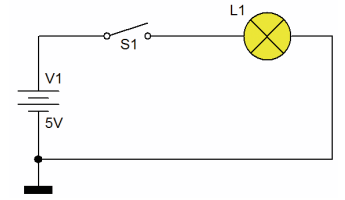


Figura 3 Circuito con interruttore

Come si nota, nonostante le aperture e le chiusure dell'interruttore siano ben distinte, il relativo passaggio di corrente risulta affetto da interferenze non desiderate.
Questo disturbo è denominato **rumore elettrico**, ossia quel breve segnale indesiderato e ad alta frequenza la cui ampiezza potrebbe risultare fastidiosa per il processo di acquisizione dati, in quanto potrebbe alterare le informazioni trasmesse. Questo problema è *sempre* presente nei dispositivi meccanici (pul-

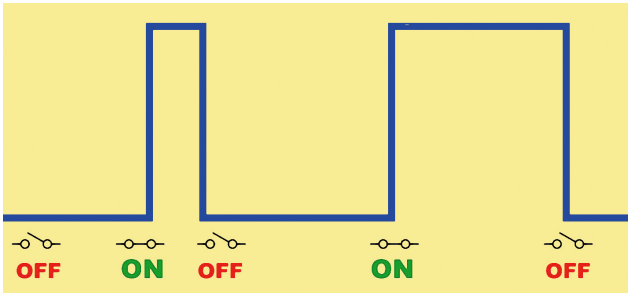


Figura 4 Andamento "teorico" della corrente derivante dallo stato ell'interruttore

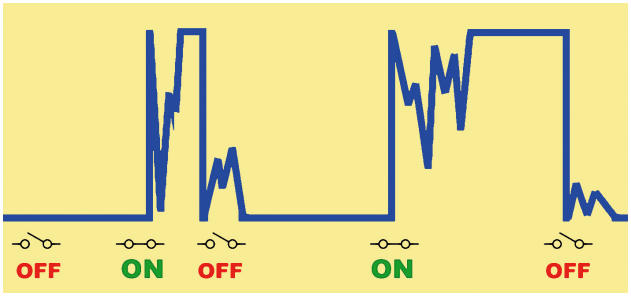
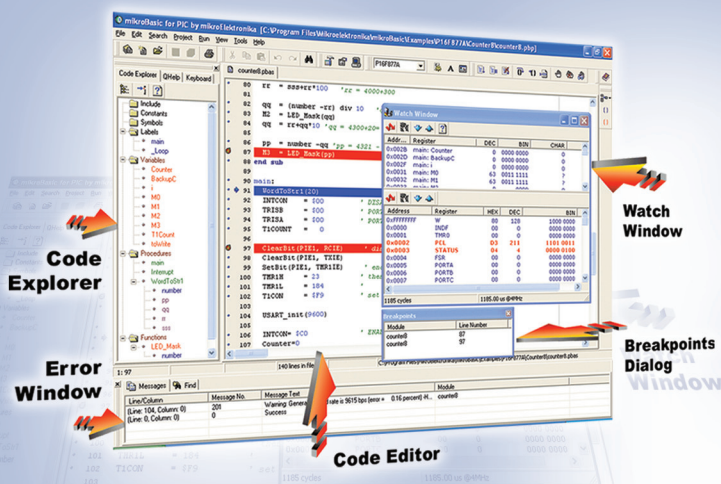


Figura 5 Andamento "reale" della corrente derivante dallo stato dell'interruttore

Software Mikrobasic



Un potente compilatore Basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

santi, relè, interruttori, deviatori, chiavi elettriche, tastiere e commutatori a contatto).

Effetti indesiderati

Se non si eliminasse il problema, i risultati delle elaborazioni potrebbero essere enormemente sfalsati. Pensiamo ad esempio a quei circuiti presenti nei supermercati che servono ad eliminare le code disordinate e a gestire ordinatamente i turni dei clienti.
Ad una pressione del tasto di avanzamento, non si avrebbe il normale incremento di una unità del contatore, bensì un aumento molto elevato e casuale, proprio per la presenza di quei segnali e rumori, che vengono interpretati dal sistema come segnali logicamente corretti.

Il rimbalzo

Il fenomeno che ne deriva è chiamato **rimbalzo** ed è dovuto *alla non perfetta adesione* delle parti mobili che vengono in contatto tra di loro (potrebbero essere anche parzialmente ossidate). Meccanicamente è impossibile da eliminarlo, ma per fortuna esistono alcuni artifizi elettronici che lo annullano completamente.

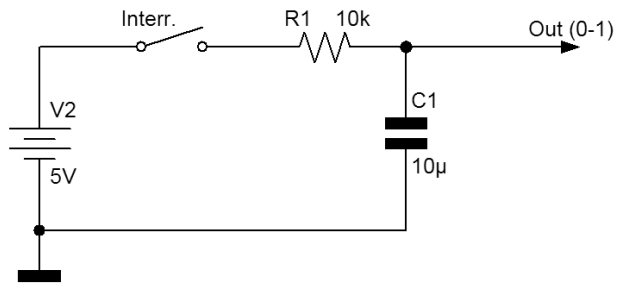


Figura 6 Circuito antirimbalzo RC

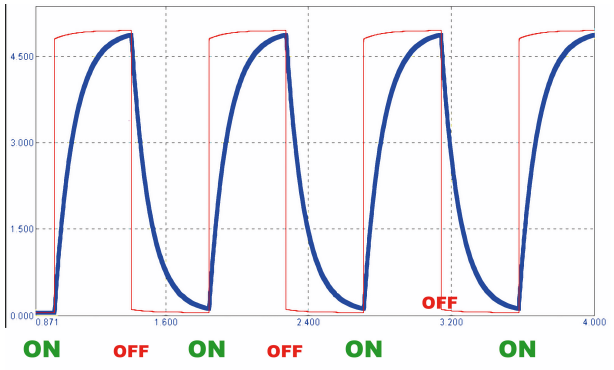


Figura 7 La tensione all'uscita del circuito RC antirimbalzo

Queste tecniche vengono definite appunto di **antirimbalzo**. Accenneremo molto brevemente due tecniche hardware, fornendo solo lo schema elettrico mentre approfondiremo le tecniche software, ossia alcune procedure in Basic che eliminano tale disturbo.

Primo metodo hardware

Questo metodo utilizza una opportuna rete RC per eliminare il problema. Si tratta di un'applicazione particolare dell'integratore, usato quale attenuatore del rimbalzo.
La figura 6 riporta tale soluzione. Il circuito permette l'applicazione di uno "0" logico quando l'interruttore è aperto e un "1" logico quando esso è chiuso.
I componenti assicurano un perfetto *smorzamento* degli effetti del rimbalzo ammorbidendo quindi il segnale risultante.

Il comportamento del circuito è raffigurato nel grafico della figura 7, in cui si evince l'andamento della tensione (molto morbida…) all'uscita del dispositivo.

Secondo metodo hardware

Quest'altro metodo è decisamente più complicato, in quanto si utilizzano le porte logiche, combinate opportunamente tra loro, per eliminare il disturbo. Il dispositivo che ne scaturisce prende il nome di flip-flop.
Si utilizzano allo scopo due porte NOR collegate come mostrato in figura 8.
In uscita Q avremo sempre un segnale pulitissimo e eventuali impulsi spuri all'ingresso non avranno effetti indesiderati e saranno eliminati alla fonte.

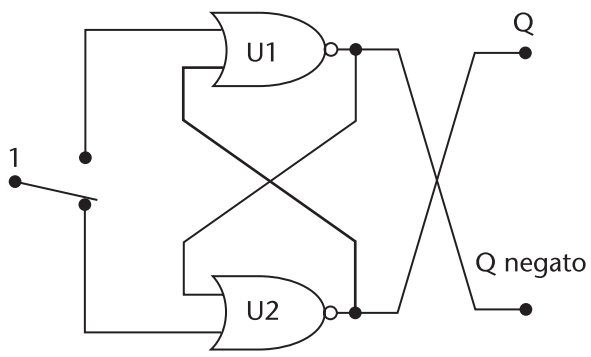


Figura 8 Circuito antirimbalzo flip-flop S-R

Terzo metodo software

È il metodo che più ci interessa, in quanto è proprio questo che utilizzeremo nei nostri studi e nei nostri progetti. Non utilizza alcun artifizio circuitale, ma è solo una tecnica di programmazione che consente di eliminare del tutto il fastidioso rumore elettrico dei contatti.

CONTATORE UP-DOWN AD UNA CIFRA

Bene, adesso che abbiamo esaminato i vari metodi per eliminare il rimbalzo dei contatti, realizziamo un contatore, composto da una sola cifra (un display a 7 segmenti) che proceda avanti ed indietro di una unità, utilizzando semplicemente due pulsanti normalmente aperti. Realizziamo per lo scopo lo schema di cui alla figura 9.

Il software

Trascriviamo il programma seguente nell'editor del compilatore, compiliamo e scarichiamo il file prodotto con estensione .HEX sul nostro micro *[Listato 1]*.

Il programma è molto corto ma contiene preziosismi e trucchi vari. Abbiamo definito l'array di costanti, *display*, che contiene i codici delle dieci cifre da visualizzare. La variabile *k* invece ha la funzione di tenere in memoria il conteggio. Si passa poi all'azzeramento e alla definizione delle porte attraverso i soliti registri PORT e TRIS.

[Listato 1]

```
program seven_segment01

    const display as byte[10]= (63,6,91,79,102,109,125,7,127,111)
    dim k as byte 'Variabile che contiene il NUMERO (0-9)
main:

    porta=0 'Azzera porta
    portb=0 'Azzera portb
    trisa=%00011 'RA0 e RA1 Input
    trisb=0 'definisce PORTB in output

    k=0 'Azzera Cifra
    while true
        '------Processa TASTO 1-------
        if (porta.0=1) and (k<9) then 'Se premo Tasto 1
            k=k+1 'Incrementa contatore
            while porta.0=1 'Ciclo anti REPEAT
            wend
        end if
        '------Processa TASTO 2-------
        if (porta.1=1) and (k>0) then 'Se premo Tasto 2
            k=k-1 'Decrementa contatore
            while porta.1=1 'Ciclo anti REPEAT
            wend
        end if
        '----Visualizzazione
        portb=display[k] 'visualizza numero
        Delay_ms(100) 'Pausa per ANTIRIMBALZO
    wend
end.
```

Adesso viene il bello. Tutto il programma itera attorno ad un ciclo infinito limitato dagli statements *while – wend*. Tale ciclo non termina mai in quanto è specificata una condizione vera (true) che non potrà mai cambiare.
Con la prima istruzione *IF* è processato il tasto su RB0 che è adibito all'incremento del numero. Per aumentare di una unità la cifra, occorre il verificarsi di **due** condizioni, ossia che venga premuto il tasto stesso (porta.0=1) e (**AND**) che il contatore non si trovi al limite del suo conteggio (k<9). Solo così il programma può far aumentare di una unità la variabile k (k=k+1). Subito dopo è eseguito un altro ciclo iterativo che blocca il programma per tutto il tempo in cui il tasto è premuto.
Si evita così il fastidioso effetto *repeat*, che modificherebbe il valore del display per tutta la durata della pressione del tasto.
Il tasto di decremento segue la stessa filosofia di ragionamento, con le ovvie variazioni del caso.Il valore della variabile *k* è così visualizzato sul display attraverso il comando `portb=display[k]` che assegna alla PORTB il valore di codifica del numero. Infine riveste tantissima importanza lo stato di attesa finale, attraverso il comando `Delay_ms(100)` che elimina completamente l'effetto rimbalzo, semplicemente rallentando l'esecuzione del programma. Se volete fare qualche studio didattico, provate ad eliminare (uno alla volta) i cicli interni al processo dei tasti (cicli anti repeat while wend) ed il ritardo antirimbalzo delay_ms.
Vi renderete subito conto che l'affidabilità del conteggio è alquanto compromessa.

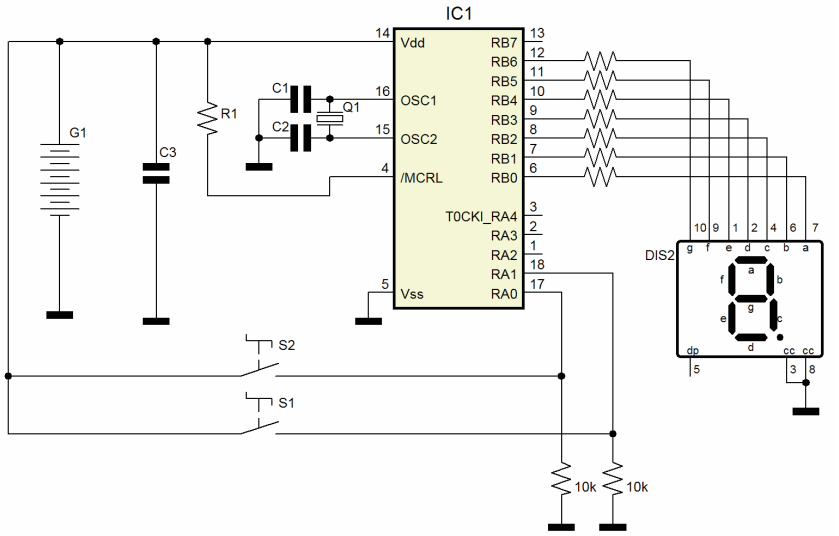


Figura 9 Schema di contatore up-down con due pulsanti n/a

EVENTI ESTERNI SIMULTANEI

Molto spesso si ha l'esigenza di processare un evento SOLAMENTE quando si verificano due o più condizioni, ma contemporaneamente.
Applicheremo in questa sezione alcuni chiari concetti a riguardo. In questi casi è sempre il software a preoccuparsi della rilevazione dei fatti esterni.

Utilizzo combinato dei tasti

Supponiamo di aver bisogno di un circuito, dotato di tre diodi Led (o lampade o motori) e due pulsanti (n/a), che abbia le seguenti caratteristiche:

- All'accensione i tre Led sono spenti.
- Se si preme il pulsante uno, si accendono in sequenza (singolarmente e continuamente) i Led 1, 2 e 3, al ritmo di 100 mS.
- Se si preme il pulsante due, si accendono sin

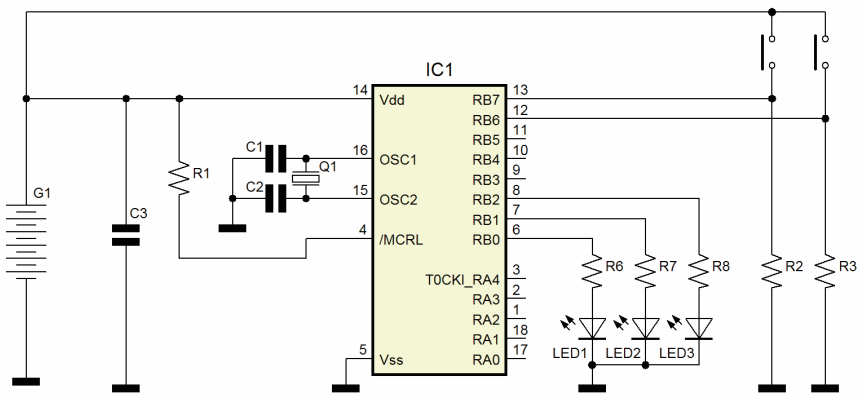


Figura 10 Schema elettrico con tre Led e due pulsanti

sequenza (singolarmente e continuamente) i Led 3, 2 e 1 (quindi in senso inverso) al ritmo di 100 mS.

- Se si premono contemporaneamente i due pulsanti, i tre Led si spengono ed il circuito continua a restare in attesa di ulteriori comandi (loop infinito).

Realizziamo allo scopo il circuito della figura 10.

Il software

Questa volta tutte le periferiche sono collegate alla PORTB, cosicché il suo funzionamento è promiscuo. Diamo un'occhiata al programma *[Listato 2]*.

C'è una grande novità: l'utilizzo del comando *SELECT…CASE*. Esso sostituisce validamente la funzione *IF* quando vi sono numerose condizioni da verificare.
In pratica viene esaminata la variabile stato: a seconda del suo valore, viene eseguita la successione dei comandi relativi, rappresentati dalla clausola *CASE n*, dove n è appunto il valore della variabile in oggetto.
Perché non abbiamo controllato direttamente lo stato dei tasti?
Poiché esso è solamente un evento momentaneo, che si modifica solamente durante la pressione di questi.
Noi abbiamo invece bisogno di un segnalatore, che tenga in memoria l'evento accaduto anche quando il tasto non è più premuto.
In questo programma abbiamo applicato un concetto molto importante, ossia quello di prevedere ben tre eventi avendo a disposizione solamente due tasti. Meditate molto su questo punto…
La tabella della *verità* illustra chiaramente il risultato finale di un'espressione, date due condizioni iniziali:

Tabella della verità della condizione AND del prototipo		
Tasto 1	Tasto 2	Effetto finale
0	0	0 - Comportamento non previsto
0	1	0 - Lampeggio avanti
1	0	0 - Lampeggio indietro
1	1	1- Spegnimento dei due Led

Un programma può essere scritto in tanti modi diversi. Non esiste una sola tipologia di routine o di algoritmo. Lo stile di codifica dipende anche dal proprio carattere, dalla propria logica di ragionamento e di raziocinio.
Pertanto gli esempi proposti non costituiscono (per fortuna) un modello assoluto da seguire ma solamente un'utile traccia da tenere in considerazione durante lo sviluppo del proprio software.

PULSANTI PER GIOCHI A QUIZ

Avete mai visto in tv i giochi a quiz? In questi tipi di giochi viene posta dal presentatore la domanda, a cui devono rispondere i concorrenti nel più breve tempo possibile.
Chi preme per primo il proprio pulsante acquisisce il diritto di risposta, ma soprattutto esclude gli altri concorrenti ad avere lo stesso diritto. In altre parole disabilita la prenotazione degli avversari. In questa sezione realizzeremo proprio questo prototipo.

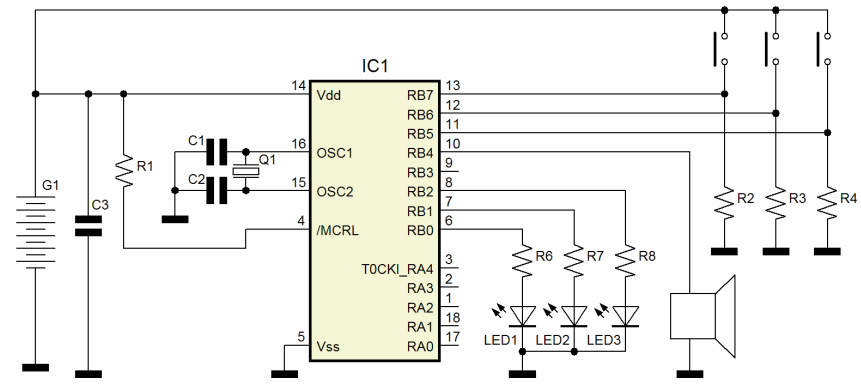


Figura 11 Schema elettrico della pulsantiera per Quiz

L'hardware

Il nostro dispositivo consisterà naturalmente solo della parte elettronica e della parte software. Non saranno quindi comprese (per ovvie ragioni) le relative cabine munite di cuffie e di vetri insonorizzati. Scherzi a parte, il prototipo sarà formato da:

[Listato 2]

```
program Contemporaneo

dim stato as byte
'La variabile stato serve per memorizzare la situazione che si viene a creare con
'la pressione dei tasti. Essa prevede questi tre valori con i quali il software
'si comporta di conseguenza:
' stato=1  quando e' stato premuto il tasto 1
' stato=2  quando e' stato premuto il tasto 2
' stato=3  quando sono stati premuti entrambi i tasti

main:
   portb=0 'Azzera portb
   trisb=%11000000 'RB7 e RB8 Input

   while true
      Delay_ms(50) 'Pausa per antirimbalzo
      '-----Processo del TASTO 1------
      if (portb.6=1) and (portb.7=0) then '(SOLO tasto 1)
         stato=1
      end if
      '-----Processo del TASTO 2------
      if (portb.6=0) and (portb.7=1) then '(SOLO tasto 2)
         stato=2
      end if
      '-----Processo del TASTO 3------
      if (portb.6=1) and (portb.7=1) then '(tasto 1 e 2)
         stato=3
      end if
      '======Adesso processa le 3 situazioni===
      select case stato
         case 1
            portb.0=1 'Accende led 1
            portb.1=0 'Spegne led 2
            portb.2=0 'Spegne led 3
            delay_ms(100)
            portb.0=0 'Spegne led 1
            portb.1=1 'Accende led 2
            portb.2=0 'Spegne led 3
            delay_ms(100)
            portb.0=0 'Spegne led 1
            portb.1=0 'Spegne led 2
            portb.2=1 'Accende led 3
            delay_ms(100)
         case 2
            portb.0=0 'Spegne led 1
            portb.1=0 'Spegne led 2
            portb.2=1 'Accende led 3
```

```
            delay_ms(100)
            portb.0=0 'Spegne led 1
            portb.1=1 'Accende led 2
            portb.2=0 'Spegne led 3
            delay_ms(100)
            portb.0=1 'Accende led 1
            portb.1=0 'Spegne led 2
            portb.2=0 'Spegne led 3
            delay_ms(100)
        case 3
            portb.0=0 'Spegne led 1
            portb.1=0 'Spegne led 2
            portb.2=0 'Spegne led 3
    end select
  wend
end.
```

- 3 pulsanti normalmente aperti (uno per ogni concorrente).
- 3 diodi Led da porre in ogni cabina, che confermano l'avvenuta acquisizione del diritto a rispondere, e quindi indicano quale dei tre concorrenti ha prenotato per primo. I led possono (e devono) essere rimpiazzati da altrettante lampade di potenza (precedute dai relativi transi-

Scheda easyPIC2



La rivoluzionaria scheda di sviluppo per PICmicro

- ✔ Programmatore USB on-board
- ✔ Tastiera a 32 tasti
- ✔ 32 LED per il monitoraggio degli I/O
- ✔ 4 cifre LED a 7 segmenti
- ✔ Predisposizione per moduli LCD alfanumerici
- ✔ Predisposizione per comunicazione RS232
- ✔ Alimentazione esterna o tramite bus USB
- ✔ Fornita con PIC16F877
- ✔ Supporta tutti i PIC a 8, 14, 18, 28 e 40 pin

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

stor) se si intende utilizzare il dispositivo durante un reale gioco a premi.
- 1 altoparlante a media impedenza (o un amplificatore BF) che emette un tono quando un qualsiasi concorrente ha premuto il proprio pulsante.

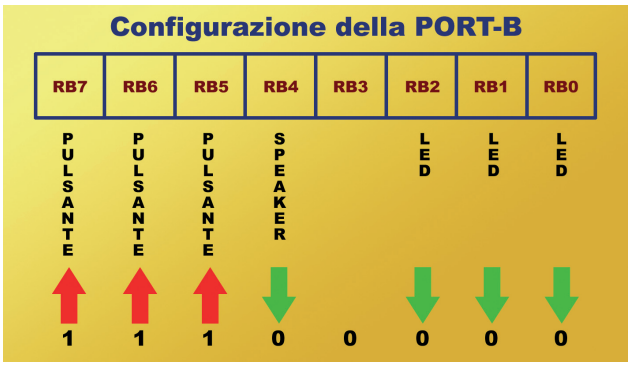


Figura 12 Mappatura della PORT-B per la pulsantiera per Quiz

Schema elettrico

Lo schema elettrico è illustrato in figura 11.
Come si nota, su RB0, RB1 e RB2 sono montati i Led indicatori, pertanto questi tre pin sono configurati come uscite. RB3 è lasciato libero. Su RB4 è collegato un altoparlante di resistenza elevata, per non saturare l'uscita. Su RB5, RB6 e RB7 sono installati tre pulsanti normalmente aperti, a disposizione dei concorrenti. La "mappatura" della PORT-B è raffigurata in figura 12.

Analisi procedurale

Il dispositivo si comporta nel seguente modo: all'accensione del circuito, i tre led e l'altoparlante saranno spenti. Un ciclo infinito molto veloce controlla ciclicamente la pressione di uno dei tre tasti. È impossibile che si verifichi la pressione contemporanea di due pulsanti: c'è sempre una frazione di tempo che intercorre tra gli eventi. Appena il sistema si accorge del tasto premuto, blocca la sua scansione ciclica, accende il relativo led per un certo numero di secondi ed emette in altoparlante una nota acustica. Dopo qualche secondo sia il led che l'altoparlante si disattivano ed il sistema continua la sua normale procedura di attesa e di controllo.

Il software

Occorre studiare a fondo, comprendendolo bene, il listato 3.

Notiamo nel programma due cicli nidificati: il primo ciclo, comandato dal comando *while – wend* costituisce un loop infinito, per assicurare la continuità del gioco. Il secondo invece è quello che controlla, assieme agli statement *if*, la pressione dei tasti.
Tale ciclo si conclude quando la condizione specificata è vera (loop until premuti=1) ossia quando la variabile *premuti* viene posta a 1, attestando così la pressione di uno dei tre pulsanti. I tre statement *if*, oltre a controllare la pressione dei tasti, assicurano, con la seconda condizione *(if (portb.5=1) and (premuti=0) then)* la discriminazione incondizionata del concorrente, ossia la sicura determinazione del giocatore che ha risposto per primo, senza alcuna ambiguità .
Appena il programma esce dal ciclo interno, segno che qualcuno ha premuto il pulsante di gioco, avviene l'accensione del relativo Led, posto sulla cabina del concorrente e, subito dopo, è generata la nota acustica di 500 Hz per la durata di un secondo, grazie alla produzione di un'onda quadra da parte del ciclo *for – next*. Dopo la pausa di 5 secondi, che assicura l'illuminazione continuata del diodo Led, e la momentanea interdizione dei tre tasti, tutti i Led (anche quelli non illuminati…) vengono spenti ed il ciclo di gioco continua sino all'infinito.

CONTA PEZZI DIGITALE

Bene, come ultima applicazione proponiamo adesso un utile conta pezzi digitale. Può essere sicuramente utilizzato in tutte quelle applicazioni in cui occorre gestire il conteggio di avvenimenti o episodi simili e ripetitivi. Esempi classici

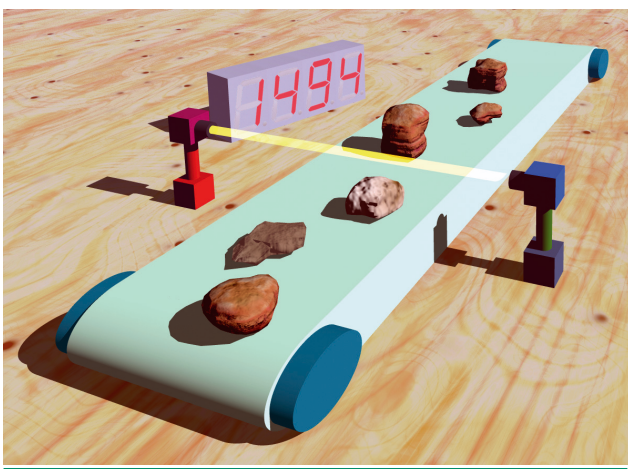


Figura 13 Simulazione di un conta-pezzi

[Listato 3]

```
program Quiz

dim tasto1 as byte
dim tasto2 as byte
dim tasto3 as byte
dim premuti as byte
dim k as word

main:
   portb=0 'Azzera PORTB
   trisb=%11100000  ' Definisce funzionalita' PORTB

   while true
      tasto1=0 ' Azzera stato tasto1
      tasto2=0 ' Azzera stato tasto2
      tasto3=0 ' Azzera stato tasto3
      premuti=0 ' Presuppone che NESSUN tasto sia premuto

      do ' Mentre non si preme NULLA
         ' Se si preme tasto1 e nessun altro e' premuto
         if (portb.5=1) and (premuti=0) then
            tasto1=1 ' Memorizza lo stato del tasto1
            premuti=1 ' Assume che c'e' stata la PRESSIONE
         end if
         ' Se si preme tasto2 e nessun altro e' premuto
         if (portb.6=1) and (premuti=0) then
            tasto2=1 ' Memorizza lo stato del tasto2
            premuti=1 ' Assume che c'e' stata la PRESSIONE
         end if
         ' Se si preme tasto3 e nessun altro e' premuto
         if (portb.7=1) and (premuti=0) then
            tasto3=1 ' Memorizza lo stato del tasto3
            premuti=1 ' Assume che c'e' stata la PRESSIONE
         end if
      loop until premuti=1

      ' Se l'esecuzione arriva QUI vuol dire che e' stato premuto un tasto
      if tasto1=1 then ' Se premuto tasto 1
         portb.0=1       ' Accende Led 1
      end if
      if tasto2=1 then ' Se premuto tasto 2
         portb.1=1       ' Accende Led 2
      end if
      if tasto3=1 then ' Se premuto tasto 3
         portb.2=1       ' Accende Led 3
      end if
```

```
        '--Genera nota a 500 Hz per 1 secondo--
        for k=1 to 500
           portb.4=1 ' Onda quadra ON
           Delay_ms(1)
           portb.4=0 ' Onda quadra OFF
           Delay_ms(1)
        next k

        Delay_ms(5000) ' Aspetta 5 secondi

        '----Spegne tutti i LED------
        portb.0=0 ' Spegne Led 1
        portb.1=0 ' Spegne Led 2
        portb.2=0 ' Spegne Led 3
    wend
end.
```


sono i *conta-pezzi* digitali, utilizzati per conteggiare i prodotti trasferiti su un rullo trasportatore, oppure gli intramontabili *conta-persone*, che si occupano di incrementare il contatore ogni volta che un individuo passa attraverso il raggio della sua azione. In figura 13 possiamo vedere la simulazione (in realtà virtuale 3D) di un ipotetico apparato conta pezzi adibito proprio alla contabilità numerica di vari oggetti.

Come avviene il conteggio

Come si vede dalla simulazione, il sistema è composto da una unità emittente di radiazione luminosa (visibile, infrarossa, laser o simili) e da una unità foto-ricevente (fotoresistenza, fotodiodo, fototransistor o altro).

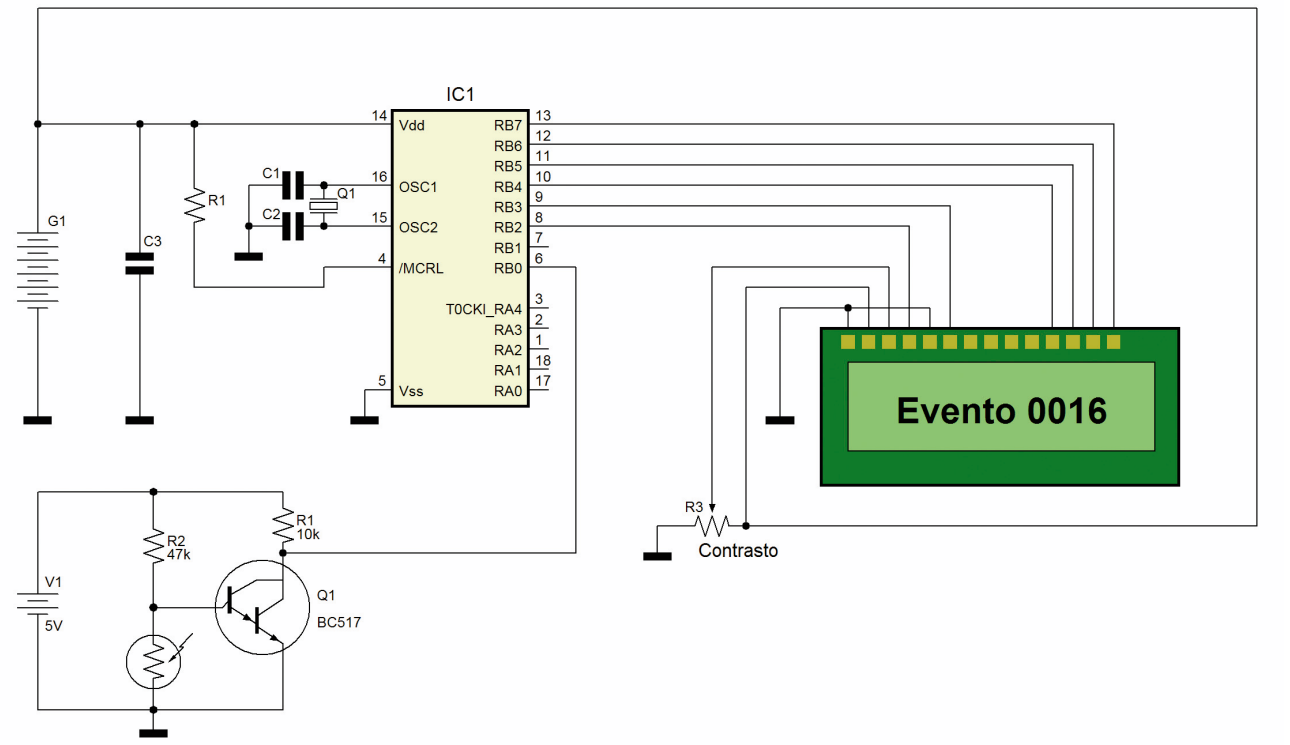


Figura 14 Schema di principio di un contapezzi

Nell'istante in cui un oggetto interrompe il fascio di luce il microcontrollore, accorgendosi immediatamente del cambio di stato, innesca il meccanismo di conteggio, che a sua volta è inoltrato ad una unità di visualizzazione, come ad esempio una serie di display a 7 segmenti o un display LCD. Anche in questo caso occorre fare i conti con il rumore industriale, poiché è spesso presente in questi tipi di applicazione.

Schema Elettrico

Come si evince dallo schema elettrico di cui alla figura 14, si tratta di una tipica applicazione di ingresso digitale collegato in porta RB0. Una foto-resistenza è utilizzata come sensore di passaggio. I valori tipici di questo componente (con variazione quasi lineare) vanno da circa 800 Kohm in assenza di luce a soli circa 200 ohm in piena illuminazione. Quando il sensore è investito dalla luce, e quindi non vi è alcun oggetto che interferisca con il raggio luminoso, la base del transistor BC517 (un darlington dal beta superiore a 30.000 !!!) è posta a massa dalla stessa foto-resistenza. Tale componente si trova pertanto in stato di interdizione ed assicura quindi un potenziale di collettore di 5 Volt (Livello Logico 1). Quando invece il raggio luminoso è interrotto dal passaggio di un oggetto, il valore della foto-resistenza sale a circa 800 Kohm, la base è polarizzata direttamente dalla resistenza di 47 Kohm, ed il transistor commuta in stato di saturazione, abbassando così il potenziale di collettore (verso massa) a 500 mV circa, generando quindi il livello logico 0. Assemblando il modulo del sensore, occorre inserire la foto-resistenza all'interno di un tubetto di plastica nero, per non correre il rischio di

[Listato 4]

```
program Contapezzi

dim evento as word
dim txt as char[6]

main:
  portb=0 'Azzera portb
  trisb=%00000001 'Definizione PORTB
  Lcd_Init(PORTB) 'Inizializza DISPLAY
  lcd_cmd(LCD_CLEAR) 'Invia comando di cancellazione
  lcd_cmd(LCD_CURSOR_OFF) 'Spegne il cursore
  evento=0 'Azzera numero eventi
  while true
     delay_ms(10) 'Pausa antirimbalzo
     if portb.0=0 then
        evento=evento+1 'Incrementa eventi
        while portb.0=0 'Routine Antirepeat
        wend
     end if
     '---Trasforma il numero in STRINGA---
     wordtostr(evento,txt) 'Trasformazione
     '----Visualizza su LCD---
     lcd_out(1,1,"PIC CONTAPEZZI")
     lcd_out(2,1,"Transito:")
     lcd_out(2,11,txt)
  wend
end.
```

illuminarla con altre fonti di luce esterna.
Il resto del microcontrollore è collegato, come abbiamo già visto, al modulo intelligente del display LCD, per visualizzare i relativi conteggi. Lo schema costituisce naturalmente il prototipo base dell'applicazione, ma in realtà anche i circuiti antirimbalzo e gli opportuni Trigger di Shmitt, per filtrare il segnale, sono necessari.

Il software

Riportiamo il software dedicandogli anche gli opportuni commenti *[Listato 4]*.

La variabile utilizzata per contare il passaggio di oggetti o di persone è chiamata *evento*, inizialmente posta a zero. La pausa di 10 ms. assicura un buon antirimbalzo, visto che il rumore dei contatti si verifica ad una frequenza nettamente superiore. L'istruzione *IF* si cura invece della rilevazione dell'occultamento del raggio luminoso e, di conseguenza, del passaggio di un oggetto. All'interno di tale clausola, l'evento non può ripetersi ciclicamente grazie al blocco interno *while wend*, che rende sicura l'univocità dello stato logico zero. Finalmente, all'uscita della condizione, troviamo le istruzioni di visualizzazione. Una nota di attenzione merita la funzione che trasforma il numero in stringa.
Attraverso questa operazione, qualsiasi valore numerico (di qualsiasi numero di cifre) è trasformato in una stringa di caratteri, già pronta per essere direttamente visualizzata attraverso l'istruzione *lcd_out*. Più semplice di così?



Figura 15 *Il display del conta-pezzi al "lavoro"*

Come funziona

Ogni volta che un oggetto passa davanti al raggio di luce, interrompendolo, causa un cambio netto dello stato logico della PORTB.0. Il software interpreta tale mutazione e si comporta di conseguenza, incrementando il conteggio e quindi visualizzandolo su display LCD. Badate bene però che in caso di interruzione dell'alimentazione elettrica, la numerazione ripartirà inevitabilmente da zero. Nelle prossime puntate vedremo come risolvere anche questo problema.

CONCLUSIONI

Bene, abbiamo imparato a conoscere anche gli ingressi del PIC. Come avete potuto vedere, basta applicare con criterio e giudizio gli elementi basilari della programmazione e potrete risolvere da soli anche gli algoritmi più complessi.
Gli esempi fin qui esaminati costituiscono in ogni caso dei prototipi base, da cui prendere spunti per applicazioni e sviluppi futuri.
Non c'è niente di meglio, in ogni caso, che provare da sé i circuiti, cercando di proseguire pian piano nell'approfondimento della materia: la fretta, anche in questo caso, costituisce una cattiva consigliera. Maturate dunque la tecnica di programmazione e "rendetela vostra", coinvolgendo anche il vostro stile di ragionamento.

Lo spazio a nostra disposizione è terminato.
Appuntamento dunque alle prossime puntate con tante nuove applicazioni e circuiti interessanti.

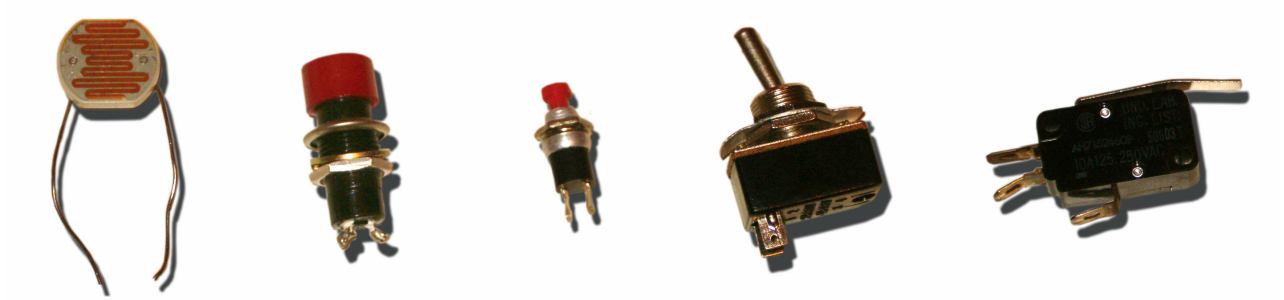


Figura 16 *Esempi di unità di Input*

Quarta parte
n° 241/242 - Luglio/Agosto 2005
Utilizzare al meglio gli ingressi del PIC

Quinta parte
n° 243 - Settembre 2005
PWM con il PIC

Sesta parte
n° 244 - Ottobre 2005
Musica maestro! Generazione di melodie, note, effetti sonori e musica con il PIC

Mikrobasic per PICmicro

Studiamo insieme il PWM, una tecnica molto utile dalle applicazioni più svariate. Utilizzando il Mikrobasic scopriremo come si può gestire e personalizzare questa tecnica al massimo livello, permettendci di accrescere il valore professionale delle nostre realizzazioni.

Il nostro microcontrollore è capace di generare solamente due livelli di tensione in uscita: 0 Volt (livello logico basso) e 5 Volt (livello logico alto). Non esistono mezze misure, nessun compromesso e nessuna altra tensione compresa tra i due limiti: solo questi due valori. Alcune volte però è necessario generare una tensione minore, per poter pilotare, con meno corrente, un determinato carico. Se colleghiamo una lampadina all'uscita del PIC, essa (normalmente) potrà assumere la condizione di spenta (se all'uscita è applicato uno 0 logico) o di accesa (se all'uscita è applicato un 1 logico). Dovendo però illuminare la lampada con minore intensità, è necessario far ricorso alla tecnica del PWM. Solo in questa maniera potremo "dosare" finemente la tensione prodotta, in modo che la stessa lampada può assumere diversi gradi di illuminazione.

IL PWM

PWM è l'acronimo dei termini inglesi *Pulse With Modulation*, ossia impulsi modulati. Si tratta in sostanza di una tecnica geniale, che consente di variare la potenza applicata ad un carico, non modificando la tensione erogata sull'utilizzatore bensì "variando" la percentuale della larghezza degli impulsi applicati al carico stesso. Se quindi si desidera che una lampadina si illumini con diverse intensità, non bisogna cambiare la tensione su di essa ma occorre semplicemente variare (con molta velocità) il tempo durante il quale la tensione massima viene applicata al carico. In sostanza esso viene sottoposto ad un "bombardamento" di impulsi positivi, ossia un **treno di impulsi**, opportunamente calibrato.
È indubbiamente un metodo più complicato, ma permette di ridurre al minimo la dissipazione di eventuali transistor di potenza utilizzati, in quanto si comportano da semplici interruttori e non da regolatori di corrente.
Guardiamo per un attimo lo schema ideale di figura 1. Esso è formato da un generatore di treno di impulsi ad onde quadre P (V1) e da un carico rappresentato da una lampada ad incandescenza (L1). Il generatore d'impulsi fornisce alla lampada solamente dei segnali ON-OFF cadenzati, con determinate temporizzazioni. Con tale tecnica è possibile dosare con continuità la potenza fornita a lampade, motori, led e tanti altri carichi utilizzatori. Alimentando i motori in corrente continua è possibile rallentarne il numero dei giri in modo drastico mantenendo inalterata la coppia specifica, operazione sicuramente impossibile variando solamente la tensione tipica di alimentazione. Esaminiamo alcuni casi.

Duty Cycle al 50%

Il generatore fornisce, nell'esempio, alcuni impulsi ON-OFF con periodo equivalenti, vale a dire 1 millisecondo a livello basso ed 1 millisecondo a livello alto. La lampadina pertanto

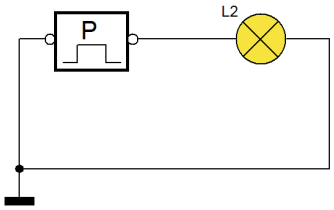


Figura 1 Schema teorico di generatore di impulsi e carico

PWM con il PIC



di Giovanni Di Maria
g.dimaria@farelettronica.com

riceve un treno di impulsi bilanciato, come visibile dal grafico in figura 2. Si illumina quindi con una intensità *dimezzata* rispetto al suo massimo teorico, proprio per il fatto che riceve alternativamente segnali di acceso e di spento. Il ciclo di lavoro dell'onda quadra generata, ossia il suo *duty cycle*, è pertanto del 50%. Il carico, in questo caso, è come se fosse alimentato da una tensione divisa per due, rispetto alla sorgente di alimentazione, con conseguente diminuzione della luminosità o della sua resa in genere.

Duty Cycle al 10%

Se il generatore fornisce un impulso positivo (attivo) della durata di 1 millisecondo ed i rimanenti 9 millisecondi restano inattivi (0 Volt) il duty cycle è pari al 10%. In altre parole, nell'ambito dell'intero periodo della durata di 10 millisecondi, esiste solamente un impulso "produttivo" della durata di 1 ms. (equivalente quindi al 10%). Gli altri 9 ms. costituiscono un periodo di inattività, di tempo morto, nel quale il carico non riceve corrente e quindi non è alimentato. La lampada si illumina in questo caso molto debolmente, al limite minimo della sua incandescenza. In figura 3 è riportato il grafico degli impulsi generati (impulsi stretti).

Duty Cycle al 90%

Se la larghezza attiva dell'impulso è di 9 ms. su un periodo totale di 10 ms. avremo in questo caso un duty cycle del 90%. Il carico resta alimentato per la maggior parte del tempo e fornisce una resa quasi ideale. La lampada si accenderà quasi al massimo. In figura 4 riportiamo il grafico dei relativi impulsi (impulsi larghi).

Tabella di conversione da percentuale del duty cycle a Volt in uscita da una tensione iniziale di picco di 5 Volt

Duty Cicle (in percentuale %)	Valore efficace ottenuto (in Volt)
0	0,0
10	0,5
20	1,0
30	1,5
40	2,0
50	2,5
60	3,0
70	3,5
80	4,0
90	4,5
100	5,0

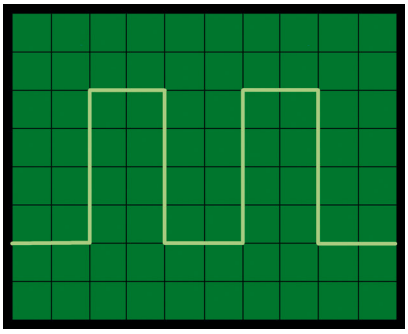


Figura 2 *Duty Cycle al 50%*

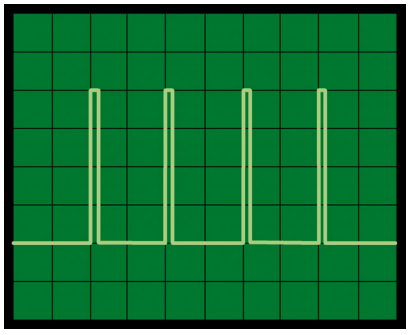


Figura 3 *Duty Cycle al 10%*

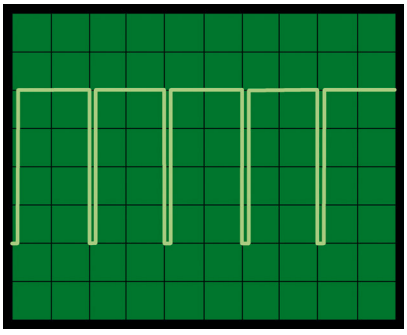


Figura 4 *Duty Cycle al 90%*

DISCO ACROMATICO

Per comprendere meglio il meccanismo del PWM possiamo eseguire l'esperimento del *disco acromatico*. Un disco acromatico è un cerchio rotante, suddiviso in tanti settori circolari alternati, di colore bianco e nero.
Il colore bianco indica (nel nostro caso) lo stato logico alto e quindi il valore analogico di 5 Volt. Il colore nero indica lo stato logico basso e quindi il valore analogico di 0 Volt. Facendo ruotare molto velocemente il disco, esso assume una graduale colorazione grigia, benché tale colore non sia presente.
La densità del grigio è proporzionale alle quantità dei settori bianchi e neri utilizzati. Un grigio scuro indica maggior presenza di aree nere, un grigio chiaro invece indica maggior presenza di aree bianche. Nelle figure 5, 6 e 7 possiamo studiarne il comportamento.

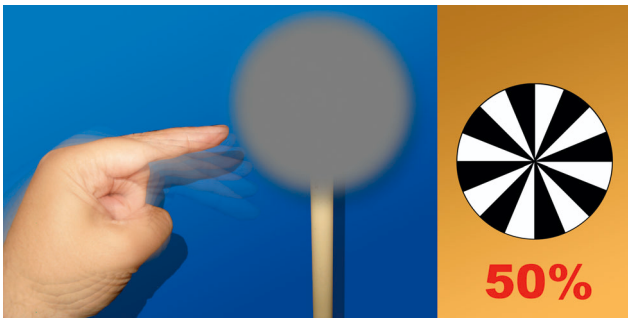


Figura 5 *Disco acromatico al 50%*

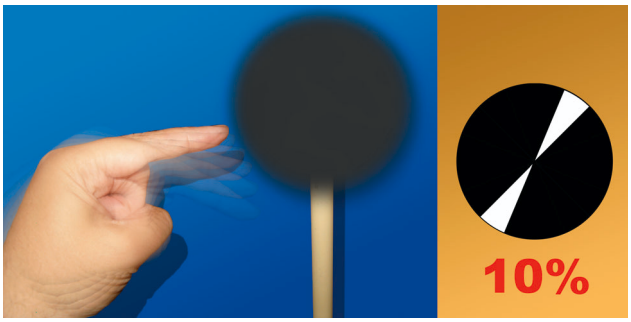


Figura 6 *Disco acromatico al 10%*

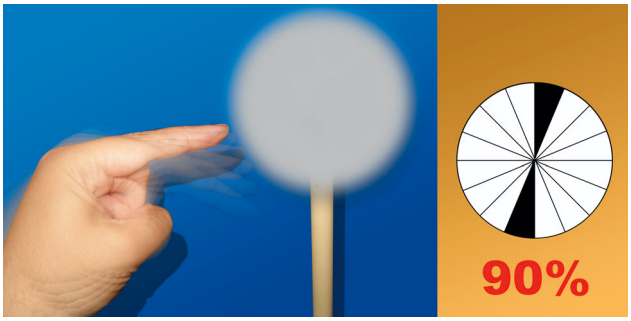


Figura 7 *Disco acromatico al 90%*

PRIMA ESPERIENZA

Sappiamo bene che le uscite del PIC possono generare solamente una tensione di 5 Volt per il livello logico alto ed una tensione di 0 Volt (tensione a massa) per il livello logico basso. Vedremo, con questo esperimento, come sia invece possibile ottenere, ad esempio, 3 Volt dalla porta RB0, con la scrittura di un opportuno programma. Realizziamo lo schema elettrico, peraltro molto semplice, della figura 8.

Come si ottengono i 3 Volt

Abilitando perennemente il pin RB0 otterremmo da questo il potenziale di 5 Volt (duty cycle 100%). Ma siccome è nostro scopo quello di ottenere solamente 3 Volt, è ovvio che dovremo sottoporre tale pin ad una cadenza alternata ed estremamente veloce di impulsi. Per conoscere la percentuale del duty cycle da impostare, basta eseguire la proporzione *%ON = (100 x Volt) : 5* dove Volt corrisponde alla tensione efficace che si vuole ottenere in uscita. Sostituendo i valori con quelli dell'esercizio proposto, il risultato finale ci fornisce la percentuale di duty cycle al 60%, ossia l'impulso a livello alto avrà durata pari al 60% dell'intero periodo dell'onda rettangolare generata, mentre l'impulso basso avrà durata (per differenza) pari al 40% dell'intero periodo.
In ogni caso, la lunghezza del periodo non è critica; possiamo decidere arbitrariamente quanto lungo deve essere. L'importante è rispettare le percentuali dei cicli di lavoro. Per il nostro esem-

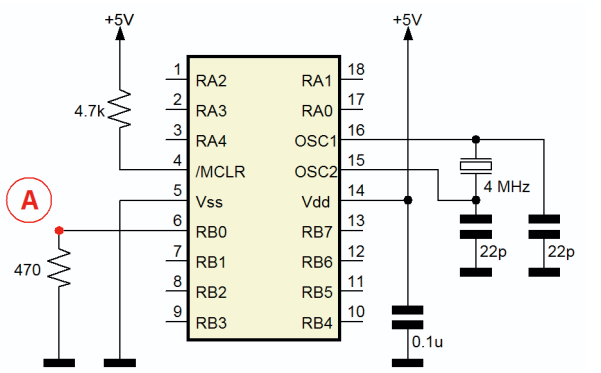


Figura 8 *Schema elettrico per la verifica del PWM*

pio adottiamo una lunghezza di periodo (T) di 100µS (cento microsecondi), per cui il 60% corrisponde a 60 µs ed il 40% corrisponde a 40 µS.

Il software

Questo primo programma è estremamente semplice. Si tratta in pratica di accendere e spegnere, per il tempo sopra calcolato, il piedino RB0. Esaminiamo il relativo software:

```
program PWM01
main:
   trisb=0 'PORTB in Output
   portb=0 'Azzera portb
      while true
      portb.0=1 'RBO ON
      delay_us(60) 'T-ON
      portb.0=0 'RBO OFF
      delay_us(40) 'T-OFF
   wend
end.
```

Verifica

Colleghiamo il tester, in posizione di Volmetro, al punto A dello schema in questione. Leggeremo esattamente 3 Volt. Attenzione però, questa tensione è solo un'illusione "elettrica". Il circuito sta simulando tale differenza di potenziale, semplicemente alternando i periodi di T-ON e T-OFF. Se vogliamo vedere cosa succede "dietro le quinte", occorre collegare un oscilloscopio, sempre al punto A del circuito. Con nostra sorpresa non vedremo una tensione continua di 3 Volt bensì un **treno di impulsi** costanti, con ampiezza 0-5 Volt che, in rapidissima successione, generano il *valore efficace* cercato. Si tratta quindi di una tecnica veramente intelligente che permette di risolvere tanti problemi nel migliore dei modi.

UNA LAMPADA CON QUATTRO GRADI DI LUMINOSITÀ

Ecco un interessante prototipo, che può trovare sicuramente applicazione in un appartamento o un ufficio. Si tratta di una lampada di potenza a 12 Volt, la cui luminosità può essere variata alla pressione di quattro pulsanti n/a. In particolare, il primo pulsante spegne la lampada, il secondo imposta una illuminazione pari al 33%, il terzo imposta una illuminazione pari al 66% ed il quarto la illumina al pieno della sua potenza, quindi al 100%. Realizziamo pertanto lo schema proposto in figura 9.

Analisi procedurale

È il classico circuito a pilotaggio PWM, con cui

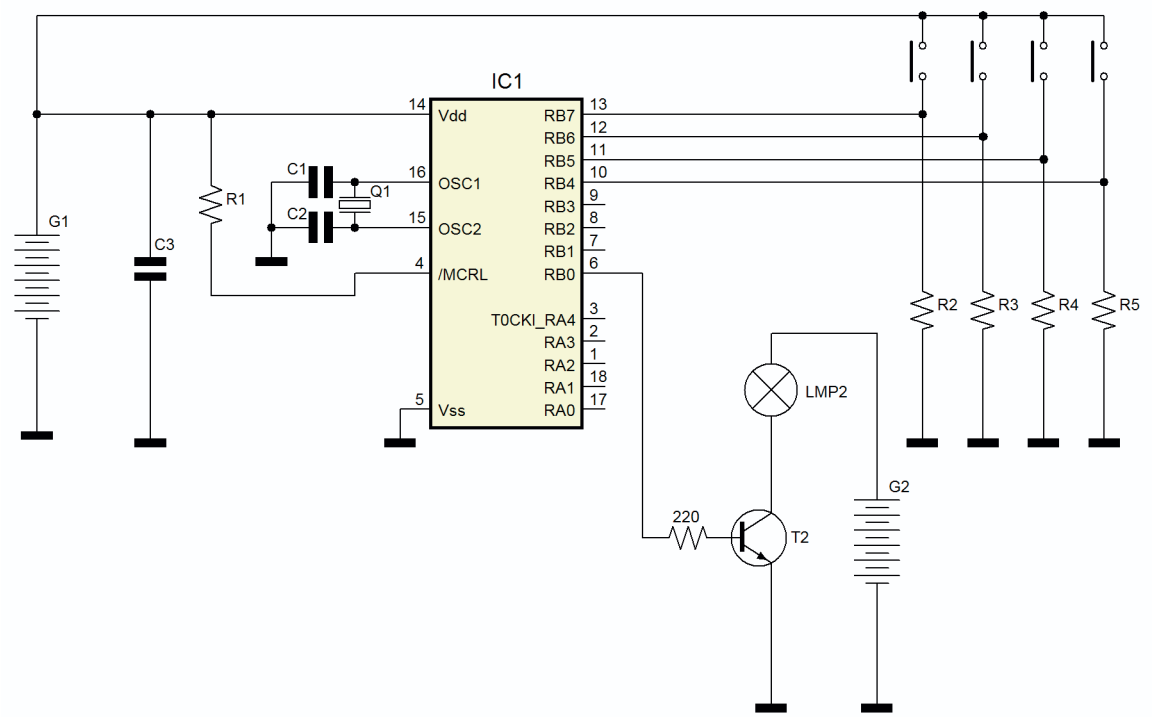


Figura 9 Schema elettrico del variatore di luminosità a PWM

si dosa la luminosità (e quindi la potenza) di una lampada, non agendo sull'intrinseco voltaggio della stessa, ma generando un treno di impulsi proporzionale alla resa desiderata.
Il compito del software è innanzitutto quello di rilevare la pressione di uno dei quattro tasti abilitati. Di conseguenza vengono stabiliti ed assegnati i relativi tempi di duty cycle, ossia di T-ON e T-OFF. Infine viene generata la cadenza relativa ai valori calcolati, per il pilotaggio del transistor di potenza.

Il software

Anche stavolta il software è molto intuitivo. Esaminiamo insieme il listato 1.
Dopo la definizione della funzionalità della PORTB e suo azzeramento, il programma, attraverso gli statements *IF THEN*, controlla lo stato delle quattro porte RB4, RB5, RB6, RB7. A seconda di quale tasto è premuto, avviene l'assegnazione di opportuni valori numerici alle variabili T_ON e T_OFF, che contengono i tempi di attesa per gli stati logici alto e basso. Questi tempi vengono infine attuati dai due loop, che determinano anche la saturazione e l'interdizione del transistor di potenza. Si è scelto di ripetere tante volte il ciclo di attesa, rappresentato dalla funzione *delay_us(1)* in quanto, nella presente release (vers. 1.1.6.0), la suddetta non supporta ancora la parametrizzazione delle variabili. In altre parole non è ancora possibile adottare la seguente clausola: *delay_us(T_ON)*.
Speriamo vivamente che in future versioni questa funzionalità venga implementata.

Collaudo

Non appena si dà tensione al circuito, la lampadina risulta spenta, in quanto inizialmente è stato programmato un valore di duty cycle pari allo 0%. Provando a premere gli altri pulsanti, si noterà la variazione di luminosità della lampada.
Consigliamo vivamente di collegare un oscilloscopio alla porta RB0 durante il collaudo, per comprendere come il PIC genera impulsi di diversa larghezza.

LUCE DI CORTESIA

Realizziamo adesso questo prototipo di luce di cortesia, molto utile in automobile.
All'accensione del circuito la lampada resta spenta. Premendo un pulsante, la lampada si illumina e resta in tale condizione per 20 secondi.

[Listato 1]

```
program PWM02
' Variatore di luce a PWM con 4 pulsanti
dim T_ON as byte
dim T_OFF as byte
dim k as byte

main:
    portb=0 'Azzera portb
    trisb=%11110000 'Definisce PORTB
    T_ON=0
    T_OFF=100 'La lampada e' spenta
    while true
        '---Imposta Duty Cicle 0%-----
        if portb.4=1 then
            T_ON=0
            T_OFF=100
        end if
        '--- Imposta Duty Cicle 33%-----
        if portb.5=1 then
            T_ON=33
            T_OFF=67
        end if
        '--- Imposta Duty Cicle 66%-----
        if portb.6=1 then
            T_ON=67
            T_OFF=33
        end if
        '--- Imposta Duty Cicle 100%-----
        if portb.7=1 then
            T_ON=100
            T_OFF=0
        end if
        '----Pilotaggio PWM---
        portb.0=1 'Accende lampada
        for k=1 to T_ON
          delay_us(1) '1us ripetuto n volte
        next k
        portb.0=0 'Spegne lampada
        for k=1 to T_OFF
          delay_us(1) '1us ripetuto n volte
        next k
    wend
end.
```

Trascorso tale periodo si spegne, diminuendo gradatamente e dolcemente (in 10 secondi) la propria potenza senza una brusca interruzione. Realizziamo per lo scopo lo schema di cui alla figura 11.

Il software

Esaminiamo il software riportato nel listato 2.
La temporizzazione degli impulsi avviene grazie alle due variabili t_on e t_off. Quando l'una incrementa, l'altra decrementa, cosicché varia anche la luminosità.

[Listato 2]

```
program PWM03
' Luce di cortesia
dim duty as byte
dim k as byte
dim ripete as byte
dim t_on as byte
dim t_off as byte

main:
    portb=0 'Azzera portb
    trisb=%00000001 'Definisce PORTB
    while true
        if portb.0=1 then 'Se premo pulsante
            portb.2=1 'Accende lampada
            delay_ms(20000) 'Attesa 20 sec.
            for duty=0 to 100
                t_on=100-duty
                t_off=duty
                for ripete=1 to 10
                    '-----Avvia PWM-------
                    portb.2=1 'Lampada ON
                    for k=1 to t_on    'T-ON
                        delay_us(100)
                    next k
                    portb.2=0 'Lampada OFF
                    for k=1 to t_off 'T-OFF
                        delay_us(100)
                    next k
                next ripete
            next duty
            portb.2=0 'Lampada OFF
        end if
    wend
end.
```

Al momento della pressione del tasto, la lampada resta illuminata per 20 secondi. Poi gradualmente diminuisce la sua intensità sino a quando si spegne del tutto. Se si vuol cambiare il periodo di spegnimento, occorre modificare il valore del ciclo rappresentato dalla variabile *ripete*.

LUCI NATALIZIE

Molti anni fa, le prime luci elettriche per l'albero di natale lampeggiavano grazie all'intermittente, un circuito primitivo che permetteva alle piccole lampadine di accendersi e spegnersi.
Tale apparecchio generava però, nelle immediate vicinanze, piccoli disturbi elettrici, dovuti proprio alla commutazione interna.
Con il corso degli anni tale circuito venne implementato direttamente nel cordone di alimentazione della treccia luminosa, risparmiando così spazio e costi. L'effetto ottenuto era in ogni caso solamente quello del lampeggio: lampade che si accendevano e spegnevano, senza mezze misure.

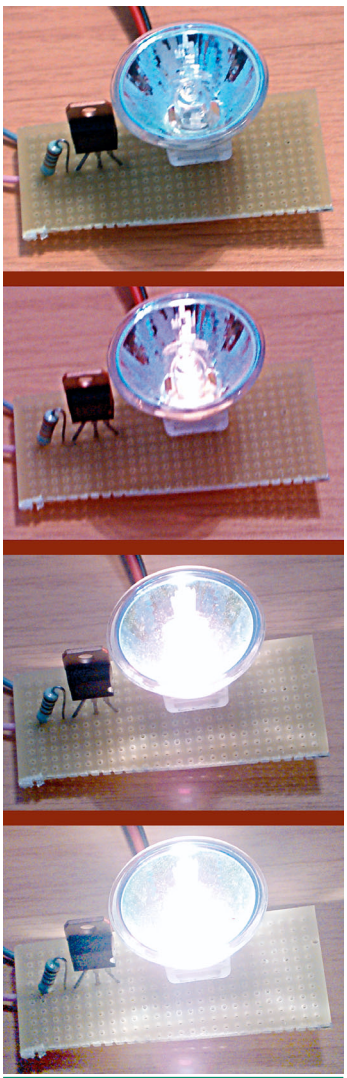


Figura 10 *Le quattro condizioni della lampada a PWM*

Oggi i tempi sono cambiati. Grazie all'adozione di piccolissimi microcontrollori integrati, si è riusciti non solo a programmare il gioco di luci, ossia la particolare sequenza seguita dalle piccole lampadine, ma addirittura a modificarne la luminosità, utilizzando la tecnica del PWM e riuscendo ad ottenere sbalorditivi effetti di transizioni e dissolvenze.
Vediamo come predisporre un progetto che ci consenta di ottenere un gioco di luci natalizio, veramente

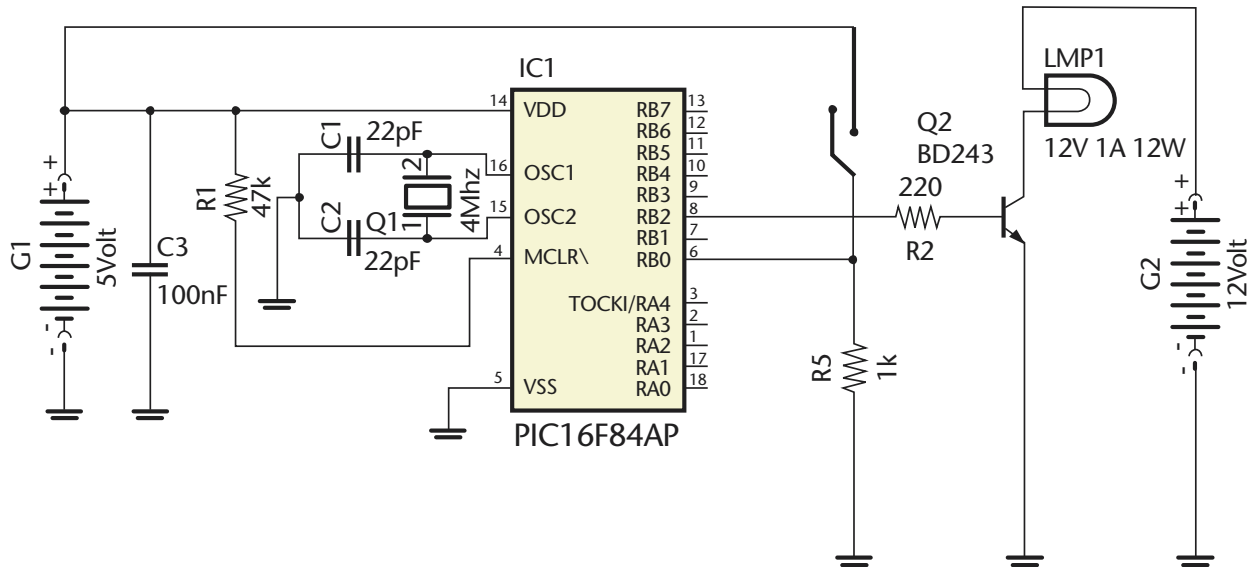


Figura 11 *Schema elettrico delle luci di cortesia*

molto grazioso e divertente.

Schema elettrico

Seguite lo schema elettrico della figura 12. Didatticamente abbiamo previsto solo quattro diodi Led collegati rispettivamente alle porte RB0, RB1, RB2 e RB3. Idealmente però potete sostituire tali utilizzatori da trecciole illuminate,

[Listato 3]

```
program PWM04
' Luci di Natale

dim duty as byte 'Percentuale duty cicle
dim k as byte 'Variabile di comodo
dim t_on as byte 'Tempo in ON
dim t_off as byte 'Tempo in OFF
dim i as byte 'Variabile di comodo
'Array dei Giochi
const led as byte[10]=(1,2,4,8,3,6,12,7,14,15)

main:
   portb=0 'Azzera portb
   trisb=0 'PORTB in OUTPUT
   while true 'Ciclo infinito
      for i=0 to 9
         '----Accensione Graduale
         for duty=1 to 100
            t_on=duty
            t_off=100-duty
            '-----Avvia PWM-------
            portb=led[i] 'Led ON
            for k=1 to t_on    'T-ON
               delay_us(50)
            next k
            portb=0 'Led OFF
            for k=1 to t_off 'T-OFF
               delay_us(50)
            next k
         next duty
         '----Spegnimento Graduale
         for duty=100 to 1 step -1
            t_on=duty
            t_off=100-duty
            '-----Avvia PWM-------
            portb=led[i] 'Led ON
            for k=1 to t_on    'T-ON
               delay_us(50)
            next k
            portb=0 'Led OFF
            for k=1 to t_off 'T-OFF
               delay_us(50)
            next k
         next duty
      next i
   wend
end.
```

composte da tante lampade, previo pilotaggio da parte di opportuni transistor.

Il Software

Esaminiamo il programma, riportato nel listato 3, passando successivamente ai commenti.
I vari giochi di luce sono memorizzati nel vettore *led[i]*, permettendo un riutilizzo del codice sottostante. Si tratta in sostanza di una serie di cicli nidificati, che modificano proporzionalmente il duty cycle degli impulsi erogati ai diodi led, consentendone la variazione luminosa con continuità.

PILOTIAMO UN MOTORE

La tecnica del PWM si presta egregiamente a pilotare un motore elettrico consentendone la variazione di velocità. Due sono gli aspetti che maggiormente spiccano usando tale metodo:

1. Si riduce la dissipazione di calore poiché i transistor sono utilizzati come interruttori e non come amplificatori.
2. Anche a bassi regimi il motore conserva un'elevata coppia, dal momento che, anche con duty cycle minimo, esso è sempre alimentato con la tensione massima di alimentazione.

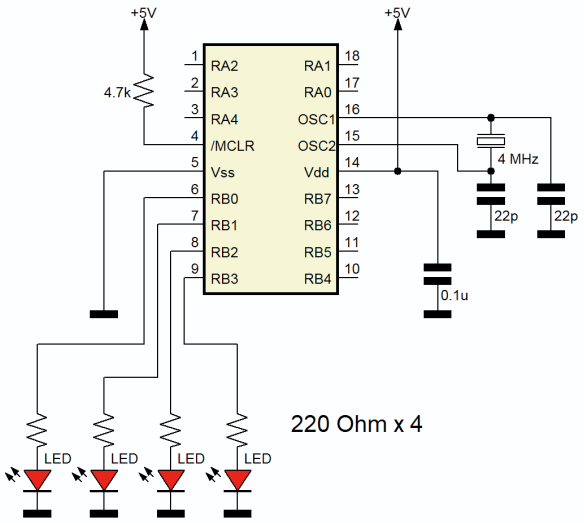
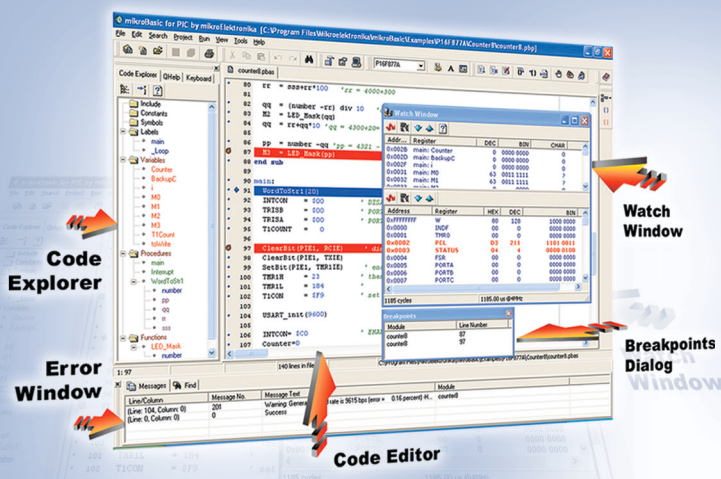


Figura 12 Schema elettrico delle luci di Natale

Software Mikrobasic



Un potente compilatore Basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

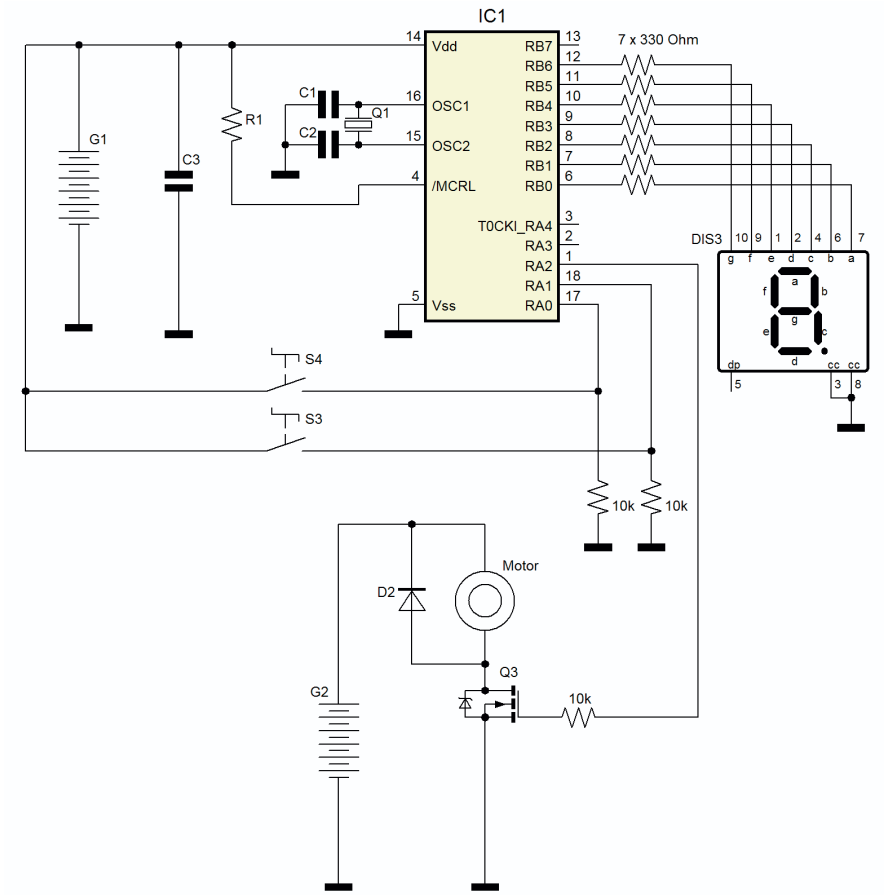


Figura 13 *Schema elettrico del variatore per motore*

Il nostro progetto

È sicuramente uno dei più interessanti progetti proposti nel nostro corso. Occorre realizzare un circuito dotato di due pulsanti ed un motore. I pulsanti sono utilizzati a mo' di cambio marce. La pressione del primo tasto aumenta la velocità del motore mentre il secondo tasto diminuisce tale velocità.

Il sistema prevede dieci velocità diverse, ossia dieci step che fanno variare il regime del motore da un minimo (fermo) ad un massimo. Il passo impostato è visualizzato attraverso un display a 7 segmenti, che in tal modo mostra i numeri da 0 a 9.

Schema elettrico

Lo schema, visibile in figura 13, riporta i componenti fondamentali del progetto. La PORT-B è configurata completamente in output e serve esclusivamente a pilotare il display a 7 segmenti.

La PORT-A ha invece un funzionamento promiscuo, in quanto i pin RA0 e RA1 fungono da ingressi per i due pulsanti, mentre il pin RA2 pilota con impulsi regolari il motore, attraverso il mosfet di potenza IRF540 o equivalente. Il diodo in controfase collegato in parallelo al motore elimina le extratensioni, potenzialmente pericolose per il transistor. È possibile in ogni caso collegare una grossa lampada al posto del motore.

Il software

Un piccolo gioiello di programmazione, che in poche righe riesce ad eseguire tante funzionalità. Si inizia con la definizione delle variabili di lavoro e del vettore contenente le codifiche per la visualizzazione dei dieci numeri sul display. Si passa quindi alla definizione delle porte di comunicazione e relativo azzeramento.

Vitale importanza riveste il flag *premuto*, che ha il compito di stabilire se un tasto è stato appunto premuto o meno.

Questa variabile consente il controllo della routine di incremento e decremento, evitando il fastidioso effetto del *rimbalzo* e del *repeat*. Il valore acquisito dallo *step* è così assegnato alla variabile *marcia* che, assieme alle variabili *t_on* e *t_off*, si occupano del pilotaggio del motore attraverso il dosaggio della larghezza dell'impul-

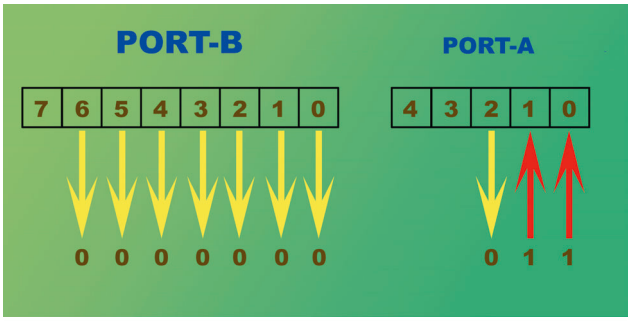


Figura 14 *Funzioni svolte dalla PORT-A e dalla PORT-B*

[Listato 4]

```
program PWM05   ' Motore

    const display as byte[10]= (63,6,91,79,102,109,125,7,127,111)
    const test as byte[7]=(1,2,4,8,16,32,64)
    dim marcia as byte 'Contiene la marcia impostata
    dim premuto as byte 'Flag che indica tasto premuto
    dim k as byte 'Variabile di comodo
    dim t_on as byte 'Durata impulso ON
    dim t_off as byte 'Durata impulso OFF

main:
    porta=0 'Azzera porta
    portb=0 'Azzera portb
    trisa=00011 'Definisce port-a
    trisb=0 'definisce PORTB in output
    marcia=0 'Imposta la marcia 0 (motore fermo)
    t_on=0 'Durata inpulso ON iniziale
    t_off=90 'Durata inpulso OFF iniziale
    premuto=0 'Si suppone che non si preme nulla
    while true
        portb=display[marcia] 'Visualizza marcia
        '----Antirimbalzo e antirepeat------
        if (porta.0=0) and (porta.1=0) then
            premuto=0 'I tasti non sono premuti
        end if
        '----Se si preme tasto incremento--------
        if (porta.1=1) and (marcia<9) and (premuto=0) then
            marcia=marcia+1
            t_on=t_on+10
            t_off=t_off-10
            premuto=1
        end if
        '----Se si preme tasto decremento--------
        if (porta.0=1) and (marcia>0) and (premuto=0) then
            marcia=marcia-1
            t_on=t_on-10
            t_off=t_off+10
            premuto=1
        end if
        '--PILOTA IL MOTORE CON IL PWM-----
        porta.2=1 'Impulso ON
        for k=1 to t_on
            delay_us(1)
        next k
        porta.2=0 'Impulso OFF
        for k=1 to t_off
            delay_us(1)
        next k
    wend
end.
```

so fornito. Studiatelo a fondo e cercate di comprendere bene tutte le routine utilizzate, in quanto si tratta di uno dei primi software aperti, e costituisce la base iniziale per la realizzazione di prototipi professionali ed industriali *[Listato 4*].
In figura 14 riportiamo la funzione svolta da ciascun piedino delle porte A e B.

Raccomandazioni per l'uso dei motori

Di seguito elenchiamo alcune piccole precauzioni da prendere durante l'utilizzo dei motori, onde evitare eventuali problemi tecnici:

- Usate sempre il diodo di blocco posto in antiparallelo al motore per impedire la formazione di pericolose extratensioni.
- Dotate sempre il transistor di una adeguata aletta di raffreddamento.
- Non bloccate MAI manualmente l'attività del motore, pena l'immediata distruzione del transistor.
- Utilizzate una frequenza di PWM sempre superiore a 20 Khz: in caso contrario potrebbe udirsi all'interno del motore stesso una nota acustica di frequenza equivalente al periodo calcolato.

CONCLUSIONI

Bene, purtroppo anche questa puntata è terminata. Speriamo di essere riusciti a dare gli spunti necessari alla realizzazione di utili apparecchiature.
Ricordate sempre che il miglior modo per imparare a programmare con disinvoltura è quello di provare e riprovare ancora. Solamente con la costanza e con la perseveranza si potranno acquisire alla perfezione molti metodi e trucchi della programmazione.

Vi diamo appuntamento alle prossime puntate, ricche di novità e di nuove idee, non prima però di avervi augurato un buon lavoro.

Scheda easyPIC3



La rivoluzionaria scheda di sviluppo per PICmicro

- Programmatore USB2.0 on-board
- Tastiera a 32 tasti
- 32 LED per il monitoraggio degli I/O
- 4 cifre LED a 7 segmenti
- Predisposizione per moduli LCD alfanumerici
- Predisposizione per moduli LCD grafici*
- Predisposizione per comunicazione RS232
- Predisposizione per tastiera PS2*
- Predisposizione per sensore di temperatura DS1820
- Supporto per tutte le famiglie PIC (anche PIC10F)*
- Predisposizione per comunicazione USB*
- Alimentazione esterna o via USB
- Fornita con 16F877

* novità rispetto a easyPIC2

Ordinala subito su **www.farelettronica.com** oppure telefona allo 02.66504794

Quinta parte
n° 243 - Settembre 2005
PWM con il PIC

Sesta parte
n° 244 - Ottobre 2005
Musika maestro! Generazione di melodie, note, effetti sonori e musica con il PIC

Settima parte
n° 245 - Novembre 2005
Pilotiamo i motori passo-passo

Mikrobasic per PICmicro

Questa volta parliamo di suono, ovvero di come poter utilizzare il compilatore MikroBasic della MikroElektronika, in abbinamento al nostro microcontrollore PIC16F84, per produrre note, melodie e motivi musicali al fine di rendere i nostri progetti più accattivanti e professionali.

In questa puntata studiamo qualcosa di diverso dal solito. Come si legge dal titolo, parliamo appunto di musika (proprio con la lettera k) in onore del nostro Mikrobasic della mikroElektronika, che da questa puntata ci offre la nuova versione 2.0 (almeno alla data della stesura dell'articolo), piena di novità e di molte potenzialità. Invitiamo pertanto i lettori a scaricare l'ultima release dal sito omonimo.

SUONI, DOVE E QUANDO

Sono moltissime le applicazioni elettroniche che usano i suoni come feetback per l'utente: ad esempio per avvertire un operatore del verificarsi di un evento oppure per richiamare l'attenzione di un addetto al controllo di un quadro elettrico, per un problema tecnico.
Insomma, dove le periferiche di visualizzazione (display, led, lcd, lampade, ecc) non riescono a dare in pieno il loro contributo, intervengono con successo, con un ruolo insostituibile, le periferiche sonore, la maggior parte delle quali sono supportate da piccoli altoparlanti e da buzzer.
In linea di massima non si può collegare un altoparlante di bassa impedenza direttamente ad un pin del microcontrollore, ma occorre usare qualche componente in più per ottenere una potenza maggiore senza, per questo, "appesantire" il lavoro svolto dal PIC.

Suono, nota e musica

Con il termine di **suono** si intende una vibrazione meccanica che, propagandosi in un mezzo elastico, ad esempio l'aria, provoca una percezione da parte del nostro sistema uditivo. Per essere tale, la sua frequenza deve essere compresa tra 20 Hz e 20.000 Hz. (almeno per l'essere umano). A seconda della frequenza, si usa catalogare il suono in sette **note** (DO, RE, MI, FA SOL, LA, SI) distribuite tra diverse *ottave*.
Infine la **musica** è *l'arte del combinare* assieme le note al fine di produrre una sensazione piacevole di ascolto e, chiaramente, tale fenomeno è puramente soggettivo.
Non addentrandoci comunque in discorsi troppo tecnici, da un punto di vista acustico: diciamo solo che un suono è caratterizzato da una specifica forma d'onda; in altre parole la variazione che segue l'ampiezza nel tempo disegna effettivamente un grafico, di cui si possono studiarne le caratteristiche.

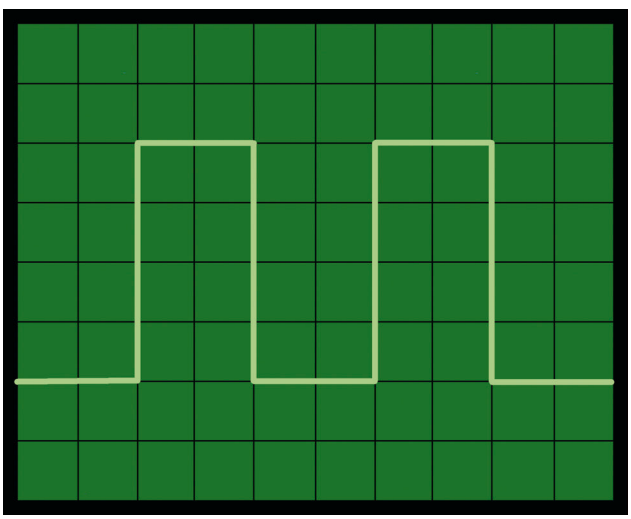


Figura 1 Tipica onda rettangolare

MUSIKA MAESTRO! Generazione di melodie, note, effetti sonori e musica con il PIC



di Giovanni Di Maria
(g.dimaria@farelettronica.com)

Dal momento che il nostro microcontrollore è capace di generare solamente due livelli di tensione (5 Volt e 0 Volt), l'unica forma d'onda che possiamo tirare fuori è **l'onda rettangolare**, ossia un treno di impulsi che, se supera una certa frequenza, genera un rumore udibile dall'uomo. Eseguiremo svariati esempi per comprendere bene tutto ciò.

T = 1 / F

Vediamo adesso, con la teoria, come sia possibile ottenere dal nostro micro, tutti i suoni udibili (e anche quelli non udibili …).
Partiamo da un semplice esempio: occorre generare una nota di 500 Hz per la durata di un secondo. Cosa vuol dire? Semplicemente che in un "solo" secondo la sua forma più semplice si ripete completamente (dall'inizio alla fine) per 500 volte. Dal momento che la nostra nota ha una forma *puramente* rettangolare, si ripeterà nel tempo una commutazione da uno stato logico ALTO ad uno stato logico BASSO per 500 volte. Se poi vogliamo sapere quanto tempo dura un singolo evento (denominato anche periodo o ciclo) basterà dividere 1 secondo per 500 (da qui la formula T=1/F) e nel nostro caso otterremo il risultato 0,002 secondi (ossia 2 millesimi di secondo, di cui 1 millesimo in stato ON e 1 millesimo in stato OFF). Con qualche grafico il concetto viene meglio chiarito (figura 2).

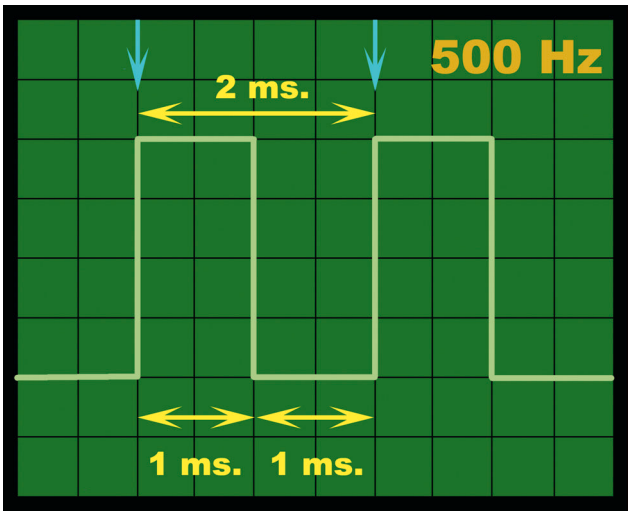


Figura 2 *Onda rettangolare di 500 Hz*

Un'onda rettangolare di 500 Hz ha un periodo di 1/500=0,002 secondi, di cui la parte attiva dura solamente 1 ms. e altrettanto la parte "a riposo". Se non vi fosse questo alternarsi di stati non potrebbe esistere una produzione di vibrazione e quindi di suono.

	OTTAVE					
NOTA	2	3	4	5	6	7
Do	65,4	130,8	261,6	523,3	1046,5	2093
Do#	69,3	138,6	277,2	554,4	1108,7	2217,5
Re	73,4	146,8	293,7	587,3	1174,7	2349,3
Re#	77,8	155,6	311,1	622,3	1244,5	2489
Mi	82,4	164,8	329,6	659,3	1318,5	2637
Fa	87,3	174,6	349,2	698,5	1396,9	2793,8
Fa#	92,5	185	370	740	1480	2960
Sol	98	196	392	784	1568	3136
Sol#	103,8	207,7	415,3	830,6	1661,2	3322,4
La	110	220	440	880	1760	3520
La#	116,5	233,1	466,2	932,3	1864,7	3729,3
Si	123,5	246,9	493,9	987,8	1975,5	3951,1

Tabella 1 *Nomi e frequenze delle note (in Hertz)*

GENERIAMO IL NOSTRO PRIMO SUONO

Bene, siamo pronti per il primo esperimento, ossia quello della generazione di un suono di

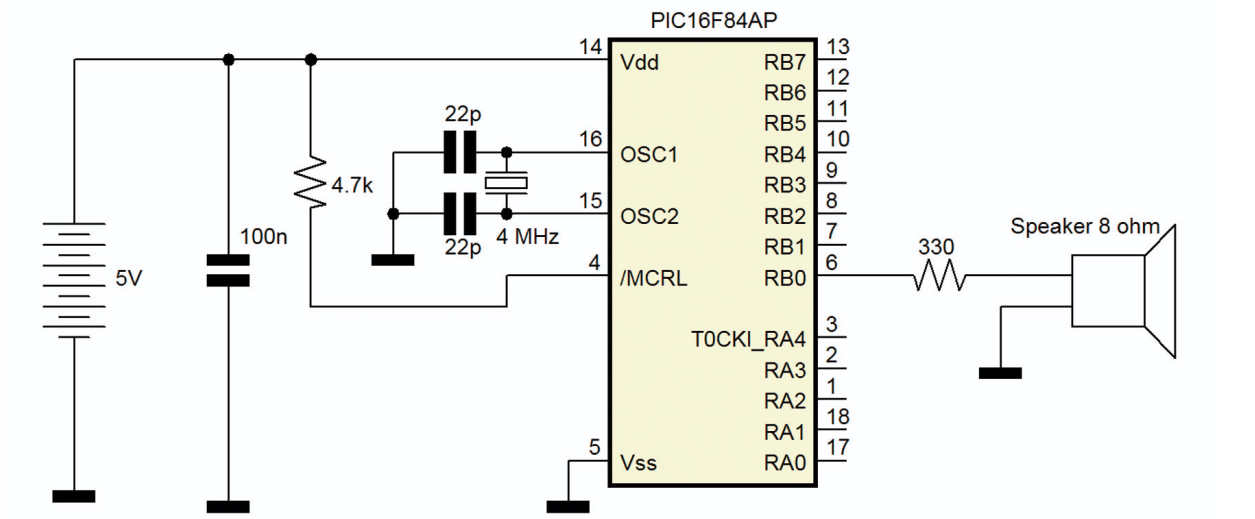


Figura 3 *Schema elettrico del prototipo sonoro*

1000 Hz (1 Khz) della durata di 5 secondi. Approntiamo per lo scopo lo schema di cui alla figura 3.

Un'onda quadra di 1000 Hz (ossia di 1000 cicli al secondo) ha un periodo base di 1 ms. ossia un intero ciclo dura 1 ms. (1 / 1000).
In un periodo completo si hanno due semiperiodi: il primo di livello logico alto (5 volt) che corrisponde ad una durata di 0,5 ms. (500 microsecondi) ed il secondo di livello logico basso (0 volt) anch'esso della durata di 0,5 ms. Tale periodo, definito anche **tick**, corrisponde ad un solo ciclo di lavoro. Per ottenere una nota occorre ripeterlo tante volte nell'arco di tempo. Pertanto se il ciclo, in questo caso, dura solamente 1 millisecondo, occorre riprodurlo per 5000 volte per ottenere un suono della durata di 5 secondi. Infatti 1 ms. x 5000 volte = 5000 ms. (5 secondi).
In figura 4 possiamo vedere l'onda quadra generata con i relativi tempi di attesa.

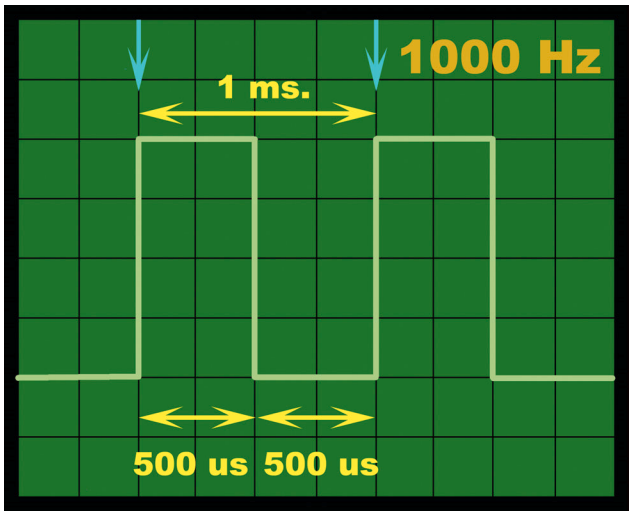


Figura 4 *Onda quadra di 1000 Hz*

Andiamo adesso a scrivere il relativo listato, completo di commenti, copiando quanto riportato nel **listato 1**.

[Listato 1]

```
rem   Generazione di un SUONO da 1000 Herz
rem   della durata di 5 secondi

program suono01

dim k as word       'Indice del ciclo
dim cicli as word   'Durata in tick

main:
trisb=0    'Definisce la PORTB in uscita

portb=0    'Azzera la PORTB
cicli = 1000 * 5   'Ripetizione che assicura
                   '5 secondi
for k=1 to cicli
    portb.0=1      ' Bit ALTO
    delay_us(500)
    portb.0=0      ' Bit BASSO
    delay_us(500)
next k
end.
```

Dopo aver compilato il sorgente, scarichiamo il relativo file prodotto (con estensione HEX) all'interno del PIC e, dando alimentazione al circuito, udiremo chiaramente in altoparlante una nota di 1000 Hz della durata di 5 secondi esatti.
Se disponete di un oscilloscopio, collegandolo direttamente tra PIN RB0 e massa, potrete osservare l'oscillogramma di un'onda quadra con i tempi precedentemente descritti.
N.B. Il suono che il microcontrollore produce possiede un livello di ampiezza (volume) molto basso. Se si vuole aumentare l'intensità occorre amplificare il segnale.

SCALA MUSICALE

Il prossimo programma, peraltro molto semplice, permette l'ascolto in altoparlante di una scala musicale, formata solamente dalle note Do, Re, Mi, Fa, Sol, La, Si, Do.
Ci riferiamo sempre allo schema base riprodotto in figura 3. Ogni nota deve avere la durata di un secondo, per cui tutto il programma termina la sua esecuzione in 8 secondi.
Dal momento che ogni nota è di frequenza diversa da ogni altra, il numero dei cicli (per determinare la durata di 1 secondo) dovrà essere calcolata proporzionalmente, come nell'esempio precedente. La tabella 2 mostra i valori da inserire nel **listato 2**.

[Listato 2]

```
rem       Scala Musicale

program suono02

dim k as word

main:

trisb=0       'Definisce la PORTB in uscita
portb=0       'Azzera la PORTB
delay_ms(1000) 'Attesa iniziale di un secondo
rem -----GENERA IL DO-----
for k=1 to 262
    portb.0=1       ' Bit ALTO
    delay_us(1908) ' 3817 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1908) ' 3817 / 2
next k
rem -----GENERA IL RE-----
for k=1 to 294
    portb.0=1       ' Bit ALTO
    delay_us(1700) ' 3401 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1700) ' 3401 / 2
next k
rem -----GENERA IL MI-----
for k=1 to 330
    portb.0=1       ' Bit ALTO
    delay_us(1515) ' 3030 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1515) ' 3030 / 2
next k
rem -----GENERA IL FA-----
for k=1 to 349
    portb.0=1       ' Bit ALTO
    delay_us(1432) ' 2865 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1432) ' 2865 / 2
next k
rem -----GENERA IL SOL-----
for k=1 to 392
    portb.0=1       ' Bit ALTO
    delay_us(1275) ' 2551 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1275) ' 2551 / 2
next k
rem -----GENERA IL LA-----
for k=1 to 440
    portb.0=1       ' Bit ALTO
    delay_us(1136) ' 2273 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1136) ' 2273 / 2
next k
rem -----GENERA IL SI-----
for k=1 to 494
    portb.0=1       ' Bit ALTO
    delay_us(1012) ' 2024 / 2
    portb.0=0       ' Bit BASSO
    delay_us(1012) ' 2024 / 2
next k
rem -----GENERA IL DO-----
for k=1 to 523
    portb.0=1       ' Bit ALTO
    delay_us(956)  ' 1912 / 2
    portb.0=0       ' Bit BASSO
    delay_us(956)  ' 1912 / 2
next k

end.
```

Nota	Frequenza (Hz)	Frequenza arrotondata	Durata di un periodo (microsecondi)
DO	261,6	262	3817
RE	293,7	294	3401
MI	329,6	330	3030
FA	349,2	349	2865
SOL	392,0	392	2551
LA	440,0	440	2273
SI	493,9	494	2024
DO	523,3	523	1912

Tabella 2 *Frequenza e durata di ogni nota*

Il programma è di una semplicità estrema. Ogni ciclo, governato dalla variabile *k*, *accende* e *spegne* la PORTB.0 collegata all'altoparlante, in modo da generare la vibrazione e quindi la nota. I cicli di attesa (delay_us) assicurano l'esatta temporizzazione tra uno stato logico alto e uno basso e tale alternanza è ripetuta sino a che la durata di una nota sia esattamente pari ad 1 secondo.

SIMULIAMO LA CORNETTA TELEFONICA

In questo prossimo esercizio realizziamo la simulazione del tono telefonico della centrale, che invita l'utente a comporre il numero da chiamare (il dial tone). In pratica dobbiamo generare una sequenza ripetitiva di toni di 425 Hz, come si vede dal flow chart rappresentato in figura 5.

Prendendo sempre come riferimento lo schema base riprodotto in figura 3, il sorgente Basic lo trovate nel **listato 3**.

Fantastico, sembra proprio di avere tra le mani un vero telefono. Commentiamo le parti salienti. Dopo aver configurato la variabile e impostata in output la PORTB, all'interno di un ciclo infinito (WHILE/WEND) sono generati i toni di cadenza e le pause.
Riprendiamo un attimo il metodo di generazione di frequenza. Si tratta di applicare "piccoli" concetti di matematica legati al tempo.

[Listato 3]

```
rem       Simulazione Cornetta Telefonica

program suono03

dim k as word     'Variabile di comodo

main:

trisb=0       'Definisce la PORTB in uscita
portb=0       'Azzera la PORTB
while true     'Ciclo infinito
   rem -----GENERA 425 Hz per 200 ms-----
   for k=1 to 85    ' (425 * 200 / 1000)
       portb.0=1        ' Bit ALTO
       delay_us(1176) ' 3817 / 2
       portb.0=0        ' Bit BASSO
       delay_us(1176) ' 3817 / 2
   next k
   rem -----GENERA pausa per 200 ms-----
   delay_ms(200)
   rem -----GENERA 425 Hz per 600 ms-----
   for k=1 to 255    ' (425 * 600 / 1000)
       portb.0=1        ' Bit ALTO
       delay_us(1176) ' 3817 / 2
       portb.0=0        ' Bit BASSO
       delay_us(1176) ' 3817 / 2
   next k
   rem ------GENERA pausa per 1000 ms----
delay_ms(1000)
wend

end.
```

Per generare, ad esempio, una nota di 425 Hz della durata di 200 ms. (1/5 di secondo) dobbiamo seguire questo iter di calcolo:

1. Calcolare il periodo (T=1/F) e quindi: **T = 1/425** ossia **T=0,002353 sec.**
2. Convertire il periodo in microsecondi, moltiplicando semplicemente il risultato per 1.000.000 (un milione) quindi: **P = 0,002353 * 1.000.000 = 2353** microsecondi.
3. Calcolare il tempo dell'onda rettangolare in ON e il tempo in OFF semplicemente dividendo per due tale periodo, quindi avremo: **T-ON e T-OFF = 2353 /2 = 1176** microsecondi.
4. I due stati logici appena calcolati devono

ripetersi per il tempo desiderato. Sapendo che, nell'esempio, un periodo completo dura solamente 0,002353 secondi, con una semplice proporzione si calcola quanti periodi occorrono per coprire il tempo necessario, quindi: **1 : 0,002353 = x : 0,2 e x = (1 * 0,2) / 0,002353 = 85** (periodi da ripetere).

Se riuscirete a far "vostra" questa procedura potrete ottenere qualsiasi frequenza e qualsiasi durata di una nota.

Esercizio per i lettori

Ora che abbiamo visto come si generano i toni, provate a programmare un simulatore di cornetta telefonica, stavolta che riproduca il tono telefonico di "linea occupata".
La nota generata è di 425 Hz con il seguente flusso: nota di 500 ms. e silenzio di 500 ms. Il tutto si ripete all'infinito.
Prima di procedere all'esercitazione consiglio "vivamente" di leggere il resto dell'articolo: potreste trovare alcuni metodi per risparmiare lavoro…

IMPARIAMO AD USARE LE PROCEDURE

Il modo di programmare (ci riferiamo anche alle precedenti puntate) è sicuramente funzionale, ma potrebbe creare qualche problema di "lunghezza del codice" nel caso si debba ripetere tante volte un certo lavoro. Ci riferiamo, ad esempio, al listato 2 delle scale musicali. Come avrete notato, all'interno di ciascun blocco FOR / NEXT è presente il codice per generare una nota. Immaginate adesso che la nostra necessità sia quella di produrre non 7 note ma molte di più. Il codice ottenuto sarebbe enorme, si potrebbe perdere facilmente il filo del discorso ma, soprattutto, la parte da copiare all'interno del PIC potrebbe non entrarvi più.
Ci vengono allora in aiuto le **procedure**, che sono come delle piccole scatole, caratterizzate da un nome (proprio come le variabili) che, a differenza, non contengono un valore numerico bensì un "pezzo" di programma, di codice da eseguire.
Con esse si possono pertanto "creare" dei nuovi comandi non presenti all'interno del compilatore. Invitiamo il lettore ad approfondire molto questo argomento poiché di importanza "vitale" ai fini della programmazione.
Ricodifichiamo nuovamente il programma della scala musicale, stavolta con una variazione: dobbiamo produrre una scala ascendente ed una discendente (DO, RE, MI, FA, SOL, LA, SI, DO, SI, LA, SOL, FA, MI, RE, DO) riferendoci sempre allo schema elettrico di figura 3. Esaminiamo il software scritto e poi passiamo ai commenti **(listato 4).**

Troviamo una grossa novità: il costrutto SUB PROCEDURE / END SUB. Come detto prima si tratta di una sorta di scatola che racchiude una parte di codice che, di solito, si deve ripetere varie volte nel programma. Utilizzando le procedure si usa dire che la programmazione diventa *modulare*.
Vediamo come esse funzionano. Innanzitutto ogni procedura deve essere "dichiarata" prima della label MAIN. All'interno di questa, possiamo scrivere tutto il codice che vogliamo ma, in ogni caso, deve trattarsi di un insieme di istruzioni generalmente "ripetitive".
Nell'esempio proposto, dopo la label MAIN, vengono elencate le note, in sequenza di riproduzione, e questa sequenza potrebbe essere completamente modificata se le esigenze di programmazione lo richiedessero.
Con le procedure abbiamo quindi creato dei nuovi comandi (Do1, re1, mi1 e così via). Basterà poi specificare quale "comando" occorre eseguire per avviare la riproduzione della singola nota.

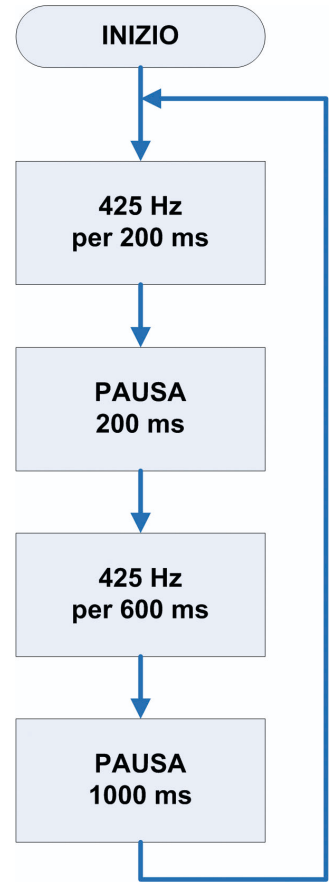


Figura 5 Flow chart del segnale di dial tone

[Listato 4]

```
rem  Scala Musicale che utilizza le PROCEDURE

program Suono04

dim k as word     'Variabile di comodo

rem -----GENERA IL DO-----
sub procedure do1
for k=1 to 262
     portb.0=1       ' Bit ALTO
     delay_us(1908)  ' 3817 / 2
     portb.0=0       ' Bit BASSO
     delay_us(1908)  ' 3817 / 2
next k
end sub
rem -----GENERA IL RE-----
sub procedure re1
for k=1 to 294
     portb.0=1       ' Bit ALTO
     delay_us(1700)  ' 3401 / 2
     portb.0=0       ' Bit BASSO
     delay_us(1700)  ' 3401 / 2
next k
end sub
rem -----GENERA IL MI-----
sub procedure mi1
for k=1 to 330
     portb.0=1       ' Bit ALTO
     delay_us(1515)  ' 3030 / 2
     portb.0=0       ' Bit BASSO
     delay_us(1515)  ' 3030 / 2
next k
end sub
rem -----GENERA IL FA-----
sub procedure fa1
for k=1 to 349
     portb.0=1       ' Bit ALTO
     delay_us(1432)  ' 2865 / 2
     portb.0=0       ' Bit BASSO
     delay_us(1432)  ' 2865 / 2
next k
end sub
rem -----GENERA IL SOL-----
sub procedure sol1
for k=1 to 392
     portb.0=1       ' Bit ALTO
     delay_us(1275)  ' 2551 / 2
     portb.0=0       ' Bit BASSO
     delay_us(1275)  ' 2551 / 2
next k
end sub
rem -----GENERA IL LA-----
sub procedure la1
for k=1 to 440
     portb.0=1       ' Bit ALTO
     delay_us(1136)  ' 2273 / 2
     portb.0=0       ' Bit BASSO
     delay_us(1136)  ' 2273 / 2
next k
end sub
rem -----GENERA IL SI-----
sub procedure si1
for k=1 to 494
     portb.0=1       ' Bit ALTO
     delay_us(1012)  ' 2024 / 2
     portb.0=0       ' Bit BASSO
     delay_us(1012)  ' 2024 / 2
next k
end sub
rem -----GENERA IL DO-----
sub procedure do2
for k=1 to 523
     portb.0=1       ' Bit ALTO
     delay_us(956)   ' 1912 / 2
     portb.0=0       ' Bit BASSO
     delay_us(956)   ' 1912 / 2
next k
end sub

rem ----DA QUI INIZIA IL PROGRAMMA
main:
trisb=0       'Definisce la PORTB in uscita
portb=0       'Azzera la PORTB
do1           ` Richiama la procedura DO1
re1
mi1
fa1
sol1
la1
si1
do2
si1
la1
sol1
fa1
mi1
re1
do1
end.
```

Adottando questo metodo otteniamo un codice molto leggibile e ben organizzato.

UTILIZZIAMO LE FUNZIONI BUILT-IN

Premessa

Il compilatore Mikrobasic è dotato di alcune funzioni "preimpostate" che generano, con *pochissime* righe di programmazione, qualsiasi frequenza.
Di conseguenza tutto ciò che è stato spiegato sino ad ora sarebbe teoricamente inutile: vedremo però che tali funzionalità automatiche di *libreria* hanno alcune limitazioni che ne pregiudicano il corretto utilizzo, pertanto il conoscere l'esatta procedura della generazione del suono, scrivendo in proprio il codice specifico, non solo dà una completa comprensione di quanto accade all'interno del PIC, ma contribuisce soprattutto a risolvere e superare gli ostacoli che queste funzioni di libreria potrebbero causare.

Le funzioni

Le funzioni supportate dal Mikrobasic, almeno nella versione attuale sono due:

```
Sound_init(porta,pin)
Sound_play(periodo/10,n. periodi)
```

La prima predispone il microcontrollore ad accettare su una porta (es. PORTA o PORTB) un dispositivo sonoro di output (altoparlante o buzzer). Tale dispositivo deve essere collegato sul *pin* specificato dal secondo parametro. Per esempio la funzione `Sound_init(PORTB,0)` prepara la PORTB, e precisamente il pin RB0, ala produzione di suoni.

La seconda invece si occupa della vera e propria riproduzione della nota, generando un'onda quadra di opportuna frequenza. Il suo utilizzo potrebbe essere inizialmente un po' complicato. Vediamo come si usa.
Come si vede dalla sintassi della funzione, essa accetta (tra parentesi) due parametri: il primo indica la lunghezza del periodo della nota (in microsecondi) da dividere per 10 e determina pertanto la **frequenza** da generare; il secondo indica invece il numero di periodi che devono comporre la lunghezza della nota e determina quindi la **durata** della stessa. Per esempio la funzione `Sound_play(114,3520)` serve per generare una nota di 880 Hz (LA) di 4 secondi di durata. Un po' macchinoso a dire il vero ma non possiamo avere tutto sul piatto d'argento…
Vediamo comunque come avviene il calcolo dei parametri di tale funzione.

Calcolo dei parametri

Un esempio chiarirà meglio il concetto.
Supponiamo di avere bisogno di una nota di 880 Hz (LA) che duri 4 secondi. Andiamo a calcolare i due parametri.

1. Primo parametro *(frequenza)*
   a. Calcolare il periodo con la formula T = 1 / F (1 / 880); quindi 0,00113636.
   b. Trasformare il risultato in microsecondi, moltiplicandolo semplicemente per un milione, quindi 1136,3636 us.
   c. Dividere tale risultato per dieci e arrotondare, in quanto la funzione lo esige, quindi **114**.
   d. Abbiamo quindi ottenuto il primo parametro.
2. Secondo parametro *(durata)*
   a. Dal momento che la frequenza desiderata è di 880 Hz (cicli al secondo) è ovvio che per durare un secondo tale nota ha bisogno di 880 periodi. Per ottenere tempi maggiori o minori bisogna calcolare proporzionalmente tale dato e in questo caso moltiplicare 880 per 4 (secondi) quindi **3520**.
   b. Abbiamo quindi ottenuto anche il secondo parametro.

Invocando pertanto la funzione `Sound_play(114,3520)` otteniamo un suono della frequenza di 880 Hz della durata di 4 secondi. In ogni caso, per il calcolo dei suddetti parametri, consiglio vivamente l'adozione e l'utilizzo di un programma per la gestione di fogli elettronici, soprattutto per accelerare lo svolgimento delle operazioni matematiche, quando si hanno molte note da gestire.

Limitazioni

Utilizzando le funzioni built-in, abbiamo un vantaggio e due svantaggi. Il vantaggio consiste nel fatto che con una sola riga di codice si può

generare facilmente una nota. Il primo svantaggio consiste nell'impossibilità di riprodurre "tutte" le frequenze udibili dall'uomo, infatti il *range* d'azione spazia da un minimo di 393 Hz ad un massimo di 100.000 Hz. Come vedete il limite superiore può andare benissimo, un po' meno il limite inferiore…
Il secondo svantaggio sta nel fatto che alcune note prodotte subiscono una "leggerissima" stonatura per via di sofisticati calcoli matematici eseguiti all'interno del compilatore.

Suoniamo un "giro di DO"

Adesso facciamo riprodurre al nostro PIC un giro di DO con questa sequenza: DO, MI, SOL, DO2, SOL, MI ripetendosi all'infinito. Ogni nota deve durare 250 ms. Guardando la tabella 2, abbiamo utilizzato le seguenti note:

- DO = 523 Hz
- MI = 659 Hz
- SOL = 784 Hz
- DO2 = 1046 Hz

Ed ecco il listato, peraltro cortissimo **(listato 5)**.

[Listato 5]

```
rem      IL GIRO DI DO

program suono05

main:

sound_init(PORTB,0)
while true
    sound_play(191,131)
    sound_play(152,165)
    sound_play(128,196)
    sound_play(96,261)
    sound_play(128,196)
    sound_play(152,165)
wend

end.
```

Come si nota, non abbiamo usato alcuna variabile. Dopo la definizione del PIN 0 di PORTB, al quale occorre collegare un piccolo altoparlante, all'interno di un ciclo infinito WHILE / WEND il PIC esegue sequenzialmente le istruzioni che generano le note. Semplice vero? Un'ultima nota: non è stato necessario configurare il registro TRISB in quanto la funzione SOUND_INIT se ne fa carico completamente.

FINALMENTE LA MUSICHETTA

Adesso che abbiamo tutti gli "ingredienti" e le conoscenze per produrre suoni e note, ci possiamo cimentare nell'arduo compito di programmare il microcontrollore per l'esecuzione di una famosissima melodia.

Tanti auguri a te

Il brano in questione da riprodurre è conosciuto universalmente e si tratta della canzoncina *"Tanti auguri a te"*, tradotta in tutte le lingue del mondo e utilizzata durante i compleanni.
Per poter scrivere sul PIC la procedura serve naturalmente conoscere non solo la sequenza delle note, ma anche la loro durata e soprattutto si devono programmare le eventuali pause di silenzio. Occorre pertanto una "minima" cultura di notazione musicale.

Lo spartito

In figura 6 possiamo leggere lo spartito del brano in questione; spartito che sarà "tradotto" in numeri e valori di frequenza e durata da far digerire al povero PIC.

Tabella delle note (tabelle 3)

Con un po' di pazienza andiamo a convertire le note del pentagramma in numeri, seguendo il metodo per il calcolo dei parametri sopra descritto.
N.B. Si assume che ogni **croma** sia della durata di 200 ms. (quindi un'intera battuta di 600 ms. essendo di 3/8 la divisione del tempo).

Spiegazione della tabella

Per la riproduzione del motivo musicale, nella tabella 2 abbiamo calcolato, di tutte le note, i due parametri fondamentali (nelle caselle in grassetto).
Essa è formata da 9 colonne che andiamo ad illustrare:

1. La prima colonna contiene semplicemente il testo e le parole "cadenzate e ritmate" della canzone (vedi spartito).

2. La seconda colonna contiene il nome della nota riprodotta.
3. La terza colonna contiene la frequenza di ogni nota prodotta (vedi tabella 2).
4. La quarta colonna contiene la durata di ogni nota, espressa in millisecondi (vedi spartito).
5. La quinta colonna contiene la lunghezza del periodo della nota, espressa in secondi e viene calcolata con la formula T = 1 / F.
6. La sesta colonna contiene la lunghezza del periodo, stavolta espresso in microsecondi e si ottiene moltiplicando il periodo calcolato per 1.000.000 (un milione).
7. La settima colonna contiene il valore della sesta colonna diviso 10, come richiesto dal compilatore, e costituisce il **primo parametro** da inserire nella funzione sound_play.
8. L'ottava colonna contiene il numeri di periodi che occorrono per ottenere una nota di un secondo (se notate bene corri-

Testo Canzone	Nota	Frequenza (Hz)	Durata (ms.)	Periodo (Sec.)	Periodo (Microsec.)	Periodo Us/10	Periodi per 1 secondo	Periodi Effettivi
Tan	Sol	784	400	0,001276	1275,51	**128**	784	**314**
ti au	Sol	784	200	0,001276	1275,51	**128**	784	**157**
gu	La	880	600	0,001136	1136,36	**114**	880	**528**
ri	Sol	784	600	0,001276	1275,51	**128**	784	**470**
a	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
te	Si	988	600	0,001012	1012,15	**101**	988	**593**
	Pausa		600					
Tan	Sol	784	400	0,001276	1275,51	**128**	784	**314**
ti au	Sol	784	200	0,001276	1275,51	**128**	784	**157**
gu	La	880	600	0,001136	1136,36	**114**	880	**528**
ri	Sol	784	600	0,001276	1275,51	**128**	784	**470**
a	Re2	1175	600	0,000851	851,06	**85**	1175	**705**
te	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
	Pausa		600					
Tan	Sol	784	400	0,001276	1275,51	**128**	784	**314**
ti au	Sol	784	200	0,001276	1275,51	**128**	784	**157**
gu	Sol2	1568	600	0,000638	637,76	**64**	1568	**941**
ri	Mi2	1318	600	0,000759	758,73	**76**	1318	**791**
ma	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
es	Si	988	600	0,001012	1012,15	**101**	988	**593**
tro	La	880	600	0,001136	1136,36	**114**	880	**528**
Tan	Fa2	1397	400	0,000716	715,82	**72**	1397	**559**
ti au	Fa2	1397	200	0,000716	715,82	**72**	1397	**279**
gu	Mi2	1318	600	0,000759	758,73	**76**	1318	**791**
ri	Do2	1046	600	0,000956	956,02	**96**	1046	**628**
a	Re2	1175	600	0,000851	851,06	**85**	1175	**705**
te	Do2	1046	600	0,000956	956,02	**96**	1046	**628**

Tabella 3 *Parametri fondamentali di tutte le note che compongono la canzone*

sponde alla sua frequenza ...).

9. La nona colonna infine contiene il numero dei periodi occorrenti per formare una nota di durata specificata nella quarta colonna, e la si ottiene con una semplice proporzione. Costituisce pertanto il **secondo parametro** calcolato della funzione musicale.

Andiamo a scrivere adesso il listato 6 nell'editor del compilatore e, senza commettere errori di battitura, carichiamolo sul PIC, dopo la sua compilazione.

[Listato 6]

```
rem     TANTI AUGURI A TE

program suono06

main:

sound_init(PORTB,0)   'Inizializza la RB0

rem      Il PIC Inizia a SUONARE
rem      ... e non la smette piu' .....
while true
    sound_play(128,314)
    sound_play(128,157)
    sound_play(114,528)
    sound_play(128,470)
    sound_play(96,628)
    sound_play(101,593)
    delay_ms(600) ' PAUSA DI SILENZIO
    sound_play(128,314)
    sound_play(128,157)
    sound_play(114,528)
    sound_play(128,470)
    sound_play(85,705)
    sound_play(96,628)
    delay_ms(600) ' PAUSA DI SILENZIO
    sound_play(128,314)
    sound_play(128,157)
    sound_play(64,941)
    sound_play(76,791)
    sound_play(96,628)
    sound_play(101,593)
    sound_play(114,528)
    sound_play(72,559)
    sound_play(72,279)
    sound_play(76,791)
    sound_play(96,628)
    sound_play(85,705)
    sound_play(96,628)
    delay_ms(600) ' PAUSA FINALE DI SILENZIO
wend

end.
```

È stata molto dura ma ce l'abbiamo fatta. Il nostro microcontrollore è adesso capace di riprodurre "fedelmente" la musichetta di augurio, senza sbagliare mai una sola nota e per un numero illimitato di volte.
Il listato, seppur lungo, non ha assolutamente bisogno di commenti.
Occorre solo impiegare il tempo per calcolare tutti i valori necessari ma, ripetiamo, con un software di foglio elettronico il compito sarà sicuramente agevolato.
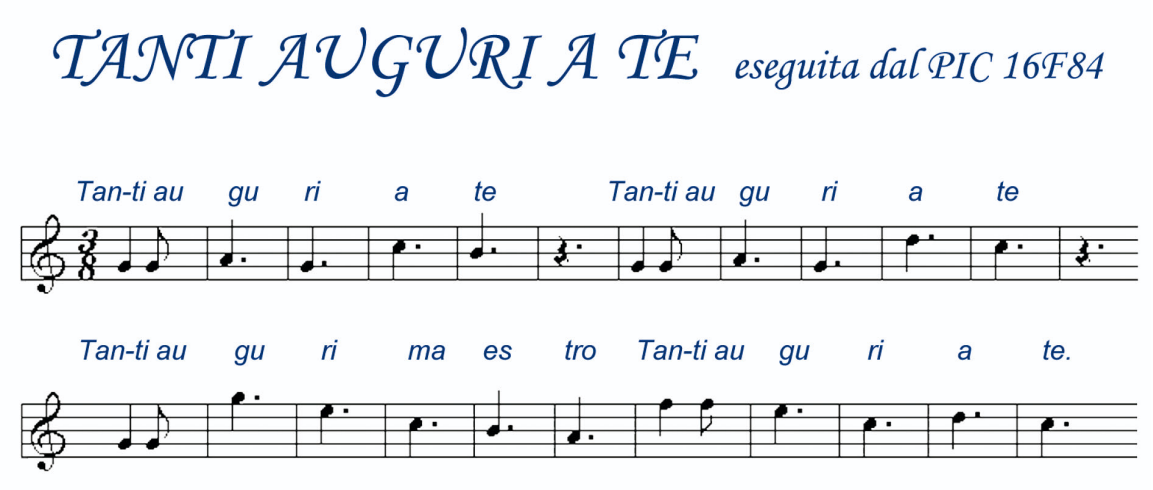


Figura 6 *Lo spartito di "Tanti auguri a te"*

PULSANTI SONORI CON BEEP

Come ultimo esempio di questo interessante articolo proponiamo una rielaborazione di un progetto presentato qualche numero fa.
Si tratta di realizzare un contatore, composto da una sola cifra (un display a 7 segmenti) che proceda avanti ed indietro di una unità, utilizzando semplicemente due pulsanti normalmente aperti. Realizziamo per lo scopo lo schema di cui alla figura 7.

Di cosa si tratta

In pratica si ha un display, composto da un visualizzatore a 7 segmenti, che può mostrare tutti i numeri da 0 a 9, semplicemente premendo uno dei due pulsanti (avanti e indietro) normalmente aperti.
Rispetto alla versione precedente però ci sono due differenze sostanziali ed elaborate:

1. Ogni qual volta che un pulsante è premuto, avviene la generazione di un "beep" di 1000 Hz.
2. Se si cerca di superare il limite raggiunto dalla cifra (ossia se ci si vuole spostare sotto lo 0 o sopra il 9) avviene la generazione di un tono lungo di 400 Hz (error tone).

Come si vede, dotando il prototipo di un'unità per la produzione di un suono, il lavoro dell'utente preposto all'utilizzo dei pulsanti è di gran lunga agevolato, in quanto ogni azione manuale (pressione del pulsante) è accompagnato da un suono che descrive l'operazione compiuta. Un esempio di quanto spiegato lo abbiamo nelle fotocopiatrici più sofisticate, dotate appunto di tastiera "sonora".

Il software

Trascriviamo il programma seguente nell'editor, compiliamo e scarichiamo il file prodotto con estensione HEX sul nostro micro **(listato 7)**.

Il programma contiene preziosismi e trucchi vari. Abbiamo definito l'array di costanti, *display*, che contiene i codici delle dieci cifre da visualizzare.
La variabile *k* invece ha il compito di tenere in memoria il conteggio.
La variabile *premuto* ha l'importante funzione di comunicare al PIC se uno dei tasti è appunto premuto o meno. Questo per evitare il fastidioso effetto del "rimbalzo". Costituisce pertanto, come segnalatore, un flag che accerta la pressione di uno dei pulsanti. Si passa poi all'azzeramento e alla definizione delle porte attraverso i soliti registri PORT e TRIS.
Adesso viene il bello. Tutto il programma itera attorno ad un ciclo infinito, limitato dagli statements *WHILE / WEND*. Tale ciclo non termina mai

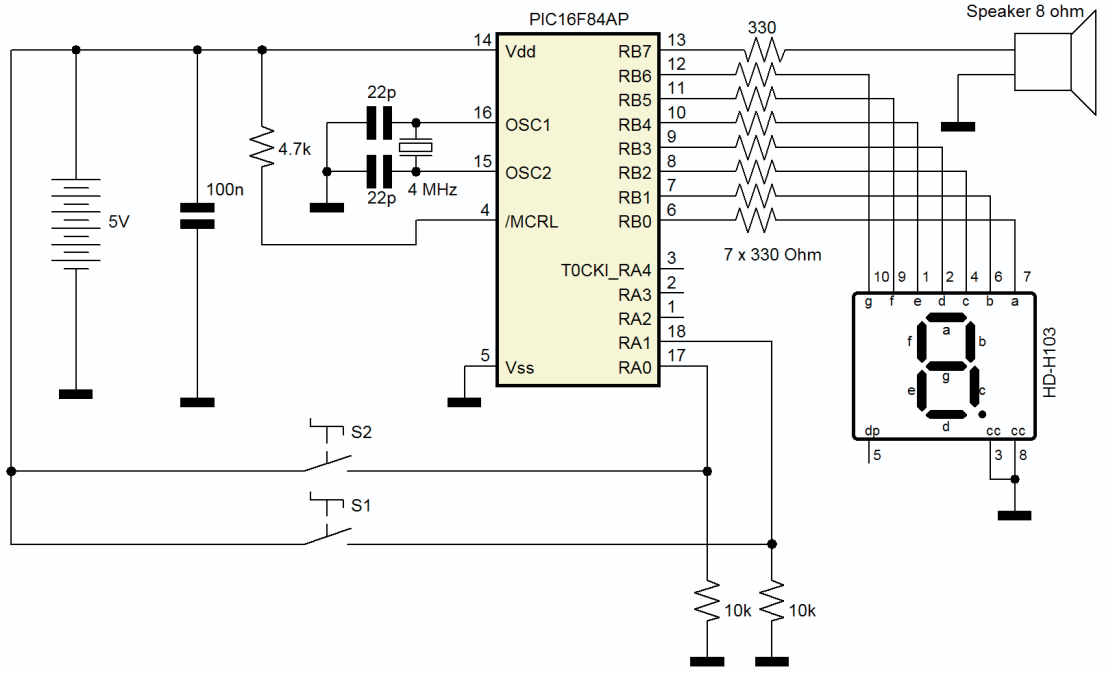


Figura 7 Schema di contatore up-down con due pulsanti n/a con beep

[Listato 7]

```
rem    Display Contatore con tasti BEEP

program suono07

const display as byte[10]= (63,6,91,79,102,109,125,7,127,111)
dim k as byte 'Variabile che contiene il NUMERO (0-9)
dim premuto as byte  'Flag che indica lo stato dei tasti

main:

   porta=0 'Azzera porta
   portb=0 'Azzera portb
   trisa=3 '00011 'RA0 e RA1 Input
   trisb=0 'definisce PORTB in output

   sound_init(PORTB,7) 'Inizializza altoparlante in RB7
   k=0         'Azzera Cifra
   premuto=0  'Presuppone che nessun tasto sia premuto
   while true
      '------Processa TASTO 1-------
      if (porta.0=1) and (k<9) and (premuto=0) then 'Se premo Tasto 1
         k=k+1 'Incrementa contatore
         premuto=1  'Imposta il flag di premuto
         sound_play(100,100) '1000 Hz 100 ms
      end if
      '------Processa TASTO 2-------
      if (porta.1=1) and (k>0) and (premuto=0) then 'Se premo Tasto 2
         k=k-1 'Decrementa contatore
         premuto=1  'Imposta il flag di premuto
         sound_play(100,100) '1000 Hz 100 ms
      end if
      '---Se non si preme niente il flag si azzera--
      if (porta.0=0) and (porta.1=0) then
          premuto=0
      end if
      '---Se si tenta di andare SOTTO il limite 0---
      if (porta.1=1) and (k=0) and (premuto=0) then
         sound_play(250,400) '400 Hz 1 sec.
      end if
         '---Se si tenta di andare SOPRA il limite 9---
      if (porta.0=1) and (k=9) and (premuto=0) then
         sound_play(250,400) '400 Hz 1 sec.
      end if
      '----Visualizzazione
      portb=display[k] 'visualizza numero
      delay_ms(100) 'Pausa per ANTIRIMBALZO
   wend

end.
```

in quanto è specificata una condizione vera (true) che non potrà mai cambiare.
Con la prima istruzione *IF* viene processato il tasto su RB0 che è adibito all'incremento del numero. Per aumentare di una unità la cifra, occorre il verificarsi di **tre** condizioni, ossia:

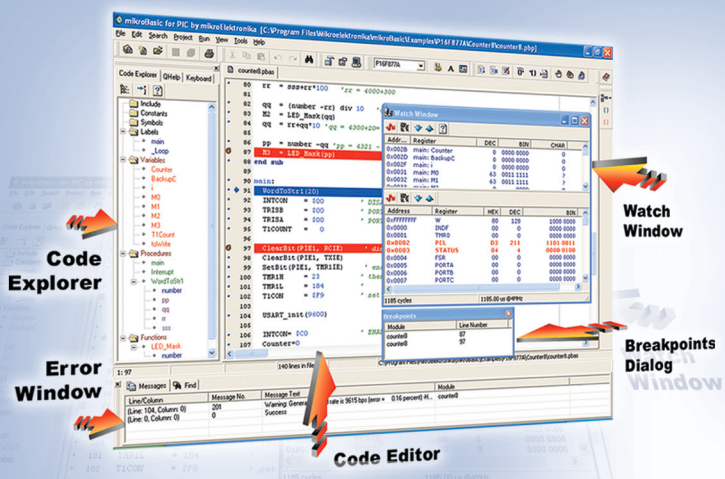
1. Che venga premuto il tasto (porta.0=1).
2. Che il contatore non si trovi al limite del suo conteggio (k<9).
3. Che il tasto stesso non sia già premuto (per evitare l'effetto repeat).

Solo così il programma può aumentare di una unità la variabile *k* (k=k+1). Inoltre, all'interno dello stesso blocco IF/THEN, è presente l'istruzione che genera il beep, ossia un tono di 1000 Hz della durata di 100 ms. grazie alla funzione *sound_play(100,100)*.
Il tasto di decremento segue la stessa filosofia di ragionamento, con le ovvie variazioni del caso.
Il valore della variabile *k* è così visualizzato sul display attraverso il comando *portb=display[k]* che assegna alla PORTB il valore di codifica del numero. Infine riveste tantissima importanza lo stato di attesa finale, attraverso il comando *Delay_ms(100)* che elimina completamente l'effetto rimbalzo, semplicemente rallentando l'esecuzione del programma a livelli più "umani".

CONCLUSIONI

Siamo arrivato alla fine anche di questa puntata. Sarebbero ancora tantissime le cose da dire ma lo spazio a disposizione è finito. Inventate da voi gli esercizi e i prototipi più disparati per verificare praticamente ciò che abbiamo imparato e ricordate che se un programmatore scrive un programma e questo funziona subito, è stato bravo, ma se scrive un programma che non funziona e poi scopre e corregge gli errori, allora è stato doppiamente bravo!
Alle prossime puntate dunque con tanti interessanti esperimenti e studi, per utilizzare al meglio il microcontrollore PIC. Vi aspettiamo.

Software Mikrobasic



Un potente compilatore Basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

Sesta parte
n° 244 - Ottobre 2005
Musika maestro! Generazione di melodie, note, effetti sonori e musica con il PIC

Settima parte
n° 245 - Novembre 2005
Pilotiamo i motori passo-passo

Ottava parte
n° 246 - Dicembre 2005
Ambiente di sviluppo

Mikrobasic per PICmicro

In questa puntata affrontiamo la programmazione e la gestione dei motori passo-passo utilizzando il Mikrobasic. Una scelta obbligata per intraprendere lo studio dell'automazione e della robotica in particolare.

L'utilizzo dei motori passo-passo, detti anche motori stepper, può sembrare difficoltoso. In effetti senza l'utilizzo di un microcontrollore diventa davvero difficile utilizzarli, soprattutto per quanto riguarda la scelta dei componenti elettronici da adottare. Grazie al PIC, certi compiti decisionali divengono subito facilmente risolvibili: basta infatti progettare un'adeguata scheda di pilotaggio ed il software farà tutto il resto.

BREVE INTRODUZIONE AI MOTORI PASSO-PASSO

A differenza dei normali motori a corrente continua o alternata, i motori passo-passo sono utilizzati nelle applicazioni dove è richiesta la massima precisione di movimento e di spostamento angolare. Esempi tipici, peraltro molto diffusi, sono le stampanti, i disk driver, i bracci meccanici, i foto-copiatori, gli scanner, eccetera.
Come dice il nome stesso, si tratta di motori che, con opportuni impulsi pilotati, ruotano il proprio asse di un determinato angolo. Il vantaggio principale consiste nella possibilità di programmare non solo la velocità di rotazione ma anche il numero di giri richiesti, il momento e la posizione esatta di partenza e di arrivo. Una programmazione quindi che consente la creazione di dispositivi con un'altissima precisione nei movimenti.
Se prendete in mano un motore passo-passo, e ruotate lentamente il suo albero, avvertirete subito un andamento a scatti (e non lineare e fluido come i normali motori). Ebbene, sono proprio questi i passi (step) che il motore rispetta nel suo avanzamento.
Nell'acquisto di un motore passo-passo, un primo dato caratteristico è appunto il numero di step necessario per completare un giro completo dell'asse; esistono infatti motori che compiono pochi step (quindi poco precisi) e motori che ne compiono moltissimi per completare una rivoluzione intera (quindi molto precisi). Nella tabella sottostante riportiamo i gradi di rotazione di un singolo passo (**S.A.**) ed il corrispondente numero di passi necessari ad effettuare un giro completo (**S.P.R.**).

S.P.R.	S.A.
10	36°
12	30°
20	18°
24	15°
40	9°
48	7,5°
50	7,2°
100	3,6°
200	1,8°
400	0,9°

Come si può notare, i due dati sono inversamente proporzionali: più elevato è il numero di step necessari a formare un giro completo e più "piccolo" è l'arco che deve compiere il motore. In altre parole, più l'angolo dello step è piccolo, più è preciso il motore, in quanto aumenta la "risoluzione" del passo.
Nella tabella successiva elenchiamo, in linea di massima, i vantaggi e gli svantaggi che derivano utilizzando simile motore, nelle applicazioni pratiche.

Pilotiamo i motori passo-passo



di Giovanni Di Maria
(g.dimaria@farelettronica.com)

A fronte di tanti piccoli svantaggi, è sicuramente consigliabile utilizzare un motore passo-passo per l'elevatissima precisione di rotazione, che i normali motori non potrebbero mai fornire, se non utilizzando sofisticati encoder, freni digitali ed ottici che, sicuramente, ne farebbero lievitare a dismisura il prezzo.

Schema base di un motore passo-passo

Un motore è formato internamente da tanti induttori che devono essere alimentati sequenzialmente per assicurare il corretto movimento dell'asse. Qualunque sia il numero di induttori interni, esso è sempre rappresentato, per comodità, da quattro bobine poiché, alla fine, sono altrettante il numero delle fasi.

Come si pilota un motore

Senza approfondire la teoria dei motori passo-passo per i nostri esperimenti ci serviremo di quelli *unipolari*, certamente più semplici da comandare. La tecnica principale consiste nel pilotare un avvolgimento per volta, sequenzialmente, in modo che il motore avanzi di uno *step* per volta. Come vedrete, appena avrete tra le mani un motore, il compito più arduo e difficile sarà proprio quello del riconoscimento dei fili di alimentazione, solitamente tutti di colore diverso.

Riferiamoci alla figura 2, che riporta la tipica struttura di un motore passo-passo. Per prima cosa occorre collegare il filo (o i fili) contrassegnati da Vcc al positivo dell'alimentazione. Per fare avanzare il motore di un passo per volta

Vantaggi
Riproducibilità di qualsiasi movimento
Possibilità di cambio di direzione senza modifica dell'hardware
Massima precisione di movimento
Coppia elevata
Possibilità di recupero da periferiche guaste
Possibilità di bloccare il motore in una posizione precisa
Velocità impostabile a piacere
Possibilità di impostare velocità bassissime
Minima usura di utilizzo
Svantaggi
Alto costo relativo
Difficoltà di reperimento in commercio
Difficoltà maggiore nel suo pilotaggio
Esigenza di una scheda supplementare per il pilotaggio ad alta potenza
Impossibilità ad ottenere velocità elevatissime
Rotazione non fluida ma con piccolissimi intervalli

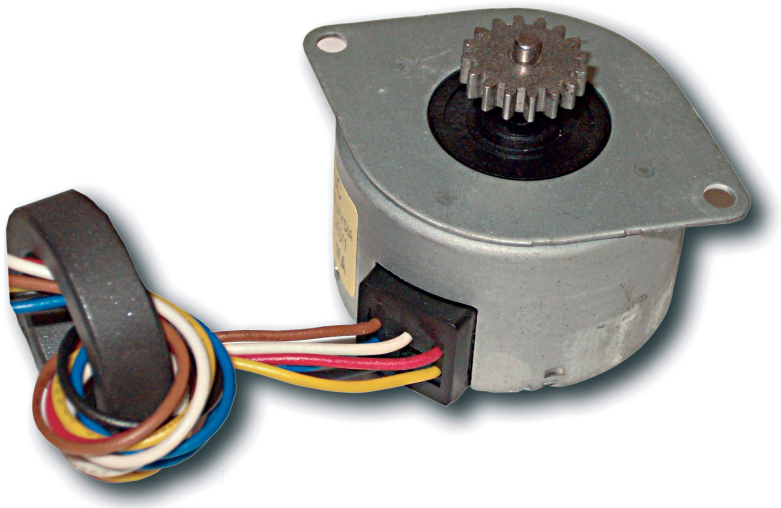


Figura 1 *Tipico motore passo-passo*

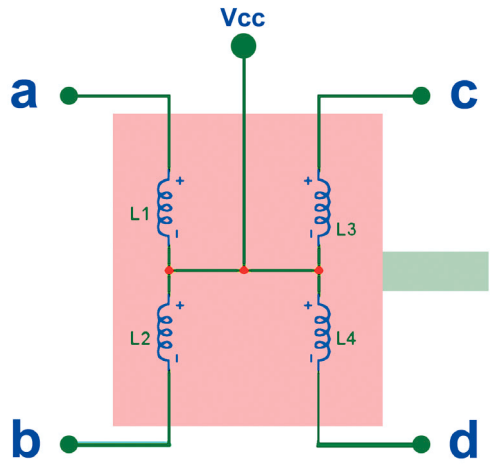


Figura 2 Struttura di principio

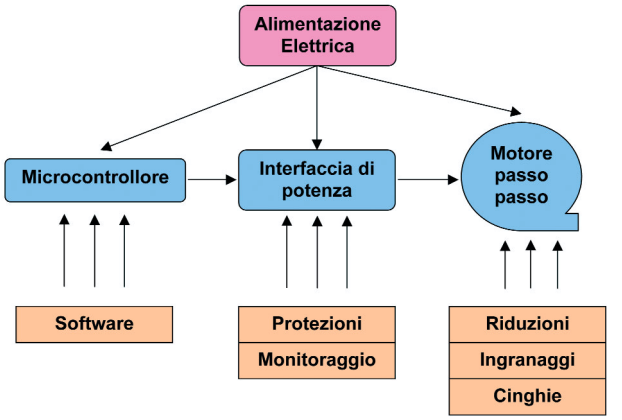


Figura 3 Schema a blocchi del sistema base

occorre, a questo punto, collegare alternativamente **a massa** i quattro terminali, contrassegnati con le lettere A-B-C-D. Solo se riuscite a trovare la "corretta" sequenza di pilotaggio si potrà far "girare" un motore stepper.

REALIZZIAMO LA SCHEDA DI PILOTAGGIO

Un motore passo-passo è un dispositivo *di potenza*, per cui non può essere direttamente pilotato dal microcontrollore.
Occorre realizzare un'interfaccia che trasformi i deboli impulsi del PIC in segnali a corrente elevata utilizzabili dal motore stesso.

Perché l'interfaccia

L'insieme di tutti i dispositivi che realizzeremo costituisce un valido esempio di sistema esperto adibito al controllo e pilotaggio di periferiche di potenza e getta sicuramente le basi a tutte quel-

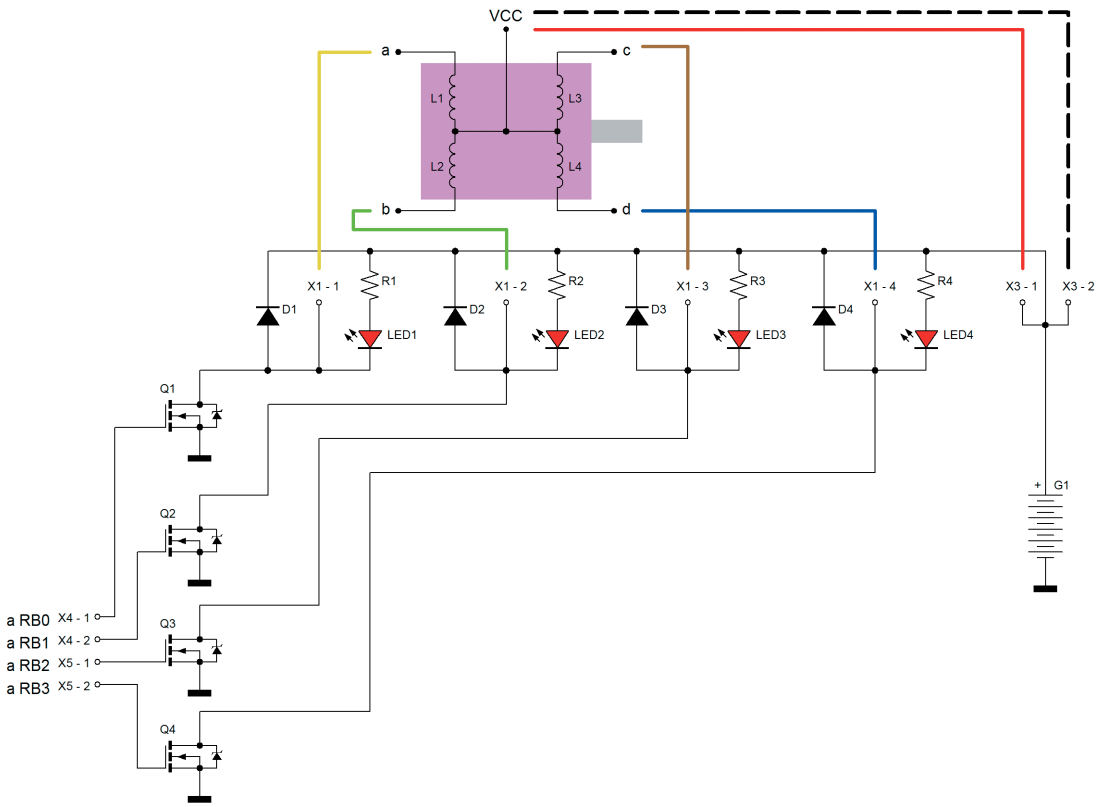


Figura 4 Schema elettrico dell'interfaccia

le attività in cui è la macchina, e non l'uomo, a svolgere i compiti più disparati, come avviene, ad esempio, nella robotica.
Il sistema base da concepire è formato da tre unità logiche fondamentali, come evidenziato nello schema a blocchi ci cui in figura 3:

- Il microcontrollore, che rappresenta il cervello e che dirige le azioni per il pilotaggio del motore.
- L'interfaccia di potenza, che trasforma i deboli impulsi del microcontrollore in impulsi più "robusti" per il pilotaggio del carico.
- Il motore vero e proprio.

Schema elettrico

Analizziamo adesso lo schema elettrico dell'interfaccia di potenza, illustrata in figura 4.

Il circuito viene alimentato dal generatore G1, una batteria o un alimentatore robusto di 5 o più Volt (sino ad un massimo di 30 Volt, dipendente dalla tensione del motore passo-passo).
Gli impulsi a bassa potenza, provenienti dal microcontrollore, sono applicati ai quattro ingressi rappresentati dai morsetti X4 (1 e 2) e X5 (1 e 2). Tali impulsi sequenziali vanno direttamente a pilotare i mos-power IRF 540, dei veri e propri di cavalli di battaglia (dati di targa: 100 Volt D-S, 0.077 ohm Rgs, 28 A Id). Tali Mosfet si comportano pertanto come veri e propri interruttori di potenza, senza inerzia e contatto meccanico, che abilitano o disabilitano il passaggio di corrente attraverso le quattro bobine del motore. Dal momento che ogni induttore, quando è attraversato da corrente pulsante, genera delle extra-tensioni, sono stati inseriti quattro diodi di protezione atti ad annullare i picchi pericolosi e rappresentati dai diodi D1, D2, D3 e D4.
Infine, gli impulsi ad onda quadra provenienti direttamente dal PIC sono visualizzati dai quattro diodi led, limitati in corrente dalle rispettive resistenze R1, R2, R3 e R4.
Come si nota dallo schema, il motore passo-passo si collega in questo modo: il filo comune si connette al positivo dell'alimentazione (tramite il morsetto X3) ed eventualmente (tratteggiato) anche il secondo filo comune (nel caso di motore a 6 fili) mentre i fili delle quattro bobine

CAD Professionale per Cattura Schemi e Progettazione Circuiti Stampati
CadSoft EAGLE
PCB Layout-Schematic-Autorouter
Low-Cost per Windows, Linux e MacOS
Distribuito da: PRECMA S.r.l. - Electronic Engineering
http://www.precma.it/ 039/5300590
Veloce, Stabile, Intuitivo, Potente, Espandibile
Manuali e supporto gratuito in italiano

si collegano ai quattro morsetti (X1-1, X1-2, X2-1, X2-2). I colori dei fili del motore dello schema sono puramente indicativi.
È importante sottolineare un aspetto fondamentale dei mos-power: il gate lavora esclusivamente in tensione (a differenza della base dei normali transistor BJT che lavorano in corrente) ed è ad altissima impedenza. Questo vuol dire che se toccate il terminale di ingresso con un dito, potreste attivare la chiusura del canale Drain Source.

Il circuito stampato ed il montaggio

Occorre adesso approntare il circuito stampato, utilizzando possibilmente la tecnica della fotoincisione, con la quale si ottengono migliori risultati. Realizzato il prototipo occorre eseguire la foratura con punta di trapano da 0,8 mm oppure da 1 mm. Si può dunque procedere alla saldatura dei componenti elettronici, iniziando naturalmente da quelli di dimensioni più ridotte, quali resistenze e diodi, per poi passare a quelli di profilo maggiore, come i mos-power, i diodi led ed i morsetti.
In figura 6 è riportata la traccia in dimensioni naturali del circuito stampato, mentre in figura 7 la disposizione dei componenti.

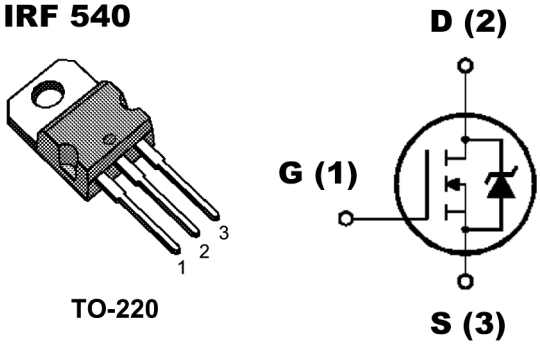


Figura 5 Pinout IRF 540

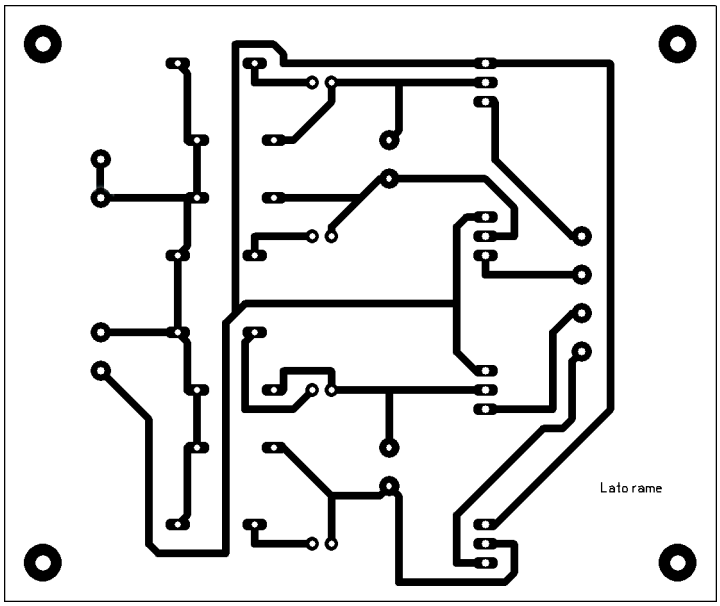


Figura 6 *Circuito stampato dell'interfaccia in scala 1:1 (lato rame)*

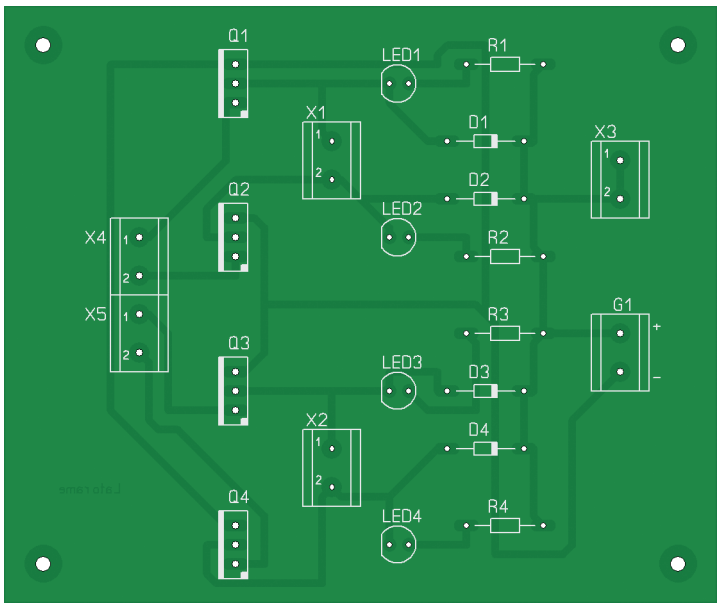


Figura 7 *Piano di montaggio dei componenti*

Elenco componenti	
Sigla	**Valore**
R1÷R4	470 Ω 1/2 W
D1÷D4	1N4007 o equivalente
LED1÷LED4	Diodo LED
Q1÷Q4	IRF540
X1÷X5	Morsettiera a 2 posti
G1	Morsettiera a 2 posti

Nel saldare i componenti, con un saldatore da 30 W, occorre fare attenzione a non surriscaldarli, inoltre, maneggiate con cura i Mosfet, in quanto una scarica elettrostatica sul terminale Gate potrebbe danneggiarli. Infine prestate cura al montaggio dei diodi, rispettando il catodo (anello bianco per gli 1N4007 e tacca piatta per i diodi led). In figura 8 si può osservare la realizzazione del circuito completo.

I componenti

I componenti utilizzati in questo progetto sono facilmente reperibili. Per quanto riguarda i diodi di protezione, si possono impiegare anche tipi equivalenti, lo stesso dicasi per i mospower. In tabella è presente la lista dettagliata con tutti i componenti elettronici utilizzati nel circuito.

Riconoscimento dei fili di uscita

Quando abbiamo tra le mani un motore passo-passo, la nostra prima reazione sarà probabilmente di ammirazione. Un oggetto di questo tipo è infatti il frutto di alta tecnologia alla portata di tutti. Ma subito dopo sopravviene una sorta di timore: quali saranno i corretti fili di pilotaggio?
I diversi colori causano sempre confusione e il motore è quasi sempre sprovvisto di qualsiasi indicazione o targhetta. Vi proponiamo pertanto un metodo semplice per la determinazione dei fili di uscita e della loro funzionalità. Tale procedimento prende il nome di metodo combinatorio.
Per i nostri esperimenti useremo solamente i motori unipolari, sicuramente più diffusi e soprattutto più semplici da pilotare. Da questi possono uscire 5, 6 oppure 8 fili, ma il procedimento è identico per tutti i modelli. Se i fili sono **5**, uno va collegato all'alimentazione positiva, gli altri quattro costituiscono le bobine. Se i fili sono **6**, due vanno collegati all'alimentazione positiva, gli altri quattro costituiscono le bobine.

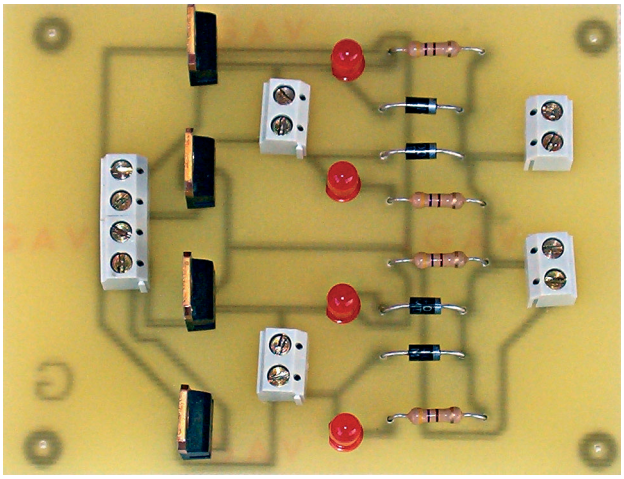


Figura 8 Il prototipo completo

Infine, se i fili sono **8**, quattro vanno collegati all'alimentazione positiva, gli altri quattro costituiscono le bobine.
Prendiamo quale esempio un ipotetico motore unipolare, provvisto di 6 fili colorati, ma senza alcuna indicazione di funzionalità, quindi completamente indecifrabile. La figura 9 riporta un tipico motore passo-passo a 6 fili.

In pratica occorre servirsi di un tester (in posizione ohm) misurando, a due a due, tutti i possibili accoppiamenti dei fili. Dal momento che il nostro motore è dotato di sei fili, dovremo effettuare 15 misurazioni diverse. In tabella 1 i risultati ottenuti.

Come potete notare dalle misure effettuate, ci troviamo davanti a bobine di fase della resistenza tipica di 10 ohm cadauna. Analizzando la tabella notiamo subito (vedi misure in grassetto) che solamente due forniscono una resistenza di 20 ohm (il doppio di 10 ohm) e che tale risultato deve esclusivamente addebitarsi al collegamento **in serie** di due bobine. Pertanto, guardando anche la struttura di figura 2, possiamo tranquillamente affermare quanto segue:

- I fili contrassegnati con colore Rosso, Giallo, Marrone e Arancio sono sicuramente quelli che

N.	Colori	Misura
01	Rosso-Nero	Aperto
02	Rosso-Giallo	**20 ohm**
03	Rosso-Bianco	10 ohm
04	Rosso-Marrone	Aperto
05	Rosso-Arancio	Aperto
06	Nero-Giallo	Aperto
07	Nero-Bianco	Aperto
08	Nero-Marrone	10 ohm
09	Nero-Arancio	10 ohm
10	Giallo-Bianco	10 ohm
11	Giallo-Marrone	Aperto
12	Giallo-Arancio	Aperto
13	Bianco-Marrone	Aperto
14	Bianco-Arancio	Aperto
15	Marrone-Arancio	**20 ohm**

Tabella 1 Risultati ottenuti

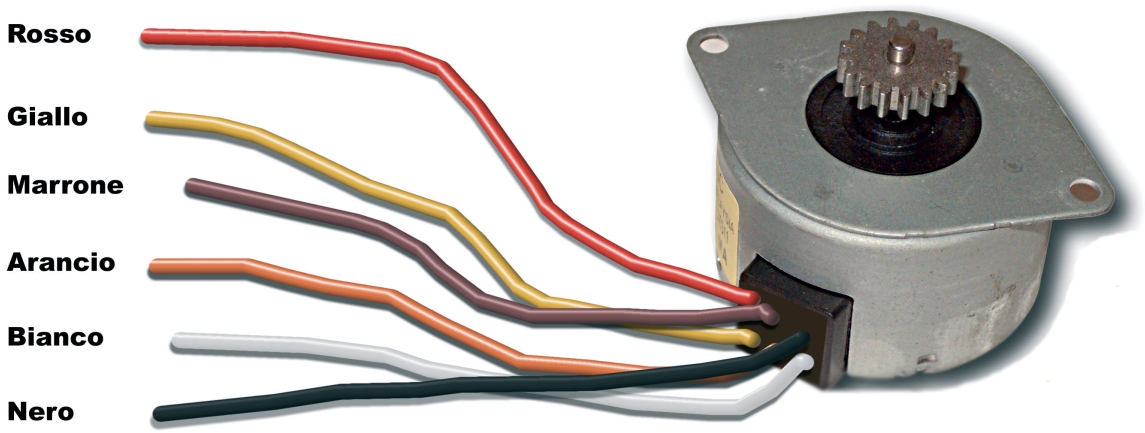


Figura 9 Un motore passo-passo "indecifrabile"

devono essere eccitati con un **impulso negativo**, per far procedere il motore di uno step.
- I fili contrassegnati con colore Bianco e Nero vanno collegati al positivo dell'alimentazione.
- Il filo Bianco è connesso elettricamente alle bobine con filo di colore Rosso e Giallo.
- Il filo Nero è connesso elettricamente alle bobine con filo di colore Marrone e Arancio.

Per quanto riguarda invece la corretta sequenza degli impulsi, questa deve essere provata a tentativi, ma dal momento che già conosciamo la funzione di tutti i fili, la maggior parte del lavoro di riconoscimento è concluso. In figura 10 riportiamo lo schema del motore con i fili colorati (e riconosciuti) seguendo il metodo proposto.
Ricordate che i colori dei fili cambiano da motore a motore, pertanto non si può generalizzare una regola, seppur approssimativa, per la determinazione delle funzioni di uscita.

Utilizzo avanzato della scheda

La nostra interfaccia può essere collegata anche ad un PC (oltre che ad un microcontrollore). I relativi ingressi pertanto si collegheranno ai relativi pin della porta parallela, il tutto coadiuvato da un opportuno software scritto in linguaggio Assembler, C o Basic.

Migliorie dell'interfaccia

In ogni caso la scheda di pilotaggio appena realizzata costituisce un prototipo con cui si può facilmente sperimentare. Si potrebbero apportare eventuali migliorie atte ad aumentarne la sicurezza e le prestazioni; in tal senso si potrebbero prevedere i seguenti accorgimenti:

- Aggiungere alcuni buffer in entrata per separare gli stadi ed adattarne meglio le impedenze.
- Optoisolare gli ingressi per prevenire danni accidentali al microcontrollore o al PC.
- Proteggere i mos-power da eventuali blocchi motore o cortocircuito degli induttori, che causerebbero certamente danni ai componenti nonché alle piste del circuito stampato (parlo per esperienza…).
- Prevedere un sistema di correzione di errore nel caso che il motore "perda" qualche step.

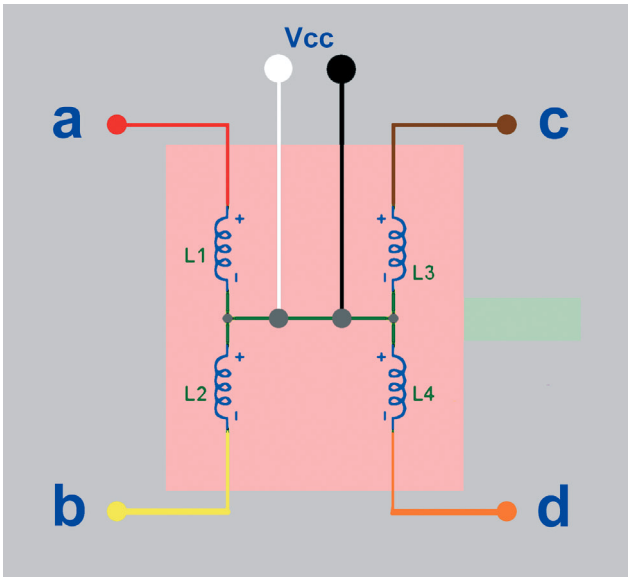


Figura 10 Schema motore riconosciuto e codificato

Finalmente mettiamo in moto

Dopo questa lunga ma necessaria premessa passiamo al collegamento dell'interfaccia e del motore al microcontrollore.

Il nostro motore

Per i nostri scopi è stato utilizzato un motore con le seguenti caratteristiche:

- Tensione nominale: 12 Volt.
- Sei fili colorati uscenti.
- Tipo unipolare.
- Colore dei fili collegabili all'alimentazione positiva: bianco, nero.
- Colore dei fili delle quattro fasi: rosso, giallo, marrone, arancio.
- Numero di passi o step: 48.
- Angolo di uno step: 7,5°.
- Numero massimo giri al secondo: 8 RPS.
- Numero massimo STEP al secondo: 384.
- Durata minima di uno step: 3 ms.

Schema di collegamento

Utilizzate lo schema riportato in figura 11. Si tratta di un assemblaggio di quattro parti fondamentali: il microcontrollore, l'interfaccia, il motore e l'alimentazione (diversa per due unità). Montate e collegate correttamente i quattro moduli per poter seguire gli esempi dell'articolo.
Ricordatevi soprattutto di collegare tra loro le

masse dei diversi moduli (scheda PIC, interfaccia ed alimentazione). Il PIC, come sempre, deve essere alimentato con una tensione di 5 Volt.

Un giro e si ferma

Bene, come primo esempio proponiamo un problema semplice da risolvere. L'esercizio è il seguente:

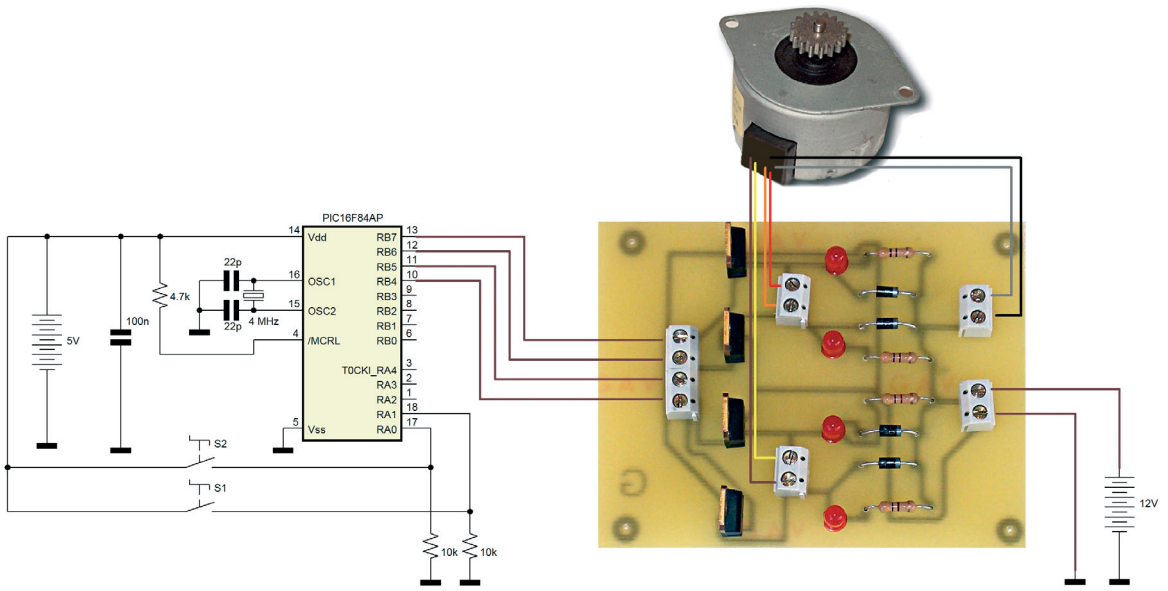
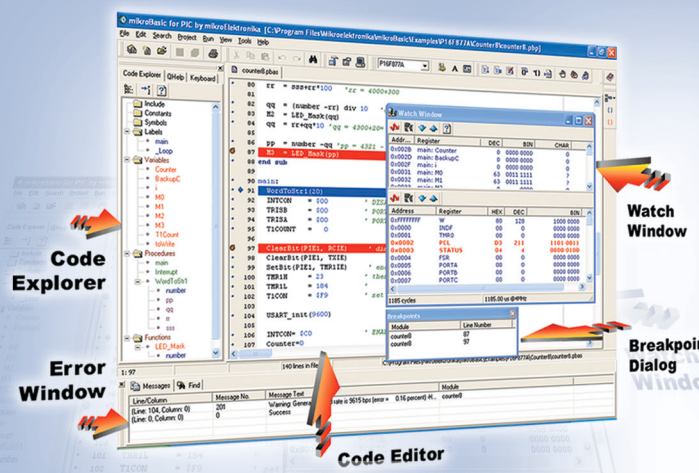


Figura 11 Schema applicativo completo

Software Mikrobasic



Un potente compilatore Basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

- All'accensione del circuito c'è una pausa di 2 secondi.
- Quindi il motore compie un giro completo attorno al proprio asse.
- Infine si ferma definitivamente.

Ricordiamo che il motore usato nell'esempio ha una "risoluzione" di 48 step. Esaminiamo il listato 1.

[Listato 1]

```
rem   Dopo una pausa di 2 secondi
rem   il motore compie un giro completo
rem   infine si ferma

program motore01

dim k as byte

portb=0   'Spegne porta B
trisb=0   'Definisce tutta PORTB in OUT

delay_ms(2000)    'Attesa 2 secondi

for k=1 to 12
  portb=1         'Step 1 motore
  delay_ms(100)
  portb=2         'Step 2 motore
  delay_ms(100)
  portb=4         'Step 3 motore
  delay_ms(100)
  portb=8         'Step 4 motore
  delay_ms(100)
next k
portb=0    'Spegne la PORTB

end
```

Il listato è molto semplice. Dopo aver definito la variabile k, che ha la funzione di eseguire tante volte il ciclo di pilotaggio, la PORTB viene azzerata ed inizializzata.
Si passa quindi alla generazione della pausa iniziale attraverso il comando delay_ms. Adesso viene il bello.
Il motore per potere girare ha bisogno di un impulso (a massa) per ogni filo di fase. Impulso che deve avere una durata specifica, nel nostro esempio, di 100 ms. Occorre pertanto "abilitare" (uno per volta) i primi quattro piedini del PIC, per poter "chiudere" i mos-power ed eccitare le bobine.
Ogni qualvolta che si imposta a 1 logico un bit della PORTB, il motore avanza di uno step avanti. Per poter abilitare in sequenza, ed uno per volta, i quattro pin, dobbiamo impostare la PORTB rispettivamente ai valori di 1, 2, 4, 8. Guardiamo la tavola della verità, riportata in tabella 2, per capire bene come avvengono gli impulsi di pilotaggio.

Cosa succede all'interno del ciclo FOR-NEXT? Le quattro istruzioni di assegnazione dei valori alla PORTB fanno semplicemente avanzare il motore di 4 step, pertanto se esso deve compiere un giro completo (48 steps del nostro stepper) occorre ripetere l'intera sequenza per 12 volte (48 : 4 = 12).
Infine, all'uscita del ciclo troviamo l'istruzione che azzera la PORTB, permettendo al motore di fermarsi e di interrompere la corrente alle bobine.

La retromarcia

Se volessimo far avanzare il motore in senso

# Step	Valore decimale PORTB	RB0	RB1	RB2	RB3
1	1	1	0	0	0
2	2	0	1	0	0
3	4	0	0	1	0
4	8	0	0	0	1

Tabella 2 *Tavola della verità per il pilotaggio a passo intero (marcia avanti)*

# Step	Valore decimale PORTB	RB0	RB1	RB2	RB3
1	8	0	0	0	1
2	4	0	0	1	0
3	2	0	1	0	0
4	1	1	0	0	0

Tabella 3 *Tavola della verità per il pilotaggio a passo intero (marcia indietro)*

opposto, per nostra fortuna non occorre invertirne la polarità, come avviene per i normali motori, bisogna invece "ribaltare" la sequenza degli impulsi da cadenzare al motore. Pertanto i valori decimali da fornire alla scheda saranno 8, 4, 2, 1 anziché 1, 2, 4, 8. In tabella 3 riportiamo la tavola della verità per la marcia indietro. Provate da voi lo stesso esercizio del listato 1, cambiando solamente il senso di marcia.

E adesso si corre

A differenza dei normali motori, per aumentare o diminuire la velocità di rotazione di un motore passo-passo è sufficiente variare il tempo tra un impulso e il successivo.
Naturalmente non si deve utilizzare un tempo di attesa molto breve, in quanto il motore farebbe "fatica" a seguirlo, con la conseguente perdita di qualche step o addirittura del mancato funzionamento. Consultate a tale proposito le caratteristiche intrinseche del motore.
Si vedrà nel prossimo esercizio come, variando semplicemente il tempo di attesa tra un passo e l'altro, il motore acquisterà una maggiore velocità. Strutturiamo pertanto il problema come segue:

- All'accensione del circuito c'è una pausa di attesa di 2 secondi.
- Il motore compie 5 giri completi a velocità bassa.
- Il motore compie 5 giri completi a velocità medio-bassa.
- Il motore compie 5 giri completi a velocità media.
- Il motore compie 5 giri completi a velocità medio-alta.
- Il motore compie 5 giri completi a velocità alta.
- Infine si ferma definitivamente.

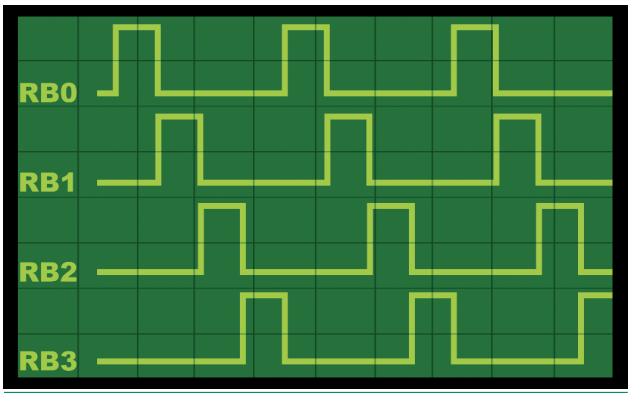


Figura 12 *Analisi uscite del microcontrollore (pilotaggio a passo intero, marcia avanti)*

Software di Simulazione Circuitale
Analogico/Digitale di tipo SPICE
BeigeBag B2Spice
The Best Price/Performance SPICE
Veloce, Intuitivo, Potente, Low-Cost
Distribuito da: PRECMA S.r.l.
http://www.precma.it/ 039/5300590
Compatibile con EAGLE e molti altri CAD

I termini velocità bassa, media e alta per il momento sono puramente indicativi.
Nel proseguo degli esercizi affronteremo come calcolarli.

[Listato 2]

```
rem   Dopo una pausa di 2 secondi
rem   il motore modifica la propria
rem   velocità di rotazione

program motore02

dim k as byte

portb=0  'Spegne porta B
trisb=0  'Definisce la PORTB in OUT

delay_ms(2000)    'Attesa 2 secondi
'---Velocità bassa---
for k=1 to 60       '5 giri
  portb=1         'Step 1
  delay_ms(100)
  portb=2         'Step 2
  delay_ms(100)
  portb=4         'Step 3
  delay_ms(100)
  portb=8         'Step 4
  delay_ms(100)
next k
'---Velocità medio-bassa---
for k=1 to 60       '5 giri
  portb=1         'Step 1
  delay_ms(80)
  portb=2         'Step 2
  delay_ms(80)
  portb=4         'Step 3
  delay_ms(80)
  portb=8         'Step 4
  delay_ms(80)
```

```
next k
'---Velocità media---
for k=1 to 60       '5 giri
  portb=1         'Step 1
  delay_ms(60)
  portb=2         'Step 2
  delay_ms(60)
  portb=4         'Step 3
  delay_ms(60)
  portb=8         'Step 4
  delay_ms(60)
next k
'---Velocità medio-alta---
for k=1 to 60       '5 giri
  portb=1         'Step 1
  delay_ms(40)
  portb=2         'Step 2
  delay_ms(40)
  portb=4         'Step 3
  delay_ms(40)
  portb=8         'Step 4
  delay_ms(40)
next k
'---Velocità alta---
for k=1 to 60       '5 giri
  portb=1         'Step 1
  delay_ms(20)
  portb=2         'Step 2
  delay_ms(20)
  portb=4         'Step 3
  delay_ms(20)
  portb=8         'Step 4
  delay_ms(20)
next k

portb=0    'Spegne la PORTB

end.
```

Eseguite il programma e noterete che ogni 5 giri il motore subisce un'accelerazione netta. Esaminate bene il listato ed eseguite tutte le variazioni e modifiche di sorta, per provare come si comporta il vostro stepper alle diverse velocità e sensi di marcia.
Una piccola annotazione: poiché, durante la rotazione, è alquanto difficoltoso contare il numero dei giri del motore, consigliamo di agganciare una molletta da bucato sull'albero, come se fosse una lancetta di orologio. Vedrete che il controllo della rotazione sarà sicuramente più agevole.

Il mezzo passo

Il mezzo passo (*half step*), è una tecnica che serve ad aumentare la risoluzione del motore. In altre parole, applicando tale metodo al nostro motore (da 48 step) ne otteniamo uno da 96 passi.
Raddoppia così la precisione dei movimenti a discapito naturalmente della velocità e della coppia motrice.
Vediamo in cosa consiste. In pratica occorre eccitare le varie fasi alternando l'erogazione di corrente ora ad una bobina, ora a due bobine. La tavola della verità di cui alla tabella 4 chiarirà ogni dubbio.

Nel listato 3 è riportato il programma per ottenere la rotazione del motore di un giro completo, sfruttando proprio il mezzo passo. Utilizzando tale metodo c'è un prezzo da pagare: la velocità massima del motore risulta dimezzata ed, inoltre, la coppia dello stesso varia in modo anomalo poiché si alternano due diverse fasi di eccitazione delle bobine in modo misto.

# Step	Valore decimale PORTB	RB0	RB1	RB2	RB3
1	1	1	0	0	0
2	3	1	1	0	0
3	2	0	1	0	0
4	6	0	1	1	0
5	4	0	0	1	0
6	12	0	0	1	1
7	8	0	0	0	1
8	9	1	0	0	1

Tabella 4 *Tavola della verità per il pilotaggio a mezzo passo (marcia avanti)*

[Listato 3]

```
rem    Il mezzo passo o HALF STEP
program motore03

dim k as byte

portb=0    'Spegne la PORTB
trisb=0    'Definisce la PORTB in OUT

delay_ms(2000)    'Attesa 2 secondi

for k=1 to 12    '1 giro
  portb=1
  delay_ms(100)
  portb=3
  delay_ms(100)
  portb=2
  delay_ms(100)
  portb=6
  delay_ms(100)
  portb=4
  delay_ms(100)
  portb=12
  delay_ms(100)
  portb=8
  delay_ms(100)
  portb=9
  delay_ms(100)
next k

portb=0    'Spegne la PORTB

end.
```

Esempio finale

Come esercizio finale proponiamo il seguente problema: un motore stepper gira inizialmente in senso orario. La velocità poco importa. Se l'utente preme il pulsante posto sul pin RA1 del PIC, il motore inverte la marcia, mentre se preme il pulsante posto su RA0 viene ripristinato il senso di rotazione iniziale. Riferiamoci sempre allo schema di cui in figura 11. Digitiamo il listato 4 e passiamo ai commenti di rito.

Scheda easyPIC3



La rivoluzionaria scheda di sviluppo per PICmicro

- Programmatore USB2.0 on-board
- Tastiera a 32 tasti
- 32 LED per il monitoraggio degli I/O
- 4 cifre LED a 7 segmenti
- Predisposizione per moduli LCD alfanumerici
- Predisposizione per moduli LCD grafici
- Predisposizione per comunicazione RS232
- Predisposizione per tastiera PS2
- Predisposizione per sensore di temperatura DS1820
- Supporto per tutte le famiglie PIC (anche PIC10F)
- Predisposizione per comunicazione USB
- Alimentazione esterna o via USB
- Fornita con 16F877
- Disponibile con o senza display

Ordinala subito su **www.farelettronica.com** oppure telefona allo 02.66504794

[Listato 4]

```
rem Inizialmente il motore va in senso orario.
rem Se si preme il tasto su RB1 cambia il senso
rem mentre se si preme il tasto su RB0
rem si ripristina il senso iniziale

program motore04
dim marcia as byte  '(0=avanti  1=indietro)

porta=0   'Azzera PORTA
porta=%00011  '(bit 0 e 1 in Input)

portb=0   'Spegne porta B
trisb=0   'Definisce la PORTB in OUT

marcia=0   'Inizialmente va avanti

while true
  '---Processa i tasti---
  if porta.1=1 then       'Se premo su RA1
    marcia=1
  end if
  if porta.0=1 then       'Se premo su RA0
    marcia=0
  end if
  '---Va in senso orario---
  if marcia=0 then
    portb=1         'Step 1
    delay_ms(50)
    portb=2         'Step 2
    delay_ms(50)
    portb=4         'Step 3
    delay_ms(50)
    portb=8         'Step 4
    delay_ms(50)
  end if
  '---Va in senso antiorario---
  if marcia=1 then
    portb=8         'Step 1
    delay_ms(50)
    portb=4         'Step 2
    delay_ms(50)
    portb=2         'Step 3
    delay_ms(50)
    portb=1         'Step 4
    delay_ms(50)
  end if
wend

end.
```

L'intero algoritmo è imperniato sulla variabile *marcia*, utilizzata come indicatore di stato (flag). Quando i tasti sono processati e valutati, è proprio questa variabile ad assumere permanentemente (e non solo durante la pressione dei tasti) il valore di 0 oppure 1.
Pertanto il flag viene poi analizzato dalle rispettive istruzioni di condizione *IF* e di conseguenza eseguiti i relativi statement di marcia avanti e marcia indietro.

RACCOMANDAZIONI FINALI

L'uso dei motori stepper non è per nulla pericoloso, ma costituisce un primo approccio all'elettronica di potenza, in quanto la corrente utilizzata nei circuiti comincia a crescere. Pertanto, durante la progettazione e l'utilizzo di questi tipi di motore consigliamo di prendere alcune precauzioni:

- Controllare sempre la tensione nominale massima sopportata dal motore.
- Verificare il numero di step del motore.
- Non alzare troppo la velocità: il motore perderebbe qualche "colpo" e si ridurrebbe la coppia.
- Tutti i motori generano un'elevata tensione di ricircolo, dovuta agli induttori; prevedere pertanto dei diodi veloci in antiparallelo.
- Se i mos-power dovessero riscaldare, applicare delle alette di alluminio adeguate per la dispersione di calore.
- Se, utilizzando un alimentatore, il motore dovesse comportarsi in maniera "strana" rispetto le vostre previsioni, inserire un grosso condensatore elettrolitico (almeno 10.000 microfarad) in parallelo all'alimentazione; evidentemente si sono verificati piccoli "ammanchi" di corrente dovuti allo *spunto* del motore. Il condensatore funge in questo caso da erogatore, durante i tempi morti.
- Agganciare una molletta da bucato all'albero rotante: potrete così contare facilmente i singoli passi e i giri del motore.

CONCLUSIONI

Come abbiamo visto, un altro importante tassello è stato inserito nello studio di questo fantastico compilatore che è il Mikrobasic. A presto dunque con tanti altri interessanti articoli di approfondimento. Buona sperimentazione a tutti.

Settima parte
n° 245 - Novembre 2005
Pilotiamo i motori passo-passo

Ottava parte
n° 246 - Dicembre 2005
Ambiente di sviluppo

Nona parte
n° 247 - Gennaio 2006
Come si memorizzano i dati e le informazioni

Mikrobasic per PICmicro

In questa puntata prendiamo confidenza con il potente ambiente IDE del compilatore Mikrobasic, per ottenere i massimi risultati nella programmazione assistita.

Dopo aver analizzato numerosi aspetti della programmazione del linguaggio Basic applicato ai microcontrollori (nella fattispecie al PIC16F84 della Microchip), con questo nuovo articolo studiamo le funzionalità che fanno parte dell'interfaccia utente del compilatore MikroBasic, meglio conosciute con il nome di **IDE**.

PERCHÉ ADESSO?

Abbiamo preparato questo articolo accontentando le numerose richieste giunte dai lettori. Perché non lo abbiamo pubblicato prima?
Innanzi tutto perché, con la lettura del presente, si ha possibilità di prendere un po' "di fiato" dopo tante nozioni di teoria e pratica, con la possibilità di studiare ed approfondire ancora meglio quanto già trattato nelle precedenti puntate.
In secondo luogo, se questo articolo fosse stato presentato all'inizio del corso, la maggior parte di lettori, in special modo quelli alle prime armi, non avrebbero potuto apprezzarne i contenuti perché ancora troppo "impreparati" per acquisire certe informazioni che invece avrebbero potuto costituire, con tutta probabilità, motivo di confusione o poca chiarezza. Pertanto la collocazione dell'articolo in questo punto del corso è giusta al fine di approfondire quello che, a detta di molti, è uno dei migliori compilatori Basic per PICmicro sul mercato.

IDE DEL MIKROBASIC

Il Mikrobasic costituisce un completo ed efficiente ambiente **IDE** (**I**ntegrated **D**evelopment **E**nvironment), infatti in un unico pacchetto software sono compresi: l'editor del sorgente, il compilatore Basic, il debugger e l'interfaccia di controllo del programmatore. Sono anche presenti numerose utility che aiutano il programmatore nella realizzazione e nella messa a punto di algoritmi critici e di programmi complicati.
Elenchiamo adesso solo alcune potenzialità del pacchetto software in questione:

- È un ambiente di sviluppo davvero completo.
- Dispone di un utilissimo convertitore in notazione HEX, BIN e DEC.
- Dispone di un Code Explorer avanzato.
- Presenta un ottimo assistente di codice.
- Permette la colorazione del codice sorgente contestualmente alla sua sintassi.
- Possiede un esauriente Help on-line.
- Genera automaticamente il file HEX.
- Visualizza il codice Assembler del programma.
- Mostra le statistiche (grafiche) sull'uso delle risorse.
- Presenta molte funzioni di utilità.
- ... e molto ancora.

Dopo questa introduzione possiamo asserire che il MikroBasic dispone di caratteristiche avanzate che ne giustificano l'acquisto, visto anche che il prezzo non è elevato ed, in definitiva, adatto a tutte le tasche.
Il presente articolo si riferisce alla *release 2.0.0* che si è dimostrata all'altezza della situazione e che sicuramente sarà soggetta a migliorie ed aggiornamenti continui, come del resto accade a tutti i compilatori prodotti dalla Mikroelektronika. Ricordiamo il MikroBasic ha due *fratelli*: il MikroC (compilatore C) e MikroPascal (compilatore Pascal) oggetto di un corso che sarà presentato nel corso del 2006.

Ambiente di sviluppo



di Giovanni Di Maria
(g.dimaria@farelettronica.com)

L'INSTALLAZIONE

Certamente semplicissima da eseguire, l'installazione del pacchetto non comporta alcun problema di sorta, e può essere eseguita anche su personal computer datati o, comunque, non eccezionalmente potenti.
Il primo passo da compiere è quello di scaricare il file di installazione dal sito di Fare Elettronica (Shop/Software/MikroBasic). Si tratta della versione demo, completamente funzionante ma con una sola limitazione: non compila programmi con dimensioni superiori a 2K (program words). Si esegua pertanto, tramite doppio click, l'installazione della procedura.
Dopo il messaggio di benvenuto, il sistema avvisa dell'inizio dell'installazione e consiglia di chiudere tutte le altre procedure eventualmente aperte. Chiaramente bisogna confermare cliccando il pulsante **Next**.
La seconda schermata invita l'utente alla lettura ed approvazione della licenza d'uso, per proseguire occorre accettare i termini della licenza, abilitando la prima opzione ("I accepts…"), e confermare la pagina cliccando nuovamente sul pulsante **Next**.
Si accede pertanto alla terza maschera, che chiede quali componenti del pacchetto si intendono installare. Il compilatore ed i file di help non possono essere naturalmente omessi dal processo, mentre si è liberi di copiare o meno i file di esempio. Consigliamo in ogni caso di installarli, in quanto una loro consultazione potrebbe essere d'aiuto durante la stesura dei propri programmi. Selezionate pertanto tutte le opzioni e cliccate sul pulsante **Next**.
Nella quarta maschera l'utente è invitato a scegliere la cartella nella quale copiare ed installare il programma. Vi consigliamo, a meno che non abbiate esigenze particolari, di accettare la cartella proposta cliccando il pulsante **Install**.
Inizia adesso la vera e propria installazione che consiste nel processo di scompattazione e di copia dei file di sistema e degli eseguibili nella cartella di destinazione. La procedura è monitorata attraverso una *progress bar* di colore blu, che avanza verso destra, mostrando la percentuale di lavoro compiuto. Finalmente l'installazione ha termine e viene mostrata l'ultima finestra di conferma. Occorre solamente cliccare sul pulsante **Finish**. Il programma risulta adesso installato sul nostro PC ed a conferma di ciò si può osservare la relativa icona sul desktop (uguale a quella mostrata a fianco).



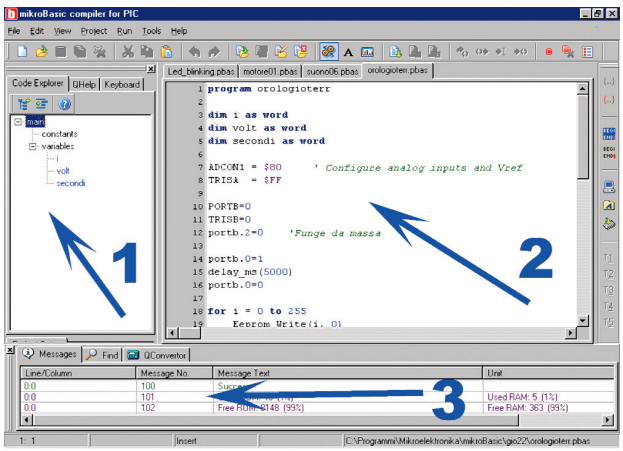


Figura 1 *L'ambiente di lavoro IDE*

L'ESECUZIONE

Con il doppio click sull'icona presente sul desktop, ha inizio la sessione di lavoro, preceduta dalla breve e momentanea comparsa di uno *splash screen*, che ci informa sulla versione del programma, subito dopo appare l'ambiente di lavoro, suddiviso in tante zone, ognuna delle quali svolge una precisa funzione (figura 1). Esaminiamo in dettaglio le principali:

1. La prima zona posizionata a sinistra dello schermo e denominata *Code Explorer* (esploratore), ha l'importante compito di mostrare, per il programma corrente caricato in Ram, molte informazioni relative al

codice Basic, come i nomi delle funzioni e delle procedure create ma soprattutto i nomi delle costanti e delle variabili. Si ha in questo modo la possibilità di tenere traccia di tutti i dati utilizzati nei nostri programmi. Il sottoscritto, usando altri compilatori, teneva traccia di queste informazioni semplicemente utilizzando carta e penna. Parlando sempre della prima zona, esiste la possibilità di selezionare altre due schede. La scheda **QHELP**, che mostra l'elenco alfabetico di tutte le funzioni *built-in* (funzioni di libreria e procedure), consultabile attraverso un semplice doppio click sulla voce che interessa. La scheda **Keyboard** invece fornisce un elenco completo di tutte le combinazioni dei tasti utilizzabili per accelerare una specifica operazione del compilatore (shortcuts).

2. La seconda zona, posizionata al centro dello schermo, contiene il codice Basic digitato dall'utente. Si tratta di un semplice ma potente editor, che mette a disposizione tanti utili strumenti di correzione e controllo.
3. Infine la terza zona, posizionata in basso, costituisce infine un'utile finestra dei messaggi (anche di errore) che il compilatore fornisce durante la compilazione e il linking del programma sorgente. Come si può notare, in questa sede esistono altre due schede supplementari: la scheda **Find** (trova) e la scheda **Qconvertor**, la quale ultima costituisce un potente convertitore tra i sistemi di numerazione decimale, esadecimale e binario.

IL CODE EDITOR

Come detto precedentemente, la zona numero 2 è rappresentata dal *Code Editor*, ovvero da un editor di testo dove l'utente scrive il proprio listato Basic. Vediamone le potenzialità.

Suggerimento dei parametri delle funzioni

Questa possibilità rappresenta una delle più interessanti potenzialità dell'ambiente IDE del Mikrobasic. Consiste in pratica in una sorta di suggerimento sull'utilizzo delle funzioni in Basic. Durante la digitazione di una qualsiasi funzione (numerica o non), all'apertura della parentesi tonda l'editor, mediante una bandierina gialla, suggerisce quanti e quali parametri dovranno essere immessi all'interno della funzione.

Code Assistant

Durante la digitazione di un comando, scrivendo i primi due o tre caratteri di questo e premendo contemporaneamente i tasti <CTRL> <SPAZIO>, la comparsa di una finestra agevola l'inserimento dello stesso. Per esempio, scrivendo le lettere "tr" e premendo i suddetti due tasti, comparirà una lista di possibili comandi, tutti inizianti con queste due lettere. Occorre infine posizionarsi su quello desiderato e premere Invio.

STATISTICHE

Il MikroBasic è uno dei pochi compilatori che posseggono una visualizzazione grafica delle risorse hardware utilizzate dal programma prodotto. Dopo aver compilato correttamente il codice sorgente, attraverso la scelta del menù **Project** e la voce **Build**, si può accedere al menù **View** e la voce **View Statistics**. Una finestra composta da sei schede mostrerà, molto dettagliatamente, le risorse utilizzate.
Rispettivamente le informazioni fornite (in veste grafica e testuale) riguardano:

- Utilizzo della Ram e della Rom (del PIC naturalmente) sotto forma di istogramma, di percentuale e di celle utilizzate.
- Disposizione e occupazione, sempre in veste di istogramma, delle procedure create e utilizzate nel codice sorgente.
- Occupazione "fisica" delle procedure nella memoria del PIC.
- Sommario delle funzioni utilizzate in formato gerarchico (tree).

Come si può notare si tratta di strumenti utilissimi che danno in tempo reale la completa "mappa" del lavoro prodotto (figura 2).

GLI STRUMENTI INTEGRATI

Il Mikrobasic dispone di alcune utility molto interessanti utilizzabili soprattutto per scopi di controllo e documentazione, vediamoli in dettaglio.

Usart Terminal

Si tratta di un terminale software di comunica-

zione che prevede il collegamento alla porta seriale (RS-232) del PC. Va eseguito cliccando sull'apposito menù **Tools** scegliendo la voce **Usart Terminal** (o semplicemente premendo i tasti CTRL-T). Sostituisce in pratica il programma *Hyper Terminal* di Windows, fornendo però un'interfaccia più gradevole e più intuitiva. Occorre ricordare di fornire correttamente i parametri operativi della comunicazione, quali: il numero della porta COM, la velocità di trasmissione, il numero di bit di stop, il controllo della parità e la lunghezza del pacchetto trasmesso. Può essere utilmente utilizzato per testare i modem seriali attraverso una serie di comandi **AT** (figura 3).

Ascii Chart

Selezionando dal menù **Tools** la voce **Ascii Chart** si accede all'elenco (molto importante) dei caratteri ASCII presenti nel sistema operativo ed utilizzati ormai dalla maggior parte dei compilatori e dispositivi elettronici (vedi ad esempio i display LCD intelligenti). La tabella (che visualizza tutti i 256 caratteri appartenenti al codice internazionale) contiene quattro colonne: la prima visualizza il nome fonetico del carattere (o la sua rappresentazione grafica), la seconda mostra il relativo carattere Ascii in notazione decimale, la terza lo stesso codice in notazione esadecimale e la quarta infine in notazione binaria. Molto utile quando si ha a che fare con conversioni e visualizzazioni di ogni genere (figura 4).

7 Segment Display Decoder

Si tratta di un utile tool che ha lo scopo di "comporre" la disposizione di un numero in un ipotetico display a Led a 7 segmenti, con la possibilità di ottenere la corretta rappresentazione e codificazione binaria e decimale, da dare "in pasto" al microcontrollore per la relativa visualizzazione. I codici possono essere utilizzati tanto per i display ad anodo comune quanto per quelli a catodo comune, con naturale differenziazione. Per accedere a questo tool occorre selezionare il menù Tools e la voce **Seven Segment Convertor**. La finestra che appare mostra un grosso display a 7 segmenti (inizialmente spenti). Occorre cliccare sul segmento desiderato per "illuminarlo" e per ottenere così la corretta informazione.
Si tenga presente che tale codifica è basata su un utilizzo di un bus di collegamento LED/PIC

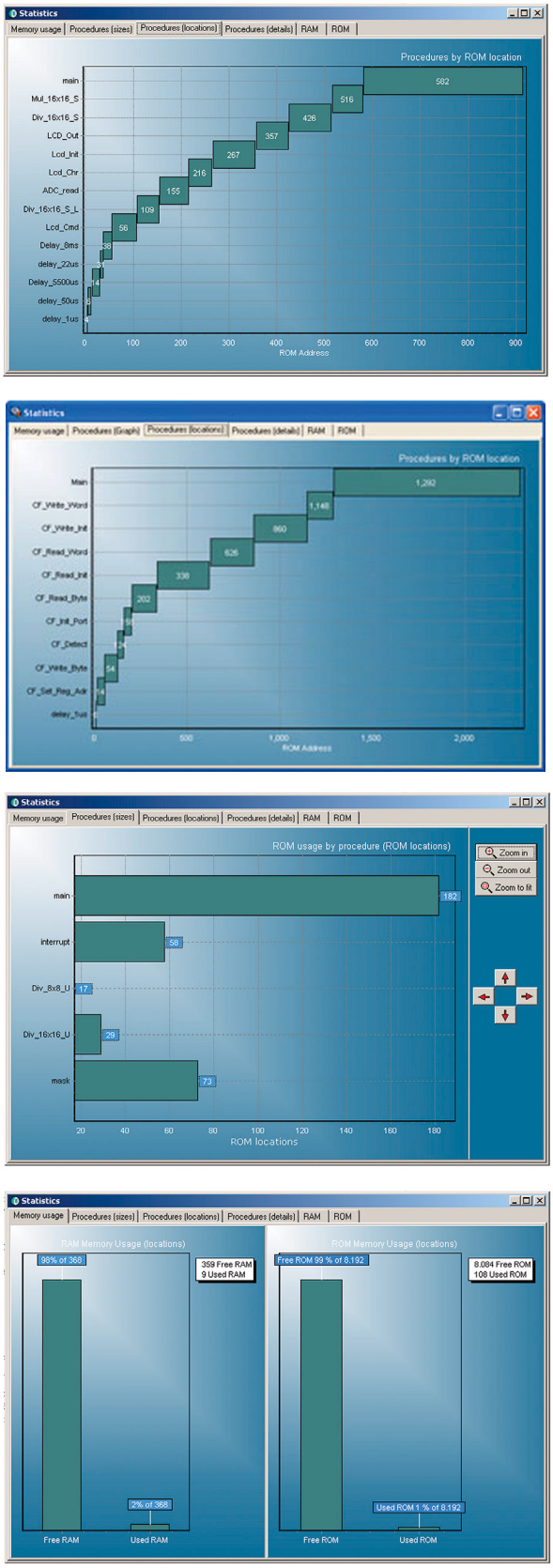


Figura 2 *Le finestre di statistica del compilatore*

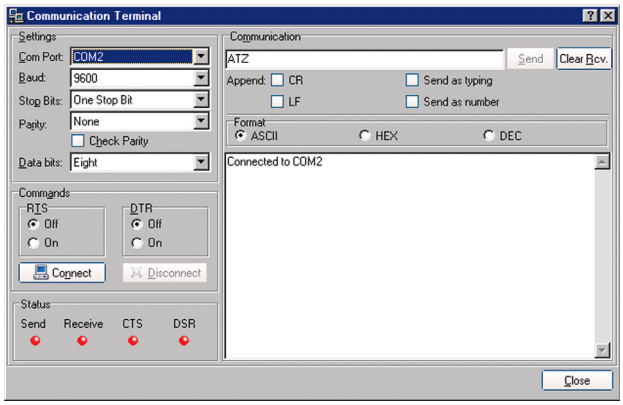


Figura 3 La finestra del terminale seriale

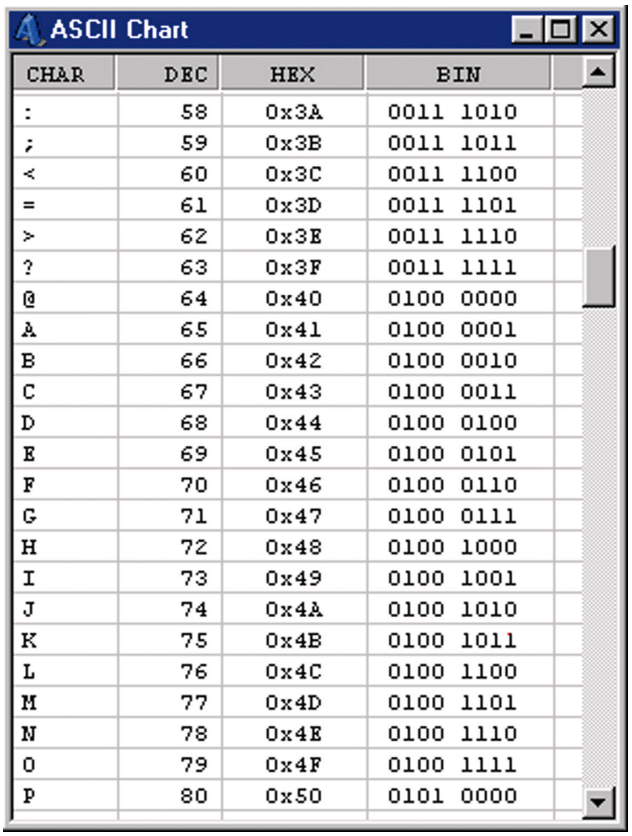


Figura 4 L'elenco dei caratteri Ascii

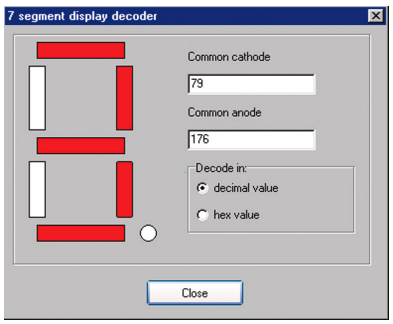


Figura 5 L'emulatore di un display a 7 segmenti con codifica

ad 8 bit, pertanto il display dovrà essere collegato ad una porta dotata di 8 pin (es. PORTB). Potrete trovare degli esempi pratici nel numero 240 di Fare Elettronica (figura 5).

MikroBootloader

I microcontrollori appartenenti alla famiglia dei PIC16F87X hanno la facoltà di modificare non solo i dati presenti nella RAM (come per il PIC16F84) ma anche quelli presenti nella FLASH. In altre parole, una routine può scrivere nella memoria programma, modificando quindi "se stessa". Questa possibilità prevede l'utilizzo di un piccolo *bootloader*, che riceve e scrive il nuovo firmware.

PicFlash Programmer

È sicuramente lo strumento più utilizzato della famiglia delle utility del compilatore. Si tratta di un programmatore software che svolge l'importante funzione del caricamento di un file .HEX, prodotto dalla compilazione, e del conseguente scaricamento nella memoria del PIC. Si accede ad esso attraverso il menù **Tools** e la voce **PicFlash Programmer** o, più semplicemente, premendo il tasto **F11** della tastiera (figura 6).
In linea generale il suo funzionamento è molto semplice. Occorre solamente caricare il file prodotto .HEX attraverso il pulsante **Load HEX**, quindi *riversarlo* nella memoria del microcontrollore premendo il pulsante **Write**. Inutile dire che bisogna collegare alla porta USB del proprio PC il relativo programmatore. In una prossima puntata tratteremo in dettaglio proprio questo aspetto.

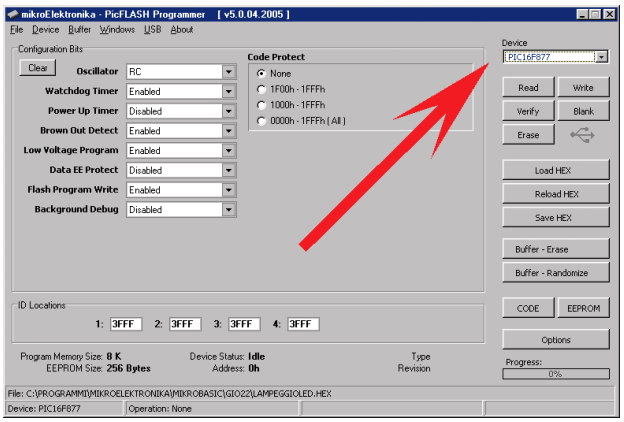


Figura 6 La finestra del programmatore PIC16F877 selezionato)

Hid Terminal
È un terminale che permette la comunicazione con I dispositivi HID (human interface device) come ad esempio mouse, tastiere e tavolette grafiche. Il mikrobasic dispone di una libreria per interfacciare al microcontrollore proprio tali periferiche (caratteristica davvero unica per un compilatore di questa classe).

Udp Terminal
È un terminale che permette la comunicazione direttamente con la rete LAN, inviando e ricevendo caratteri. Occorre fornire naturalmente l'indirizzo IP desiderato ed eventualmente il numero della porta di comunicazione. È possibile visualizzare altresì i caratteri inviati nella notazione Ascii, decimale ed esadecimale. Anche questa è una funzione davvero unica.

Mmc Terminal
Si tratta di un utile terminale che consente l'interfacciamento alle nuove memorie flash MMC (Multi Media Card). La comunicazione avviene attraverso la porta seriale del PC ed è possibile leggere o scrivere interi dati da queste memorie ormai di largo uso. Il collegamento al computer avviene attraverso un apposito adattatore. Questo tipo di memorie ormai è utilizzato in tutti i dispositivi di ultima generazione quali fotocamere, cellulari, lettori MP3, ecc, con spazio di massa fino a 1 GB. Inutile dire che questo apre le porte a tutta una serie di idee di progetto, altrimenti irrealizzabili vista proprio la complessità della gestione di tali memorie, MikroBasic lo rende davvero semplice ed intuitivo.

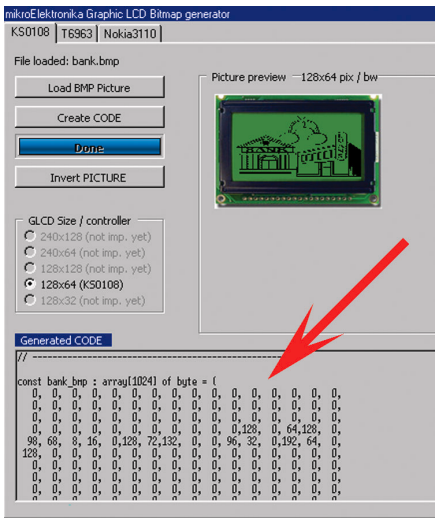


Figura 7 *GLCD Bitmap Editor*

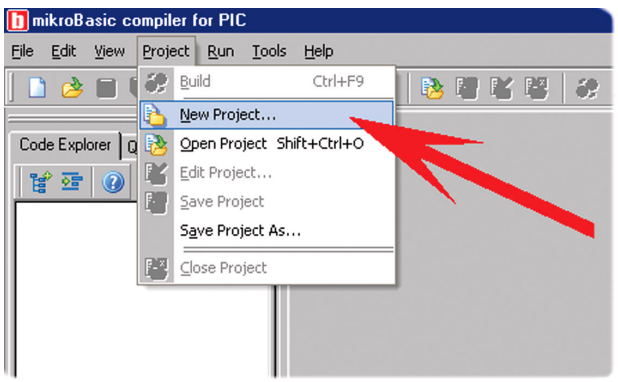


Figura 8 *Il menù di creazione Progetto*

Glcd Bitmap Editor
Sicuramente uno strumento utilissimo per chi programma ed utilizza I display grafici Lcd (Glcd). Il tool prevede (alla versione attuale) la gestione dei tre tipi più diffusi display:

- KS0108 con matrice bianco/nero di 128 x 64 pixel.
- T6963 con matrice bianco/nero di 128 x 128 pixel.
- Nokia 3110.

Per visualizzare un immagine o un testo su questi tipi di display, occorre convertire le informazioni in sequenze di numeri binari, corrispondenti ai pixel accesi o spenti, e dare infine tale codifica al microcontrollore. Tale preparazione potrebbe essere alquanto lunga e laboriosa, ma grazie a questo potente strumento, basta caricare un file grafico bitmap, dalle appropriate dimensioni, per generare automaticamente il listato basic già pronto contenente la codifica del disegno (figura 7).

CREIAMO UN PROGETTO
Dopo aver visionato le possibilità offerte da questo fantastico compilatore, iniziamo a studiare le diverse fasi della realizzazione di un progetto. Fasi che, ovviamente, molti di voi già conoscono, in special modo chi segue dall'inizio questo corso. La prima fase da seguire è naturalmente la creazione di un nuovo lavoro, appunto di un nuovo progetto. Occorre accedere al menù **Project** e scegliere la voce **New Project** (figura 8). Apparirà una finestra (figura 9) che invita l'utente a fornire dati e indicazioni generali sul tipo di progetto che si vuole creare:

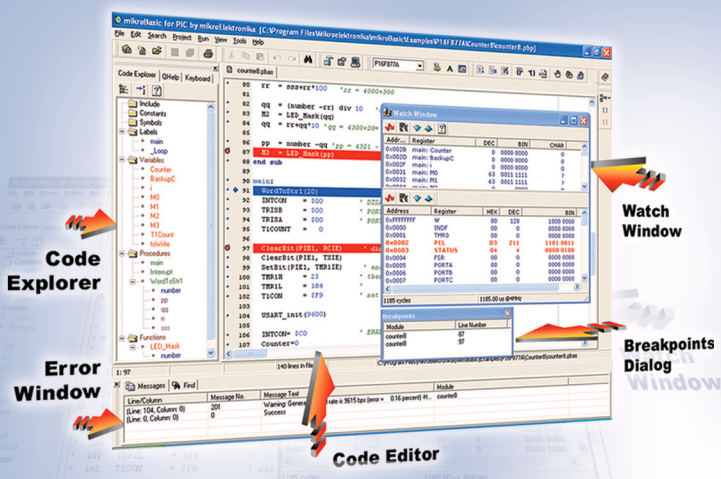
- **Project Name** è il nome che si vuol attribuire al programma e che in definitiva costituisce il nome dei file prodotti.
- **Project Path** è il percorso completo dove il sorgente e gli altri file prodotti saranno memorizzati. Si può selezionare il percorso cliccando sul pulsante **Browse**.
- **Description** è una breve descrizione (opzionale) sul programma che si sta creando, utilissima per tenere traccia delle diverse versioni e delle loro peculiarità.
- **Device** è il tipo di Microcontrollore utilizzato. Nel nostro caso si scelga il P16F84A.
- **Clock**: è la velocità del quarzo che utilizzeremo nel progetto (espresso in Mhz).

Per quanto riguarda i Flag, occorre "spuntare" quello relativo alla voce **WDT_OFF** e quello relativo alla voce **XT_OSC**.
Infine occorre confermare le proprie scelte cliccando il pulsante **OK**. Si è adesso pronti a scrivere il codice Basic del programma.

SCRIVIAMO IL PROGRAMMA

Nelle precedenti puntate si è ampiamente approfondito tale aspetto. Si tratta infatti di scrivere nella finestra dell'editor il programma con una corretta codifica del linguaggio. In figura 10 ne riportiamo un esempio. Si tratta di uno dei primi programmi incontrati nel corso. Una volta scritto il codice, si può lanciare la *compilazione* del sorgente.
Tale procedura effettuerà una serie di operazioni per la produzione del file HEX che sarà memorizzato all'interno del PIC. Per compilare il programma occorre cliccare sulla barra degli strumenti il pulsante che evidenzia due ingranaggi gialli, oppure sul menù **Project** e quindi sulla voce **Build**.
Il compilatore inizierà il lavoro della compilazione e, se tutto è a posto, produrrà il file HEX. Lo stato di tale processo è visualizzato da una *Progress-Bar* (figura 11).
Se il codice è stato scritto correttamente, il successo sarà testimoniato dalla finestra dei

Software Mikrobasic



Un potente compilatore Basic per PICmicro

✔ Code Editor
✔ Code Explorer
✔ Debugger
✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

messaggi, come mostra la figura 12. Invece, nel caso in cui il sorgente sia affetto da errori di qualsiasi genere (e ciò accadrà molto spesso credetemi…) la finestra dei messaggi sarà un tantino più "severa": nessun file sarà generato e compariranno dei messaggi riportanti il tipo di errore, magari logico o di sintassi. Tali messaggi saranno sempre visualizzati nella stessa finestra, come visibile in figura 13.

I FILE GENERATI

Se la compilazione ed il linking del sorgente hanno successo, il MikroBasic genera, nella cartella di lavoro, alcuni file con estensioni particolari, andiamo a descriverli:

- **.ASM** è un file Ascii contenente il sorgente codificato in Assembler. Notare che tale codifica è eseguita totalmente dal compilatore, quindi non dispone di alcuna ottimizzazione e, in generale, non è efficiente come quello creato dall'uomo.
- **.HEX** è il file più importante, in quanto rappresenta quello che deve essere "copiato" (attraverso il programmatore hardware) all'interno del microcontrollore PIC.
- **.LST** è un file di testo che contiene un report di riepilogo dell'allocazione della memoria, degli indirizzi, dei registri, delle routine e delle etichette utilizzate nel sorgente;
- **.MCL** è un file binario contenente l'applicazione creata, che può essere linkata ad altre applicazioni.
- **.PBAS** è il file che effettivamente contiene il sorgente scritto in linguaggio Basic.
- **.PBP** è un file di testo che raccoglie le informazioni circa i dati principali della definizione del progetto (vedi figura 9).

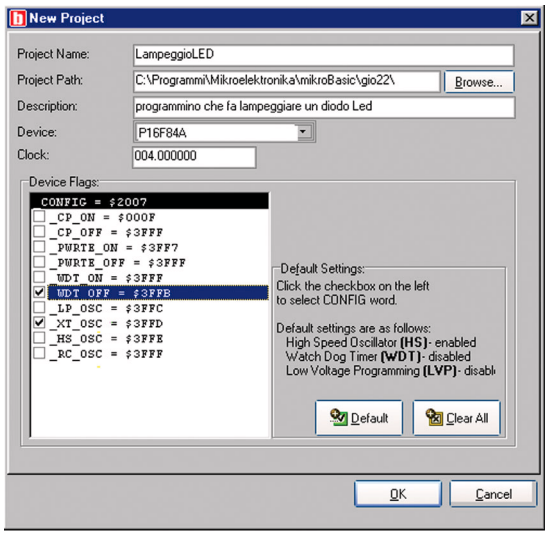


Figura 9 *Informazioni basilari del progetto*

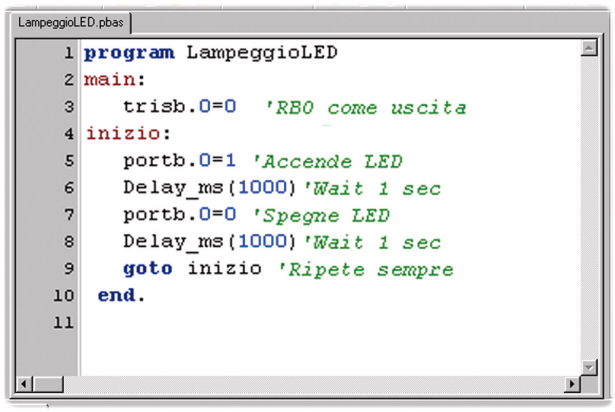


Figura 10 *L'Editor con un programma corretto*

IL DEBUG

Il debugger è uno dei tool più sofisticati e richiesti dai programmatori. È un mezzo estremamente potente con cui si simula, nel PC, l'esecuzione del programma creato, consentendone il controllo totale. Ecco in sintesi le possibilità offerte da un debugger:

- Iniziare e interrompere l'esecuzione del programma a piacimento.
- Inserire dei *break point* (punti di interruzione) nel codice.
- Comandare l'esecuzione del programma passo a passo.
- Ispezionare le variabili contestualmente all'esecuzione.
- Modificare il valore delle variabili senza bloccare la procedura.
- Variare il corso del programma senza alterarne la struttura.

Un debugger è come usare un microscopio che ci consente di esaminare a livello di "bit" il programma scritto, onde scoprire errori, malfunzionamenti e imprecisioni in generale.
La cosa più interessante è che il debugger del Mikrobasic avviene a livello di codice sorgente, nel senso che il controllo è eseguito sul programma Basic e non sull'eseguibile prodotto; questo per permettere una maggiore flessibilità nel controllo e nella verifica. È importante

sottolineare che il processo di debug può essere correttamente eseguito solo dopo che la compilazione è stata conclusa con successo.
Dopo aver scritto il proprio sorgente e averlo correttamente compilato, si può passare direttamente al debug del codice, non è una fase obbligatoria ma la si può utilmente utilizzare quando il programma non fornisce i risultati desiderati. Per eseguire il debugger occorre premere il tasto **F9** oppure accedere al menù **Run** e scegliere la voce **Start Debugger**, si aprirà la finestra *Watch* (figura 14). Come si può notare, in essa sono riportate le *variabili* usate nel programma (le variabili *utente* sono evidenziate dalla parentesi graffa), gli indirizzi della loro locazione e, soprattutto, il valore corrente, in notazione *esadecimale*.
Bisogna considerare un aspetto importantissimo: il debugger simula il flusso del programma e l'esecuzione della varie linee di istruzioni, ma **non** emula esattamente il comportamento del PIC. Vediamo in dettaglio i tasti da utilizzare per controllare le diverse opzioni:

- **F9** inizia la procedura di debug.
- **F6** alternativamente ferma e riavvia il debug.
- **F5** posiziona un break point alla posizione del cursore; se è già presente, questi è annullato.
- **F4** esegue tutte le istruzione dall'ultima espletata fino alla posizione del cursore.
- **F7** (Step Into) esegue passo per passo le istruzioni. Se è chiamata una funzione o procedura, esegue anch'essa passo per passo, mostrando la scansione dell'esecuzione. Utilissima per controllare il flusso in una UDF.
- **F8** (Step Over) esegue passo per passo le istruzioni. A differenza del precedente *Step Into*, l'eventuale esecuzione di una funzione o procedura avviene in maniera nascosta.
- **CTRL+F8** (Step Out) esegue completamente il programma sino alla fine.
- **CTRL+F2** (Stop Debugger) termina il processo di debug.

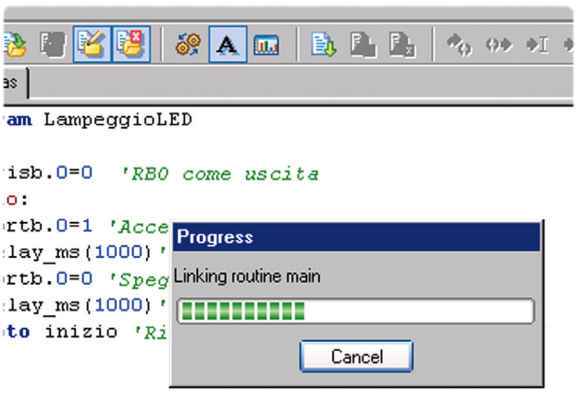


Figura 11 *Compilazione in corso*

Manipoliamo le variabili

L'esecuzione passo per passo permette di "sbirciare" all'interno di una variabile, con la possibilità di cambiarne il valore, vediamo come operare.
Durante l'uso dello *Step Into* oppure dello *Step Over*, i valori delle variabili sono aggiornati "in tempo reale" e visualizzati nella stessa finestra. Se occorre modificare il valore di una variabile (ad esempio per mutare il corso di un *ciclo* oppure per ribaltare il risultato di un *flag*) baste-

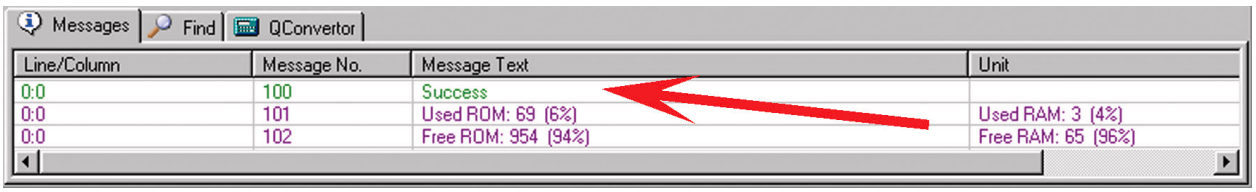


Figura 12 *Compilazione eseguita con successo*

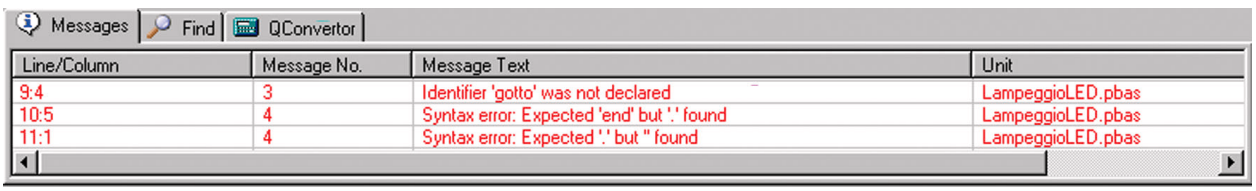


Figura 13 *Compilazione interrotta per la presenza di un errore di programmazione*

rà fare doppio click sul nome della stessa variabile all'interno della *Watch Window*. Pertanto una nuova finestra (Edit Value ossia Modifica Valore) inviterà l'operatore ad inserire un nuovo valore, che andrà subito ad aggiornare la variabile. Il parametro da modificare potrà essere immesso nella notazione binaria, decimale, esadecimale, float ed ascii (figura 15). Si ha altresì modo di scegliere la modalità di visualizzazione sulla *Watch Window* (binario, esadecimale, carattere, decimale, float).

Lo Stopwatch

Durante l'esecuzione "virtuale" del programma sviluppato, quindi nella fase del debug, è possibile *monitorare* un'altra famiglia di informazioni.
Nella Watch Window (attiva durante il debug), alla pressione della cartella *Stopwatch*, si accede ad un insieme di informazioni, relative soprattutto all'andamento *temporale* del flusso del programma scritto. Come si vede dalla figura 16, durante l'esecuzione del software in Basic in modalità *step by step* (con la pressione dei tasti F7 oppure F8) alcuni "contatori" sono continuamente aggiornati.
In pratica essa mostra il totale dei cicli (passi o step) eseguiti dal microcontrollore ed il relativo tempo trascorso (tempo virtuale – reale) dall'avvio del debugger, espresso in microsecondi (milionesimi di secondo). Il campo *delta* mostra invece il numero dei cicli (e relativi tempi) che intercorrono tra la *precedente* istruzione che il debugger ha eseguito e l'istruzione *attiva* (selezionata da una barra di colore blu) che deve essere ancora eseguita.

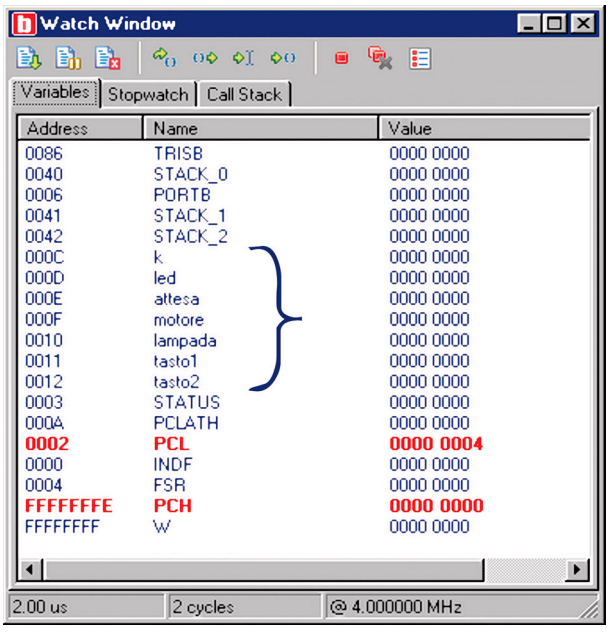


Figura 14 Finestra Watch

CONCLUSIONI

Bene, siamo arrivati anche questa volta al termine della puntata. L'esperienza è, come sempre, la migliore maestra, quindi datevi sotto con le sperimentazioni perché, come si sa, sbagliando si impara.
Dopo un lungo utilizzo, anche l'ambiente del compilatore del Mikrobasic, diventerà familiare e, pian piano, si troveranno anche trucchi e metodi per "ottenere" di più.
Vi rinnovo l'appuntamento alle prossime puntate con altre sperimentazioni divertenti, utili ed interessanti, al fine di ottenere sempre i massimi risultati dai nostri progetti in MikroBasic.

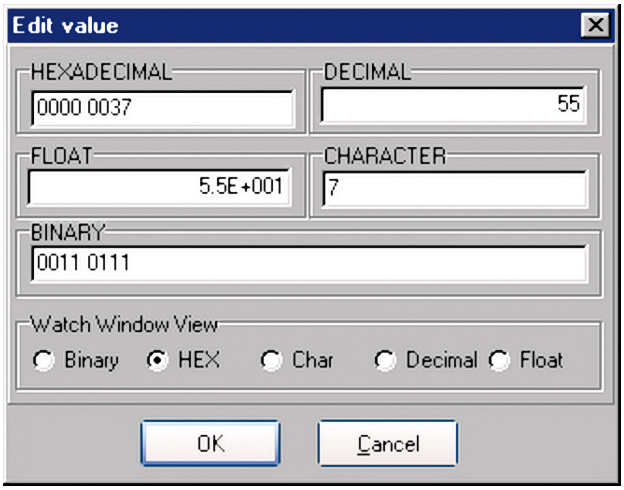


Figura 15 Finestra Edit Value

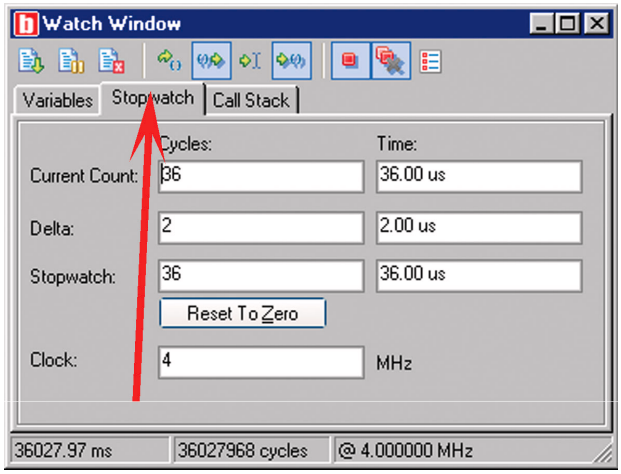


Figura 16 Le informazioni temporali dello Stopwatch

Ottava parte
n° 246 - Dicembre 2005
Ambiente di sviluppo

Nona parte
n° 247 - Gennaio 2006
Come si memorizzano i dati e le informazioni

Decima parte
n° 248 - Febbraio 2006
La scheda di sviluppo EasyPIC3

Mikrobasic per PICmicro

In questa puntata impariamo a memorizzare nella memoria del microcontrollore i dati elaborati, al fine di riutilizzarli in un secondo tempo anche dopo un'assenza di alimentazione elettrica.

Un programma memorizzato in un microcontrollore solitamente fa uso intensivo di variabili numeriche provvisorie. Questo vuol dire che se, per un qualsiasi motivo, si interrompe l'alimentazione elettrica del circuito, questo "perde" tutti i dati elaborati ed il processo inizia daccapo non appena si fornisce nuovamente la corrente. Vi sono dei casi in cui i dati devono restare permanentemente memorizzati, anche senza tensione, poiché il software potrebbe avere la necessità di riutilizzarli successivamente. In alcune applicazioni, il sistema deve aver la possibilità di "conservare" in maniera definitiva le informazioni, anche senza la presenza di tensione elettrica.

LA MEMORIA

Come dice la parola stessa, una memoria è un dispositivo capace di "ricordare" nel tempo una informazione. In essa i dati possono essere agevolmente inseriti, mantenuti e recuperati anche dopo molto tempo. In senso generale una memoria può essere di tipo *volatile*, se i dati in essa contenuti si cancellano in assenza di tensione, o di tipo *non volatile* se essa mantiene inalterate le informazioni anche senza corrente. Sono due le caratteristiche principali che contraddistinguono una memoria: il *tempo di accesso* (più è basso e più essa sarà veloce) e la *quantità di dati* memorizzabili.

Unità di misura

Come qualsiasi altra grandezza fisica, anche la memoria ha la sua unità di misura. Dal momento che essa è capace di ricordare "solamente" il *bit*, cioè uno stato logico alto (1 – acceso) oppure uno stato logico basso (0 – spento), si deve ricorrere alla rappresentazione dei suoi multipli, semplicemente per chiarezza e brevità. In generale la capacità delle memorie può essere così rappresentata, tenendo conto che 1 BIT (0-1) è l'unità fondamentale:

- **4 bit** formano **1 Nibble**
- **8 bit** formano **1 Byte**
- **16 bit** formano **1 Word**
- **1024 byte** formano **1 Kilobyte** (KB)
- **1024 KB** formano **1 Megabyte** (MB)
- **1024 MB** formano **1 Gigabyte** (GB)
- **1024 GB** formano **1 Tterabyte** (TB)

Si tratta, come si vede, di un'utile rappresentazione delle quantità di bit. I prefissi utilizzati sono

PREFISSO	NOME	VALORE
y	yocto	10^{-24}
z	zepto	10^{-21}
a	atto	10^{-18}
f	femto	10^{-15}
p	pico	10^{-12}
n	nano	10^{-9}
u (µ)	micro	10^{-6}
m	milli	10^{-3}
c	centi	10^{-2}
d	deci	10^{-1}
D	deca	10^{1}
h	etto	10^{2}
k	chilo	10^{3}
M	mega	10^{6}
G	giga	10^{9}
T	tera	10^{12}
P	peta	10^{15}
E	exa	10^{18}
Z	zetta	10^{21}
Y	yotta	10^{24}

Tabella 1 *Multipli e Sottomultipli delle grandezze*

Come si memorizzano i dati e le informazioni



di Giovanni Di Maria

comuni in tutte le unità di misure internazionali e sono rappresentate in **tabella 1**, che raccoglie una lista dei multipli e sottomultipli.

LA MEMORIA DEL PIC16F84

Il microcontrollore che stiamo studiando è composto da tanti tipi di memoria, ognuna delle quali con una funzione diversa. Con l'aiuto dello schema a blocchi del dispositivo (figura 1) andiamo ad esaminare le varie tipologie:

- **Memoria Programma** (flash program memory) contrassegnata in *verde*, che ha uno spazio di 1K di Words, corrispondenti a 1024 istruzioni.
- **Memoria Dati** (RAM) contrassegnata in *azzurro*, contenente 68 Bytes.
- **Memoria EEPROM** (Electrical Erasable Programmable Read Only Memory) contrassegnata in *rosso*, con una capacità di 64 Bytes. Nel corso dell'articolo faremo numerosi riferimenti ad essa.

La EEPROM

Si tratta di una speciale memoria, contenuta all'interno del PIC (ma non in tutti i modelli) che può conservare i dati anche in **assenza di alimentazione** elettrica. Costituisce in pratica una sorta di piccolo *Hard Disk* con una capienza di ben **64 bytes** (almeno per il modello 16F84). È un valore sicuramente irrisorio, ma per le applicazioni generali di elettronica è adatto a soddisfare la maggior parte delle esigenze.

La Eeprom è funzionante in lettura e scrittura durante il normale funzionamento del microcontrollore e, a *basso livello*, non è facilmente utilizzabile in quanto gli accessi ad essa devono essere eseguiti in modo indiretto. Per fortuna il Mikrobasic si occupa automaticamente di tutta la gestione della lettura/scrittura a livello Assembler, senza minimamente coinvolgere il lavoro del programmatore, che deve occuparsi di cose ben più importanti.

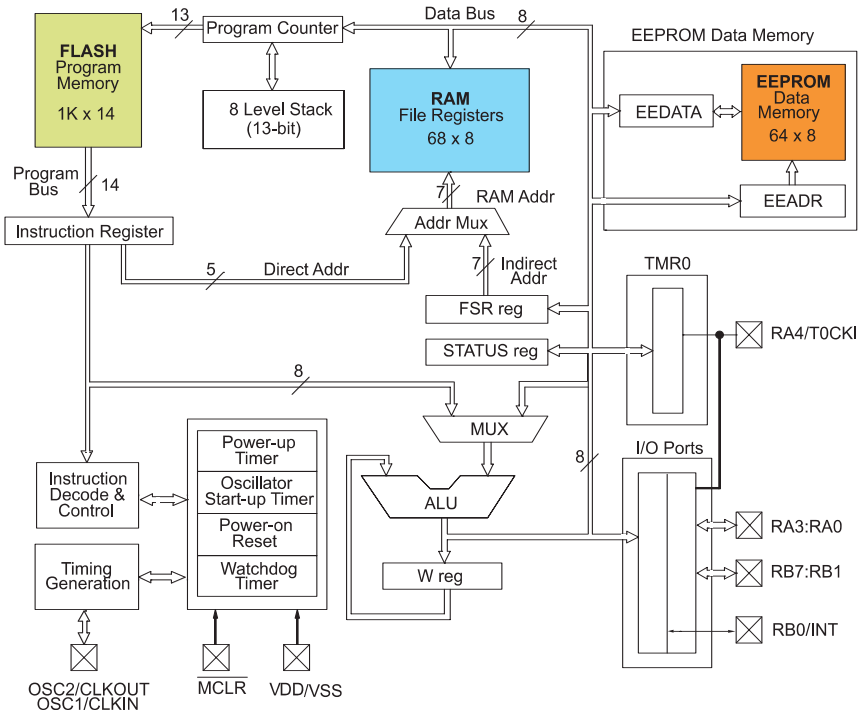


Figura 1 Schema a blocchi del PIC16F84

In lettura ogni byte è semplicemente letto con un'opportuna istruzione, mentre in scrittura il valore del dato è immesso all'interno di una specifica locazione andando a cancellare il contenuto precedente. Le operazioni di lettura e scrittura sono eseguite a gran velocità e il loro controllo è affidato al timer interno del micro.

Cosa possiamo memorizzare?

La Eeprom costituisce una sorta di *pila*, formata da 64 celle di memoria, ognuna delle quali più contenere un valore. Più precisamente, in ogni locazione può "entrare" solamente un byte, ossia un numero compreso tra 0 e 255 (ovvero 8 bit, esprimendoci in notazione binaria).
A differenza delle normali memorie statiche, nelle quali l'accesso ai dati può avvenire sequenzialmente, con metodi **FIFO** e **LIFO** (stack), nella Eeprom del PIC il processo avviene direttamente per una qualsiasi cella, senza dover necessariamente "leggere" tutte le precedenti.
Quindi se, ad esempio, si ha la necessità di accedere al byte numero 44, si può eseguire l'operazione di lettura o scrittura direttamente su tale cella, senza dover "disturbare" tutte le altre.

LE FUNZIONI BASIC

Il Mikrobasic mette a disposizione due semplici ma potenti istruzioni che permettono la gestione dei dati. Anche se in Basic tali funzioni si riducono ad **una sola** riga di programma, in Assembler esse si traducono in tantissimi comandi e op-code. Si può dunque capire come, attraverso un linguaggio ad alto livello, quale il Basic, si riesca a ridurre drasticamente le difficoltà ed il tempo dedicato alla programmazione, però a discapito della completa conoscenza dell'architettura interna del PIC.
Elenchiamo pertanto le uniche due funzioni dedicate al controllo della Eeprom:

- *Eeprom_Write*, che ha il compito di memorizzare un valore in una specifica locazione.
- *Eeprom_Read*, che ha il compito di leggere un valore da una specifica locazione.

La prima è una **procedura**, perché esegue semplicemente una memorizzazione, senza ritorno di risultati, mentre la seconda è una **funzione**, in quanto *restituisce* un valore letto dalla memoria.
In figura 2 si può comprendere il flusso dei dati **da** e **verso** la memoria, in relazione alle istruzioni Basic utilizzate nel programma.

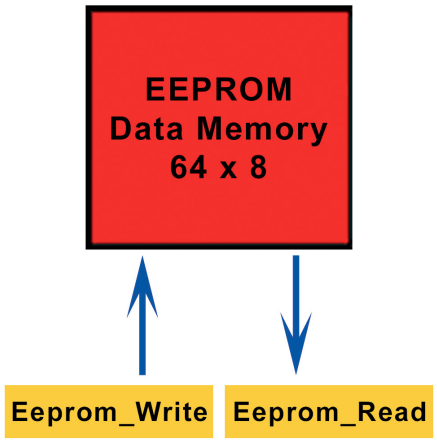


Figura 2 Flusso dei dati secondo le istruzioni utilizzate

Eeprom_Write

Approfondiamo adesso l'utilizzo di questa procedura. Essa si usa, come detto, per poter scrivere un byte in una qualsiasi locazione della Eeprom. La sua sintassi, ovvero la modalità di utilizzo, riportata dal manuale, è la seguente:

```
sub procedure Eeprom_Write(dim address as
byte, data as byte)
```

dove

- **sub procedure** indica che si tratta di una procedura.
- **eeprom_Write** è il nome della procedura vera e propria.
- **address** è il primo parametro della procedura, di tipo byte, che contiene l'indirizzo della cella nella quale si vuole memorizzare il dato.
- **data** è il secondo parametro della procedura, di tipo byte, che contiene il dato vero e proprio da memorizzare.

Da quanto detto si evincono due fatti importanti:

1. Dal momento che il primo parametro della procedura (address) è di tipo byte, si possono solamente indirizzare al massimo 256 celle di memoria, o locazioni. In ogni caso tale valore è più che accettabile poiché nel PIC16F84 esistono solamente 64 celle.
2. In ogni cella è possibile scrivere solamente un valore di tipo byte, in quanto il secondo parametro è proprio di questo tipo.

Con alcuni esempi vediamo adesso il funzionamento.

Esempio 1. Si vuol scrivere nella locazione numero 10 il valore 44 (ricordiamo che la memoria inizia dalla cella numero 0 e termina alla cella numero 63).
Si dovrà pertanto usare questa semplicissima riga di codice:

```
Eeprom_Write(10, 44)
```

Esempio 2. Si vuol scrivere nella locazione numero 27 il valore 123.

```
Eeprom_Write(27, 123)
```



Figura 3 *Obiettivo della procedura Eeprom_Write*



Figura 4 *Obiettivo della funzione Eeprom_Read*

Se si tenta di *immagazzinare* un valore superiore a 255, il compilatore non visualizza nessun tipo di messaggio di errore, ma sarà solo memorizzata la parte *meno significativa*, come scomposizione di una cifra in un sistema di numerazione a base 256.

Eeprom_Read

Questa funzione è la complementare della precedente, nel senso che esegue una opposta procedura, ossia quella di leggere un dato dalla memoria Eeprom. La sua sintassi di utilizzo è la seguente:

```
sub function Eeprom_Read(dim address as byte)
as byte
```

Dove:

- **sub function** indica che si tratta di una funzione.
- **Eeprom_Read** è il nome della funzione vera e propria.
- **address** è il parametro della funzione, di tipo byte, che contiene l'indirizzo della cella da cui si vuole leggere il dato.
- la funzione ritorna un valore di tipo byte (il valore letto).

Esempio 3. si vuole leggere il valore memorizzato nella cella numero 4 della Eeprom e memorizzarlo nella variabile *temperatura*. Si utilizzi pertanto la seguente codifica:

```
temperatura = Eeprom_Read(4)
```

Naturalmente occorre la dichiarazione preliminare della variabile in questione.

Scrittura e lettura

Queste due potenti funzioni, molto semplici da usare, si occupano pertanto della scrittura e della lettura dei dati sulla Eeprom del PIC. Inoltre, durante il loro funzionamento, sono assicurati i tempi minimi di attesa (20 ms.) necessari al microcontrollore per garantire le corrette operazioni di lettura/scrittura.
Per ultimo possiamo anche aggiungere che queste due procedure possono essere utilizzate proficuamente anche all'interno di un ciclo *for-next*, per automatizzare una serie di memorizzazioni e letture.
In figura 3 e 4 sono schematizzate le azioni della procedura *Eeprom_Write* e della funzione *Eeprom_Read*, paragonando la Eeprom ad una grande botte di legno e i dati al vino rosso.

Paragone con l'Assembler

Nella prima puntata si è definito il Basic come linguaggio ad *alto livello*; questo vuol dire che le varie funzioni ed istruzioni si avvicinano lessicalmente all'espressione umana.
Al contrario l'Assembler, definito linguaggio a basso livello, è *prossimo* al "ragionamento" dei processori.
Ebbene, solo a titolo di curiosità, vogliamo dare un'occhiata alla differenza tra la procedura **Eeprom_Write** del Basic e l'equivalente codifica in *Assembler* per compiere la medesima azione.
In **Basic** l'istruzione `Eeprom_Write(0, 238)` corrisponde alla seguente codifica in **Assembler**:

```
bcf     STATUS, RP0
movlw   0x00
movwf   EEADR
movlw   0xEE
movwf   EEDATA
bsf     STATUS, RP0
bcf     INTCON, GIE
```

```
bsf     EECON1, WREN
movlw   55h
movwf   EECON2
movlw   AAh
movwf   EECON2
bsf     EECON1, WR
bsf     INTCON, GIE
```

Come si nota, le differenze sono evidenti: una sola istruzione in Basic corrisponde a tante altre in Assembler, con conseguente risparmio di tempo per lo sviluppo dell'applicazione, questo vale anche per tutte le altre istruzioni, comandi e procedure.

LA EEPROM NEI VARI MODELLI

Esistono sul mercato centinaia di modelli di microcontrollori PIC, differenti per prestazioni, numero porte, velocità, prezzo e, naturalmente, capacità di memoria. Pertanto vi sono anche delle differenziazioni per quanto riguarda la memoria Eeprom e la sua ampiezza.
In tabella 2 riportiamo i modelli più noti con la corrispondente capacità di memoria.

Modello	Byte Eeprom
16F870	64
16F871	64
16F872	64
16F873	128
16F874	128
16F876	256
16F877	256
16F818	128
16F819	256
16F83	64
16F84	64

Tabella 2 *Capacità di memoria di alcuni modelli di PIC*

PRENDIAMO CONFIDENZA CON LA EEPROM

Ecco un esempio veramente semplice per prendere confidenza con le istruzioni che gestiscono la Eeprom. A tale scopo si approntilo schema proposto in figura 5: si tratta di un utilizzo "nudo e crudo" del PIC senza alcuna utilizzazione delle porte di I/O.

Abbonati oggi!



www.farelettronica.com/abbonamento

fare elettronica

CULTURA ELETTRONICA APPLICATA

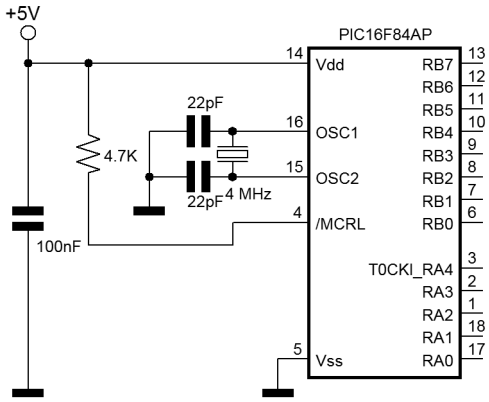


Figura 5 Primo schema applicativo

Il software

Nell'editor del Mikrobasic scriviamo il programma Basic proposto nel listato 1.

[Listato 1]

```
program Eeprom01
rem Questo programma memorizza un
rem insieme di caratteri nulla Eeprom
rem dopo la sua esecuzione

delay_ms(1000)
eeprom_write(0,76)
eeprom_write(1,101)
eeprom_write(2,103)
eeprom_write(3,103)
eeprom_write(4,101)
eeprom_write(5,116)
eeprom_write(6,101)
eeprom_write(7,32)
eeprom_write(8,70)
eeprom_write(9,97)
eeprom_write(10,114)
eeprom_write(11,101)
eeprom_write(12,32)
eeprom_write(13,69)
eeprom_write(14,108)
eeprom_write(15,101)
eeprom_write(16,116)
eeprom_write(17,116)
eeprom_write(18,114)
eeprom_write(19,111)
eeprom_write(20,110)
eeprom_write(21,105)
eeprom_write(22,99)
eeprom_write(23,97)
end.
```

Dopo la sua compilazione occorre, come al solito, scaricarlo sul PIC. Prima di dare alimentazione al circuito, si noti la situazione della mappa di memoria della Eeprom del PIC16F84 proposta in figura 6. Come si può vedere, la sezione dei dati di memoria è vuota ed è completamente inizializzata al valore esadecimale "FF".
Dopo aver dato tensione per almeno un secondo al PIC, apparentemente non accade nulla, perché le porte non sono state utilizzate, ma andando a "leggere" il contenuto del microcontrollore, si nota subito l'azione svolta dal programma: la zona di memoria Eeprom è stata modificata, come mostrato in figura 7.

Da notare

Le modifiche apportate alla memoria Eeprom sono permanenti, naturalmente sino a nuova programmazione. A questo punto è importantissimo

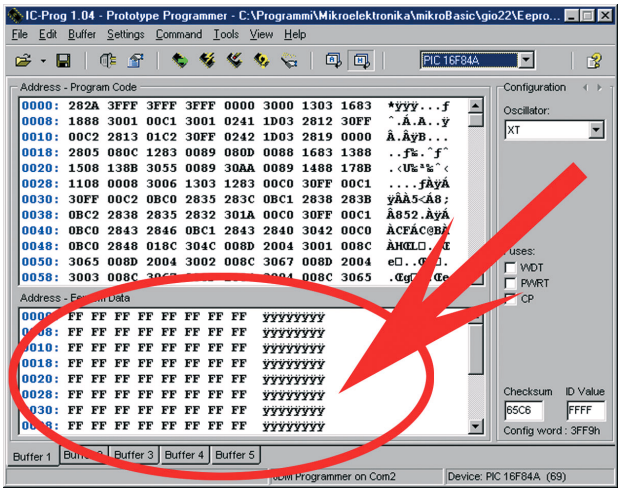


Figura 6 La mappa della Eeprom prima dell'esecuzione

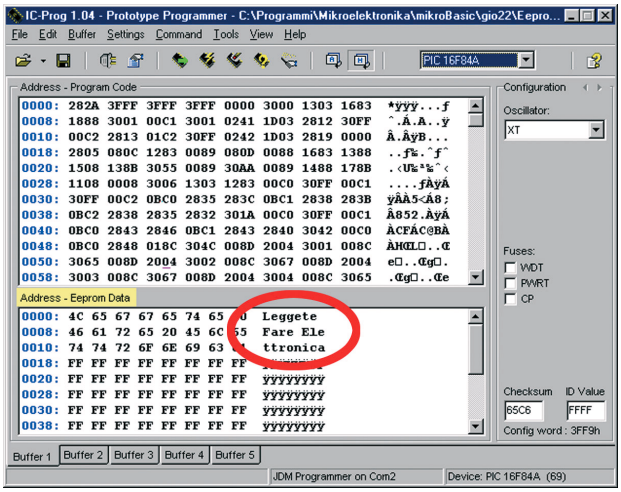


Figura 7 La mappa della Eeprom dopo l'esecuzione e lettura

sottolineare un aspetto fondamentale: per modificare e scrivere nell'**area programmi** del PIC (con capienza massima di H03FF byte, equivalente a 1024 locazioni di memoria, almeno per il modello 16F84) è necessario utilizzare un programmatore hardware, mentre per poter accedere e scrivere nella **memoria Eeprom** è sufficiente lo stesso programma basic.

Commentiamo il listato

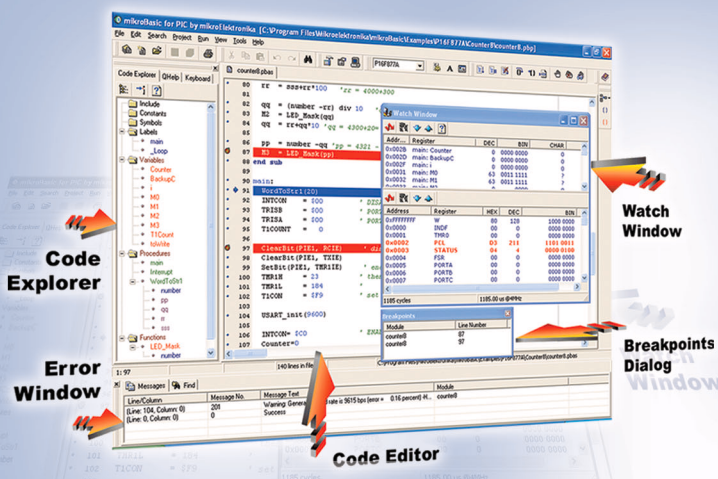
Il listato 1 è veramente semplice da scrivere e da comprendere. Dopo aver introdotto una pausa iniziale di un secondo, grazie all'istruzione *delay_ms(1000)*, avviene immediatamente il processo di memorizzazione dei caratteri, grazie ai 24 *statements* che eseguono la scrittura. I parametri della procedura **eeprom_write** sono due:

1. Il primo numero indica la locazione di memoria nella quale memorizzare il dato e, come si può notare, tale scrittura avviene "riempiendo" di caratteri le celle numerate dalla 0 alla 23, poiché la frase da salvare è formata da 24 caratteri.
2. Il secondo parametro indica il dato vero e proprio da memorizzare: in questo caso si è utilizzato direttamente il codice *Ascii* del carattere da scrivere, reperito attraverso il prezioso tool **Ascii Char** presente sul menù **Tools** del compilatore.

PRIMO PROGETTO: CONTA TRANSITO AUTOMOBILI

Bene, siamo pronti per iniziare la sperimentazione pratica sull'argomento. Cominciamo ad esaminare un reale esempio di applicazione che risulta utile nel monitoraggio stradale dei veicoli.
Si tratta di una centralina da piazzare lungo il ciglio di una strada, possibilmente a senso unico. La luce di un raggio laser, posto ad un lato della via, deve colpire permanentemente la superficie un sensore posizionato dall'altro lato. Naturalmente la presenza di luce testimonia il *non passaggio* di vetture; nel caso di transito di un veicolo, al contrario, il raggio sarà interrotto per un

Software Mikrobasic



Un potente compilatore Basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

breve istante (dipendente della velocità e dalla lunghezza dello stesso) e l'evento verrà rilevato dal circuito che provvederà al conteggio. Dato che il prototipo viene utilizzato normalmente in zone dove manca totalmente l'alimentazione elettrica, il suo funzionamento è garantito da apposite batterie, naturalmente il circuito deve assicurare il "mantenimento" del conteggio in memoria.

Quant'è veloce?

Dal momento che i dispositivi elettronici sono di gran lunga più veloci dei dispositivi meccanici, praticamente non esistono limiti per la rilevazione di automobili ad altissima velocità. Vediamo di applicare qualche piccolo concetto matematico per la valutazione e l'analisi dei tempi e degli intervalli.

Partiamo da una semplice domanda: durante il passaggio di un veicolo, e conseguente interruzione del raggio luminoso, quanto tempo dura l'inattività del sensore?

Naturalmente tale grandezza dipende da due parametri fondamentali che entrano in gioco: la *velocità* e la *lunghezza* del veicolo stesso. Il prototipo deve poter interpretare come unico transito un intera interruzione di luce, dall'inizio alla fine, e ciò si verifica chiaramente prima e dopo uno stato di illuminazione da parte del laser. Vediamo di calcolare quindi il tempo di tale interruzione con un esempio.

Supponiamo di rilevare un'automobile lunga 3 metri lanciata alla velocità di 90 Km/h. Occorre eseguire una proporzione: se 90 chilometri vengono percorsi in un ora, tre metri in quanto tempo saranno percorsi?

90 Km/h : 1 ora = 3 metri : x

Riduciamo le grandezze alle stesse unità di misura, specificando la lunghezza in **metri** ed il tempo in **secondi**:

90.000 mt : 3.600 sec = 3 mt : x

da cui

x = 3.600 x 3 : 90.000

x = 0,12 (secondi)

Pertanto, durante il passaggio dell'automobile davanti il sensore, il raggio sarà ostruito per 0,12 secondi (oppure 12 centesimi di secondo, che dir si voglia).

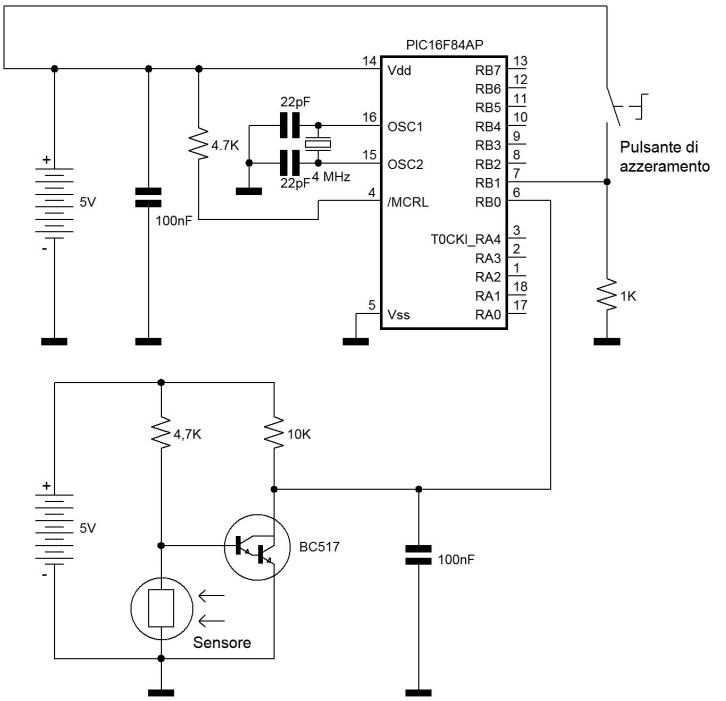


Figura 8 *Schema elettrico del prototipo del Conta Transito*

Il sensore

Il sensore utilizzato nello schema naturalmente è generalizzato. È possibile utilizzare qualsiasi tipo di trasduttore sensibile alla luce, purché sia capace di fornire in uscita la tensione di 5 Volt se illuminato dal laser e circa 0 Volt se non illuminato. Pertanto vari tipi di foto-diodi e foto-resistenze, collegate a circuiti adeguati, vanno senz'altro bene, anche se l'utilizzo del foto-diodo è fortemente consigliato, in quanto tale componente presenta una velocità di commutazione di gran lunga superiore a quella della foto-resistenza.

Schema elettrico

In figura 8 è riportato lo schema elettrico, si notano i due generatori di tensione a 5 Volt, che possono essere sostituiti da uno solo equivalente.

Il sensore di luce è formato dal transistor BC517, dalla foto-resistenza e dalle relative resistenze di polarizzazione. I valori tipici di questo componente (con variazione quasi lineare) vanno da circa 800 Kohm in assenza di luce a circa 200 ohm in piena illuminazione. Quando il sensore è investito dalla luce, la base del transistor BC517 (un darlington dal beta superiore a 30.000) è posta a massa dalla stessa foto-resistenza, si trova pertanto in stato di interdizione ed assicura quindi un potenziale di collettore di 5 Volt (livello Logico 1). Quando invece il raggio luminoso è interrotto dal

passaggio di un veicolo, il valore della foto-resistenza sale a circa 800 Kohm, la base è polarizzata direttamente dalla resistenza di 47 Kohm, ed il transistor commuta in stato di saturazione, abbassando così il potenziale di collettore (verso massa) a 500 mV circa, generando quindi il livello logico 0. Assemblando il modulo del sensore, occorre inserire la foto-resistenza all'interno di un tubetto di plastica nero, per non correre il rischio di illuminarla con altre fonti di luce esterna. Infine il pulsante n/a collegato all'ingresso RB1 serve per azzerare il valore del conteggio, sia in Ram che in Eeprom del PIC.

Il programma

Anche in questo caso il software in Basic è molto semplice ed è riportato nel listato 2.
Dopo aver dichiarato la variabile *veic*, essa è immediatamente inizializzata al valore presente nella cella numero 0 della Eeprom. Questo perché il programma deve poter reperire il numero totale dei veicoli anche alle successive esecuzioni del software. Si passa pertanto alla consueta inizializzazione della PORT-B.
Il cuore del programma è racchiuso in un ciclo infinito *while-wend*. All'interno del quale spiccano due cicli che si ripetono al verificarsi di due condizioni, cioè:

- Il primo ciclo ripete la sua iterazione durante lo stato di illuminazione del sensore da parte del raggio laser pertanto è una sorta di ciclo di attesa.
- Il secondo ciclo invece ripete la sua iterazione durante lo stato di oscuramento, ossia durante il passaggio di un automobile ed è quindi un ciclo provvisorio.

Naturalmente tra i due episodi avviene l'incremento di una unità della variabile di conteggio e la conseguente memorizzazione definitiva nella Eeprom. Gli statements *delay_ms(10)* assicurano

[Listato 2]

```
program Eeprom02
rem  Questo programma costituisce
rem   un prototipo di Conta Transito
rem   e memorizza il conteggio su EEPROM

dim veic as byte              'Contiene numero veicoli
veic=eeprom_read(0)           'Legge contenuto dalla Eeprom
portb=0                       'Azzera PortB
trisb=%00000011               'Configura 2 ingressi e 6 uscite
while true                    'Ciclo infinito
  while portb.0=1             'Mentre laser COLPISCE sensore
    delay_ms(10)              'Pausa antirimbalzo
    if portb.1=1 then         'Se premo tasto AZZERA
      veic=0                  'Azzera variabile in RAM
      eeprom_write(0,veic)    'Azzera valore in Eeprom
    end if
  wend
  rem  Se l'esecuzione del programma arriva QUI
  rem   vuol dire che il raggio laser Èstato
  rem   interrotto e quindi il flusso del progr.
  rem   esce dal primo CICLO WHILE
  veic=veic+1                 'Incrementa numero veicoli
  eeprom_write(0,veic)        'Salva risultato in Eeprom
  while portb.0=0             'Aspetta che il veicolo passa
    delay_ms(10)              'Pausa antirimbalzo
  wend
wend
end.
```

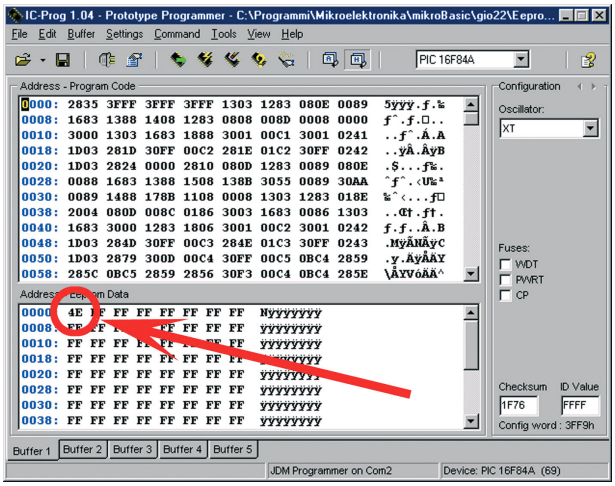


Figura 9 Verifica del conteggio Transito su Eeprom (in esadecimale)

una buona condizione di pausa antirimbalzo. Il listato 2 è abbastanza ricco di commenti, pertanto invitiamo a studiarlo con molta attenzione.

Collaudo ed utilizzo

Per il collaudo basta posizionare il sensore ed il raggio laser uno di fronte all'altro e simulare il passaggio dei veicoli interrompendo il raggio con la mano. Consigliamo di premere inizialmente il pulsante di azzeramento per assicurare il corretto valore iniziale del conteggio.

Si provi ora a simulare il passaggio di automobili, quindi staccate il PIC dal circuito ed effettuate la lettura della Eeprom. In corrispondenza della **prima cella di memoria** vedrete il valore raggiunto sino a quel momento. Riposizionate quindi il PIC sul circuito e simulate nuovamente il passaggio di altre automobili, stavolta **senza** azzerare il conteggio con il pulsante. Rileggendo il valore dalla Eeprom dovreste vedere il totale assoluto del conteggio sino a quel momento (figura 9). Solitamente i software di lettura Eeprom forniscono un valore in notazione **esadecimale**, pertanto convertitelo in decimale.

Il prototipo può trovare un utile impiego come conta-pezzi oppure come conta-persone.

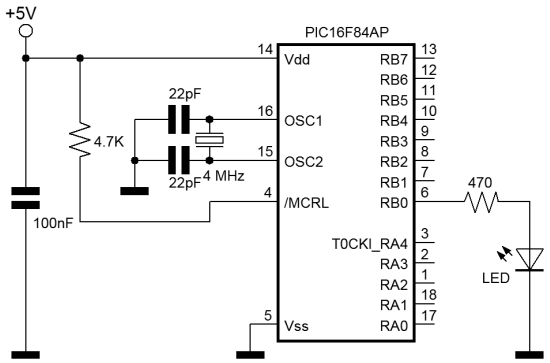


Figura 10 Schema elettrico di un lampeggiatore

Limitazioni

Purtroppo, alla versione attuale del compilatore (2.0), le due istruzioni studiate permettono solamente la gestione di dati di tipo *byte*, quindi da 0 a 255 decimale. Questa è una grossa limitazione poiché, almeno per l'esempio proposto, il numero delle automobili di passaggio supera facilmente tale limite. Per il momento una soluzione è quella di creare un sistema di numerazione composto da due byte (ognuno dei quali risiede in una cella diversa) e trattarli come cifre di un sistema di numerazione a base 256 (byte più e meno significativo).

SECONDO PROGETTO: APPLICAZIONE A TEMPO

Un altro valido esempio di utilizzo è quello di un'applicazione demo. In tali prototipi il sistema funziona perfettamente svolgendo egregiamente i propri compiti, ma…. dopo 30 esecuzioni (per esempio) va in tilt e cessa di funzionare.

Qual'è il trucco?

La nostra applicazione (di qualsiasi tipo) funziona dunque a dovere. Ma ad ogni suo avvio, in maniera del tutto trasparente e nascosta, memorizza in una cella della Eeprom un valore sequenziale (1, 2, 3, ecc) in modo che, arrivato al valore limite prefissato, il software si blocca e non potrà più svolgere le sue normali mansioni. Vediamo i passi basilari da seguire:

1. Solo per la **prima volta**, occorre azzerare la cella della Eeprom nella quale sarà contenuto il contatore.
2. Nella parte iniziale del programma eseguire la lettura da tale cella.
3. Eseguire pertanto l'incremento di uno di questo valore.
4. Memorizzare in Eeprom il valore ottenuto.
5. Se il valore arriva al valore limite previsto (esempio 30) il programma deve entrare in un ciclo infinito.

Il programma

Riferiamoci come esempio ad uno dei primi programmi scritti durante il corso, cioè ad un sempli-

ce lampeggiatore. Il programma (senza protezione demo) è contenuto nel listato 3. Esso funziona regolarmente ad ogni esecuzione e determina il lampeggio di mezzo secondo del diodo led collegato sulla porta RB0. In figura 10 riportiamo lo schema elettrico.
Nel listato 4 riportiamo invece il programma che

[Listato 3]

```
program lampeggio
rem Prototipo che funziona SEMPRE
trisb.0=0            'RB0 come uscita
while true           'Ciclo infinito
  portb.0=1          'Accende LED
  Delay_ms(250)      'Wait 1 sec
  portb.0=0          'Spegne LED
  Delay_ms(250)      'Wait 1 sec
wend
end.
```

esegue lo stesso lavoro del precedente ma che limita il suo funzionamento a soli 30 avvii o applicazioni, poi non funziona più.
Si noti all'inizio del listato la parte relativa al controllo del conteggio raggiunto in Eeprom. Il ciclo infinito di blocco viene eseguito solamente se il conteggio arriva al valore limite.
Occorre prestare molta cura alla programmazione iniziale del software, stando bene attenti ad *azzerare manualmente*, attraverso il programmatore, la locazione della Eeprom nella quale risiede il byte di conteggio.
Per permettere nuovamente il funzionamento per altre 30 volte occorre semplicemente riazzerare manualmente il byte numero 0 della Eeprom.
Per il collaudo del prototipo montate il microcontrollore sul suo zoccolo e date tensione una prima, una seconda, una terza volta e così via. Tutto funzionerà a meraviglia.
Ma la trentesima esecuzione sarà l'ultima, dopo di che il PIC si rifiuterà di operare e il circuito sembrerà rotto o danneggiato.

CONCLUSIONI

Nelle prossime puntate impareremo a memorizzare svariate informazioni in più celle di memoria, in maniera permanente ma soprattutto vedremo come salvare i dati provenienti da dispositivi analogici, quali sensori di luce, di temperatura, di suono e cosi via, al fine di monitorare successivamente i dati acquisiti per una idonea e utile elaborazione statistica.
Buona memorizzazione a tutti!

Codice MIP 247046

[Listato 4]

```
program lampeggio
rem Prototipo che funziona solo per 30 ESECUZIONI
rem '-------- Controllo conteggio ----
dim conta as byte
delay_ms(100)              'Pausa antirimblazdo di accensione
conta=eeprom_read(0)       'Legge la Eeprom
if conta<30 then           'È giunta al limite?
  conta=conta+1            'se NO incrementa
  eeprom_write(0,conta)    'e memorizza
else
  while true               'altrimenti si blocca
  wend                     'in un ciclo INFINITO
end if                     'senza USCITA
rem '--------------------------------
trisb.0=0                  'RB0 come uscita
while true                 'Ciclo infinito
  portb.0=1                'Accende LED
  Delay_ms(250)            'Wait 1 sec
  portb.0=0                'Spegne LED
  Delay_ms(250)            'Wait 1 sec
wend
end.
```

GRANDI PRESTAZIONI

La Microchip garantisce alcuni parametri veramente interessanti, per quanto riguarda l'utilizzo dei propri microcontrollori nel tempo. La memoria **Flash** del PIC (in pratica dove risiede il software in linguaggio macchina) può essere riprogrammata sino a 1.000 volte senza problemi. Ancora più robusta è la **Eeprom**, che può essere riscritta e azzerata sino a 10.000.000 di volte. Tutti i dati infine hanno una garanzia di *prolungato stazionamento*, infatti essi possono rimanere immagazzinati all'interno del chip (anche senza alimentazione elettrica) sino ad un periodo di 40 anni!

Nona parte
n° 247 - Gennaio 2006
Come si memorizzano i dati e le informazioni

Decima parte
n° 248 - Febbraio 2006
La scheda di sviluppo EasyPIC3

Undicesima parte
n° 249 - Marzo 2006
La conversione A/D col PIC

Mikrobasic per PICmicro

Parliamo della EasyPic3, la fantastica scheda di sviluppo per microcontrollori PIC che consente la programmazione, simulazione e sperimentazione con assoluta semplicità.

Si sa, l'appetito vien mangiando. E la stessa regola vale per la programmazione dei microcontrollori. Si inizia ad utilizzare un piccolo programmatore, magari auto-costruito, collegato alla porta seriale del PC. In seguito si approntano delle piccole schede aggiuntive che permettono di simulare il proprio software creato, ma ciò implica la realizzazione di prototipi adatti ad ogni progetto.
E allora si spazia dalla matrice a led al set di pulsanti, dall'array di display a 7 segmenti alle schede di potenza per il pilotaggio di carichi più "robusti".
Sarebbe bello invece possedere ed utilizzare un'unica scheda che consenta di programmare e di provare il microcontrollore senza la necessità di rimuoverlo ogni volta dall'apposito zoccolo.
Oggi questo è possibile grazie alla nuova scheda **EasyPic3**, la quale mette a disposizione un completo ambiente di programmazione e test per i propri programmi.

LE TAPPE DEL SOFTWARE

Come si è visto durante le puntate del corso di programmazione *Mikrobasic*, un software, prima di vedere "la luce", deve attraversare diverse fasi di sviluppo, alcune obbligatorie e altre no, per essere scritto nel migliore dei modi e in maniera professionale.
Questi passaggi devono essere seguiti ed eseguiti al fine di ottenere un programma sicuro, affidabile e, soprattutto, esente da errori e da difetti. In altre parole esso deve risultare assolutamente perfetto!
In figura 2 viene riportato l'iter da seguire durante la stesura di un programma standard. Guardando tale schema, la scheda EasyPic3, oggetto del presente articolo, trova la sua applicazione pratica attraverso i riquadri contrassegnati di colore verde, e precisamente:

- È utilizzata quale *programmatore hardware*, per permettere lo scaricamento del file binario all'interno della memoria flash del PIC.
- È utilizzata come *simulatore* del programma creato, senza dover necessariamente "montare" il microcontrollore sul prototipo finale.
- È utilizzata per *provare* tutte le possibili varianti che si possono verificare nell'applicazione pratica, grazie alla presenza di moltissime unità per l'ingresso e l'uscita del segnale digitale ed analogico.

PANORAMICA GENERALE

Iniziamo a descrivere la scheda per sommi capi, per poi approfondirne le caratteristiche nei successivi paragrafi.

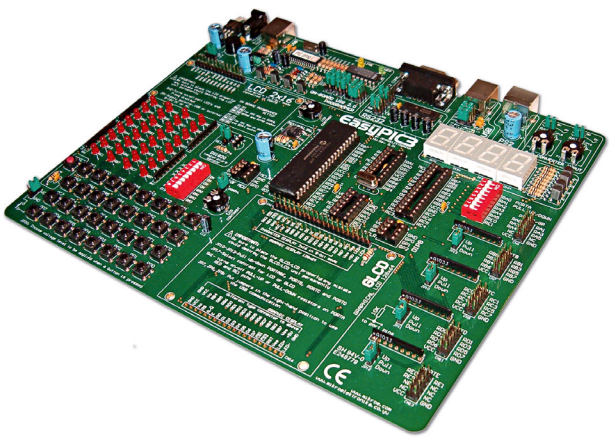


Figura 1 La scheda di sviluppo EasyPic3

La scheda di sviluppo EasyPIC 3



di Giovanni Di Maria

A cosa serve

Riducendo proprio al massimo la descrizione della scheda, essa basa la sua esistenza su due scopi principali:

1. È un *programmatore hardware* di microcontrollori, ossia con essa è possibile memorizzare all'interno del PIC il **file binario eseguibile** generato dal compilatore.
2. È un sistema completo di *simulazione* del microcontrollore programmato, in quanto dispone di un massiccio numero di circuiti che analizzano gli ingressi e le uscite, monitorando nel contempo la bontà del lavoro eseguito.

Le due funzioni sopra elencate procedono assieme senza la necessità di rimuovere il PIC dallo zoccolo, per alloggiarlo magari in un'altra scheda e ciò, indubbiamente, costituisce un ulteriore punto di forza della EasyPic3.

Caratteristiche principali

Veramente notevoli sono le particolarità che caratterizzano la EasyPic3. Elenchiamo le più significative:

- Dimensioni: 19 cm x 23 cm.
- Programmatore USB 2.0 on board.
- Tastiera formata da 32 tasti indipendenti.
- 32 Diodi Led per il controllo e monitoraggio delle uscite digitali.
- 4 display a sette segmenti pilotati da altrettanti transistor.
- Predisposizione per moduli LCD alfanumerici.
- Predisposizione per moduli LCD grafici.
- Predisposizione per comunicazione RS232.
- Predisposizione per tastiera PS/2.
- Predisposizione per sensore di temperatura DS1820.
- Supporto per tutte le famiglie PIC con 8, 14, 18, 28 e 40 pin (anche PIC10F).
- Predisposizione per comunicazione USB.
- Alimentazione esterna o via USB.
- Fornita con PIC16F877.
- Disponibile con o senza display.
- Molte altre ancora…

Ulteriori caratteristiche

Di seguito elenchiamo ulteriori caratteristiche che fanno di questo prodotto un accessorio indispensabile per lo sviluppatore, sia a livello hobbistico che professionale:

- La scheda è dotata di tanti *jumper* con cui si può liberamente personalizzare l'ambiente di lavoro.

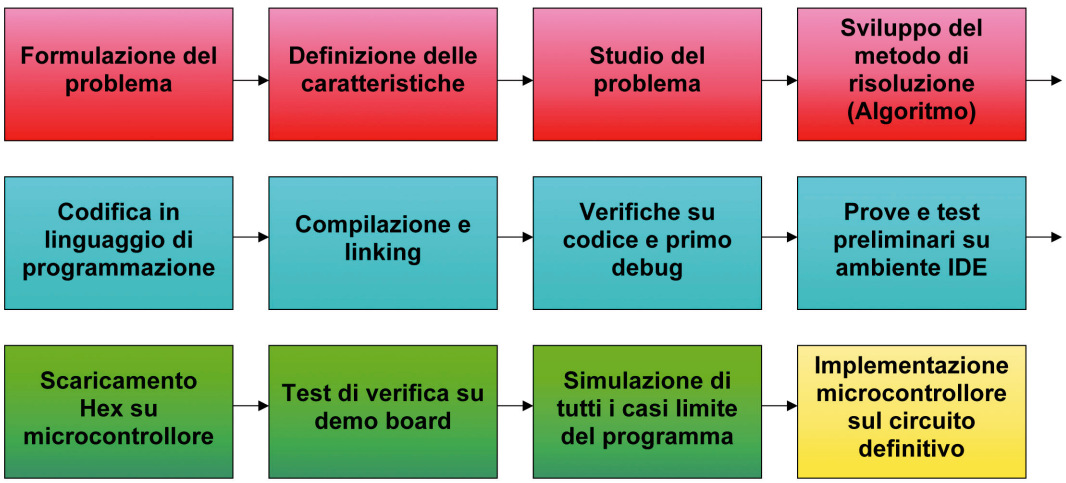


Figura 2 *Le diverse fasi dello sviluppo di un programma*

- Ogni jumper è contrassegnato da una specifica *label* sulla scheda in modo da poterlo identificare immediatamente e comprenderne così la funzione.
- Molti esempi in Pascal, Basic e C sono forniti col sistema.
- Il sistema può essere facilmente configurato secondo le proprie esigenze, agendo sui *dip-switch* esplicitamente contrassegnati sulla scheda.
- Predisposizione per il chip **DS1820** (a 3 piedini) per la misura delle temperature da –55° C a 125° C.
- Dotazione di due potenziometri *on board* che fungono da semplici partitori resistivi e forniscono una tensione continua variabile a piacere da 0 Volt a 5 Volt, con cui provare applicazioni che prevedono l'utilizzo del convertitore A/D.
- I 4 display a sette segmenti sono predisposti per il pilotaggio in *multiplex*.
- Tutte le porte dei PIC sono direttamente collegate a dei connettori, in modo da consentirne il collegamento ad altri dispositivi esterni.
- Possibilità di definire le resistenze di pull-up e pull-down per ogni porta.
- Controllo del contrasto di eventuali display LCD.
- Il quarzo (a corredo della scheda) può essere liberamente sostituito con un altro di diverso valore, in quanto provvisto di zoccolo.
- Tutti i pin sono chiaramente contrassegnati sulla scheda.
- Possibilità di scelta del funzionamento dei pulsanti quando premuti (verso GND oppure 5 Volt).
- Tasto reset on board.

LA SCHEDA EASYPIC3 AL MICROSCOPIO

Possiamo adesso intraprendere un viaggio virtuale al fine di esplorare, dettagliatamente, le particolarità della EasyPic3.

L'alimentazione

La scheda può essere alimentata attraverso due diverse fonti:

- Direttamente dalla porta USB del PC (in questo caso occorre assicurarsi che il PC fornisca corrente sufficiente, in particolare se si utilizza un portatile).
- Con un alimentatore esterno.

Nel primo caso basta collegare la scheda direttamente alla porta USB (tipo 1 o 2) del computer, utilizzando il cavo in dotazione. Inutile dire che non occorre utilizzare l'alimentatore esterno.
Nel secondo caso occorre possedere un alimentatore (o un trasformatore) in grado di fornire una tensione (continua o alternata) compresa tra 7 e 15 Volt.
La tensione continua viene assicurata attraverso il raddrizzatore a ponte presente sulla scheda e relativo condensatore di livellamento mentre i 5 volt richiesti dai circuiti sono ottenuti grazie al regolatore 7805 presente a monte dell'ingresso (figura 3).
La scelta della fonte di energia viene effettuata posizionando il jumper **JP1** (SELECT SUPPLY) sul lato sinistro della scheda (alimentazio-

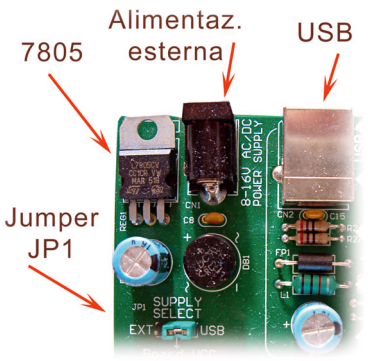


Figura 3 L'alimentazione della schedaEasyPic3

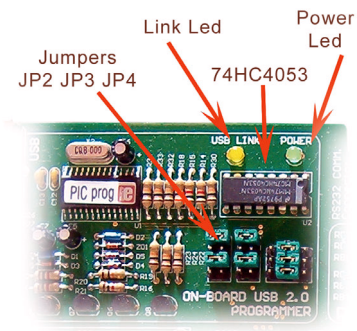


Figura 4 Il programmatore on-board



Figura 5 JP2 e JP3 disposti per DIP18, DIP28 e DIP40

ne esterna) o destro (alimentazione PC USB).
Si notino le descrizioni sulla scheda: esse sono riportate in modo chiaro e leggibile, per cui qualsiasi tipo di etichetta è subito localizzata.
Ricordiamo sempre di effettuare gli spostamenti dei *jumper* e dei *dip-switch* mentre la scheda NON è alimentata. Con questi tipi di circuiti la prudenza non è mai troppa.

Il programmatore USB

Si tratta di un veloce e potente programmatore di microcontrollori PIC *on-board*. Il collegamento avviene tramite la porta USB (di tipo 1 o 2) del PC, assicurando una velocità elevata di comunicazione.
Il punto di forza del programmatore sta nella sua espansibilità, infatti scaricando dalla rete nuovi software e firmare, l'intero sistema può essere aggiornato per la compatibilità con nuovi dispositivi.
Il collegamento è monitorato attraverso due mini diodi led: uno di colore giallo, che mostra la fase di transazione con il software del PC, l'altro di colore verde che attesta il funzionamento generale della scheda.
Nota importante: non confondere il connettore femmina USB situato sulla sinistra della scheda con quello situato sulla destra (anch'esso USB).
Il trasferimento dei dati via bus è garantito dall'integrato 74HC4053, un chip ad alta velocità che si occupa di *multiplexare* e *demultiplexare* i segnali *bufferizzandoli* adeguatamente.
Particolare attenzione meritano i due *jumper* siglati con **JP2** e **JP3**. Essi stabiliscono quale è il tipo di microcontrollore da programmare, selezionandone la corretta posizione. In particolare essi determinano l'utilizzo dei microcontrollori tipo DIP18, DIP28 e DIP40 (figura 5) oppure tipo DIP8 e DIP14 (figura 6).

Termometro Digitale DS1820

É possibile inserire sulla scheda il termometro digitale **DS1820** (non compreso nella confezione). Tale componente, composto solamente da tre piedini, dispone all'interno di un sistema completo per la gestione, la misura e la codifica della temperatura sul suo corpo, con un range di –55° C a 125° C. La precisione dello stesso si attesta su +/- 0,5° C. Molto utile quando si vogliono misurare e/o monitorare le temperature ambientali, anche se ciò comporta la messa in opera della scheda di sviluppo nel luogo da controllare.
Tale termometro offre una deriva termica quasi nulla, almeno nel range di temperatura consigliata (in pratica non potete mettere la scheda dentro il forno per misurarne la quantità di calore, per ovvi motivi...).
Il piccolo chip, dalla forma caratteristica di un piccolo transistor al silicio, va posizionato in modo che la sua parte arrotondata sia posta sulla destra (vedi figura 7). Sulla piastra in ogni caso è ben marcata la corretta posizione. L'orientamento del componente è alquanto critico, in quanto una sua inversione potrebbe causarne addirittura la distruzione.
Il jumper **JP14** posto sulla sinistra stabilisce semplicemente se il pin di comunicazione deve essere collegato alla porta RA5 oppure RE2 del microcontrollore, poiché non si dispone di altre alternative di connessione.

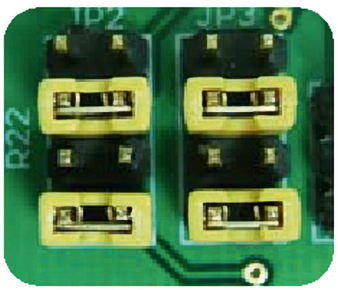


Figura 6 *JP2 e JP3 disposti per DIP8, DIP14*

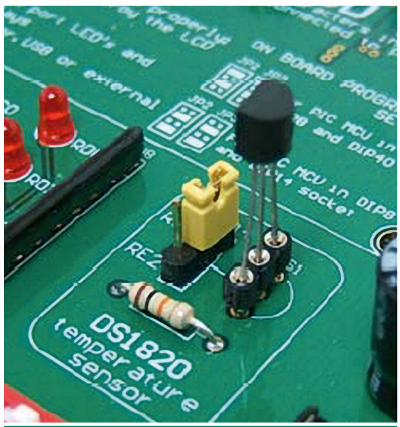


Figura 7 *Lo zoccolo per il termometro digitale DS1820*

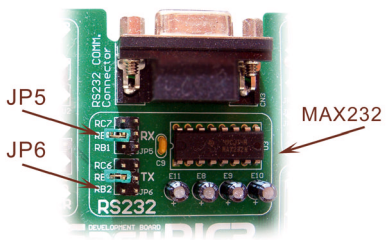


Figura 8 *Dispositivo per la comunicazione RS-232*

Comunicazione seriale RS232

Con la comunicazione seriale **RS-232** si rende possibile il trasferimento dei dati da punto a punto. Di solito viene utilizzata per l'acquisizione dei dati tra applicazioni simili su microcontrollore e Personal Computer.

Dal momento che la tensione della porta seriale del computer è ben diversa da quella di funzionamento del microcontrollore (-12V/+12V contro 0V/5V) occorre che entrambe siano rese compatibili. A questo scopo provvede il circuito integrato MAX-232 montato sulla scheda (figura 8).

Per ottenere un sistema più flessibile, il PIC è collegato al MAX-232 attraverso due jumper: **JP5** e **JP6**. Il primo è utilizzato per trattare, come linea di ricezione (Rx) la porta RC7, RB2 oppure RB1 del microcontrollore. Il secondo invece è utilizzato per considerare, come linea di trasmissione (Tx) la porta RC6, RB5 oppure RB2 del microcontrollore.

Occorre fare attenzione affinché la porta RB2

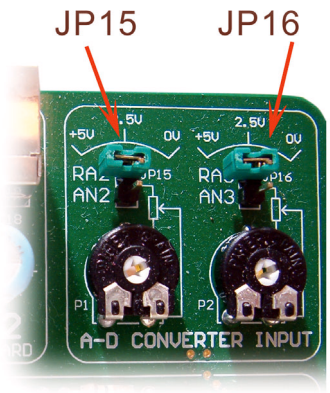


Figura 9 Misure analogiche con microcontrollore

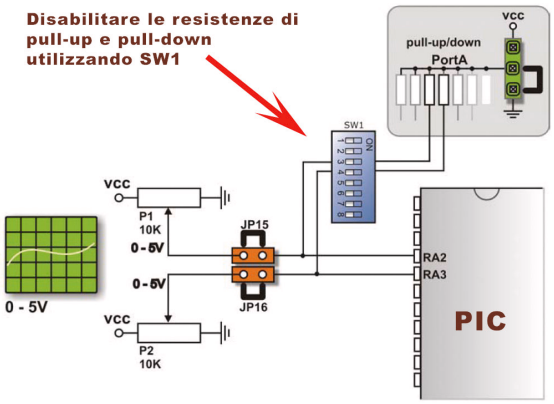


Figura 10 Schema dei potenziometri

Abbonati oggi!



www.farelettronica.com/abbonamento

fare elettronica

CULTURA ELETTRONICA APPLICATA

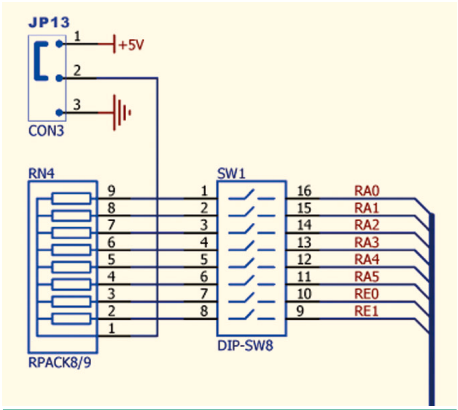


Figura 11 Schema connessioni dello switch SW1

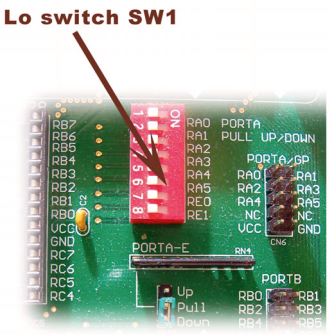


Figura 12 Lo switch SW1 sulla scheda

e un massimo, in maniera semplice e veloce.
Per le misure analogiche è indispensabile disabilitare le resistenze di pull-up/pull-down attraverso lo *switch* SW1 (pin 3 e 4) onde prevenire rilevazioni errate.
La figura 10 mostra chiaramente l'esempio di connessione.

non sia contemporaneamente configurata come linea di ingresso e uscita.

Trimmer per input Analogico

Con i due trimmer (P1 e P2) disponibili on-board è possibile simulare facilmente l'andamento analogico di un segnale (proveniente teoricamente da sensori o trasduttori). Naturalmente il *range* d'azione è quello sopportato dal PIC, vale a dire da un minimo di 0 Volt a un massimo di 5 Volt, variabili con continuità. I nodi centrali dei potenziometri, da cui si preleva il segnale variabile analogico, sono connessi direttamente agli ingressi RA2 e RA3 del microcontrollore, attraverso l'abilitazione dei due jumper **JP15** e **JP16**, che ne permettono il collegamento o l'esclusione (vedi figura 9).
I due potenziometri pertanto fungono da altrettanti partitori resistivi variabili con i quali ottenere una tensione continua tra un minimo

Resistenze Pull su Port A-E

Le Port-A/E di ogni microcontrollore sono collegate, attraverso lo *switch* SW1, ad una rete di resistori, che portano le stesse a massa (GND) oppure a +5Vcc (figura 12).
Se SW1 è posto in posizione ON, la relativa porta (a scelta tra RA0, RA1, RA2, RA3, RA4, RA5, RE0, RE1) sarà connessa a massa oppure al potenziale di alimentazione, attraverso la rete di resistori, che fungeranno pertanto da pull-up o pull-down, in dipendenza della posizione del jumper **JP13**, posto sotto lo switch SW1.
Se SW1 è invece in posizione OFF, le stesse porte saranno libere da connessioni, soprattutto per la loro abilitazione quale ingresso analogico (figura 11).

Resistenze Pull su Port B-C-D

Agendo sui jumper **JP10**, **JP11** e **JP12** è possibile impostare al livello logico alto (ON) oppure

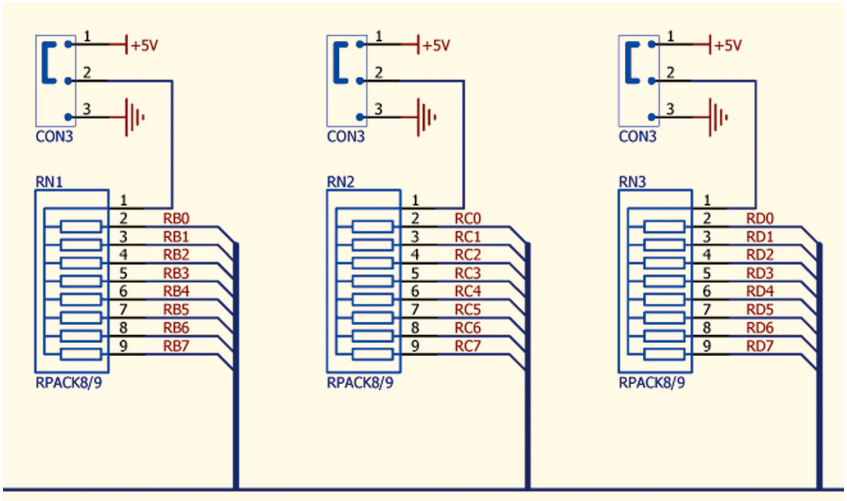


Figura 13 Schema delle connessioni dei jumper JP10, JP11 e JP12

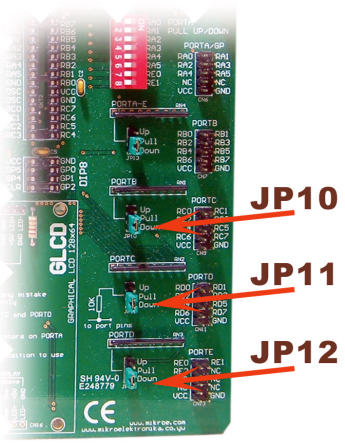


Figura 14 I jumper JP10, JP11 e JP12 sulla scheda

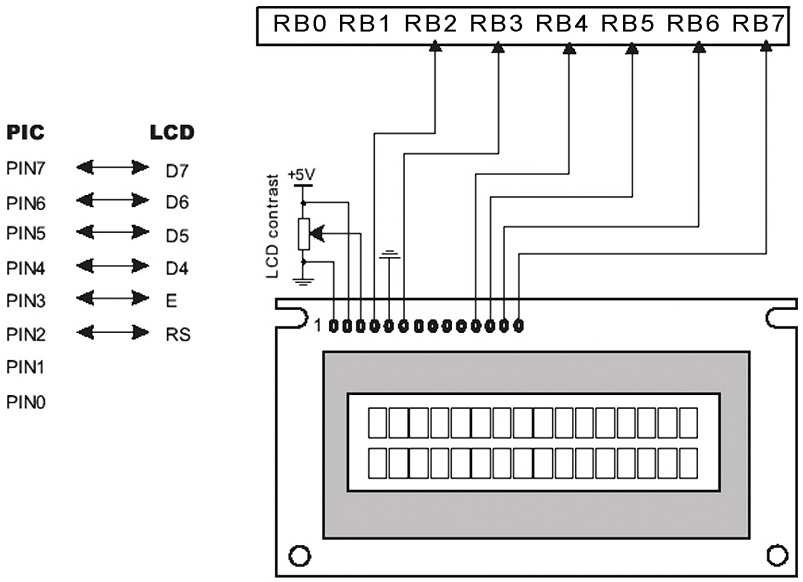


Figura 15 *Connessione PIC-LCD*

basso (OFF) i relativi pin del microcontrollore (figura 13).
Se il jumper si trova nella posizione superiore, i pin saranno posti al livello logico alto, in quanto la rete dei resistori (RN1, RN2 e RN3) è collegata a VCC. Viceversa, se si trova nella posizione inferiore, gli stessi pin si troveranno a livello logico basso (figura 14).
Questa possibilità è molto importante in quanto, configurando le porte del PIC come ingressi, è opportuno "forzare " uno stato logico iniziale (a 0 o 1) per non ottenere dei risultati imprevisti in risposta.

Display LCD

É possibile saldare direttamente sulla scheda, o tramite apposito adattatore, un display LCD *intelligente*. Il display deve permettere la visualizzazione dei caratteri alfanumerici con codifica Ascii organizzati su una matrice di 2 righe e 16 colonne (in pratica 32 caratteri in una sola schermata), in pratica un modulo LCD *standard*. Lo stesso è altresì connesso alla PORT-B del microcontrollore, quindi occorre prevedere tale configurazione all'interno del programma (figura 15).
Sulla lato destro del display è presente un trimmer **(P4)** che ha la funzione di regolarne il contrasto. Costituisce in pratica un partitore resistivo variabile che regola la tensione tra 0 Volt e 5 Volt fornendola al pin 3 del display (figura 16).
Infine occorre dire che la transazione dei comandi dal PIC al display e viceversa avviene con una larghezza di 4 bit (nibble) anziché di 8, facendo così risparmiare ben 4 pin e rendendoli così utilizzabili per altri scopi.
Importante: i display LCD che lavorano in modo 8 bit condividono lo slot dei display grafici LCD (GLCD) pertanto dovranno essere collegati a questa sezione della scheda (vedi paragrafo successivo).

Display LCD Grafico

É possibile utilizzare anche i nuovi display LCD grafici (GLCD) con la possibilità di visualizzare testo ed immagini.



Figura 16 *La zona della scheda dedicata al display LCD*

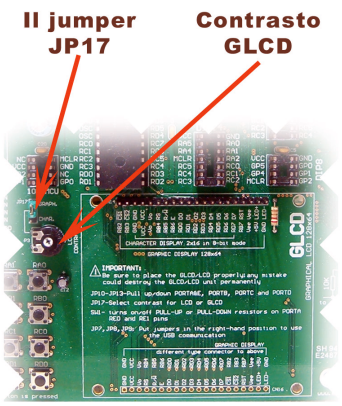


Figura 17 *La zona della scheda dedicata al display grafico*

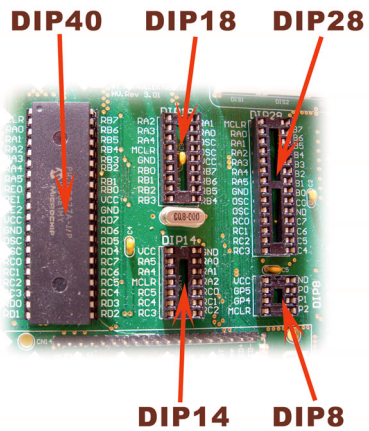


Figura 18 *Alloggiamenti per i microcontrollori*

Si acceda pertanto alla sezione collocata in basso al centro della scheda, con le stesse precauzioni per i normali display.
Il contrasto è regolabile attraverso il trimmer P3. Particolare importanza riveste il jumper **JP17**, che seleziona un particolare display per la taratura: la posizione superiore attiva il controllo del display grafico mentre la posizione inferiore attiva il display testuale (figura 17).
Il display grafico che può essere utilizzato deve essere formato da una matrice di 128x64 pixel.

Molti Mcu Sockets

La scheda è fornita di un microcontrollore PIC16F877A, dotato di ben 40 piedini, su relativo zoccolo. É possibile però utilizzare tutti gli altri modelli, semplicemente montandoli negli altri appositi alloggiamenti, togliendo natural-

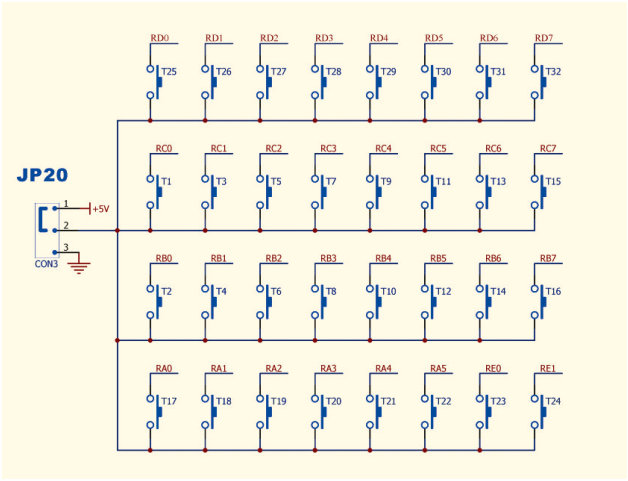


Figura 19 Schema delle connessioni dei 32 pulsanti

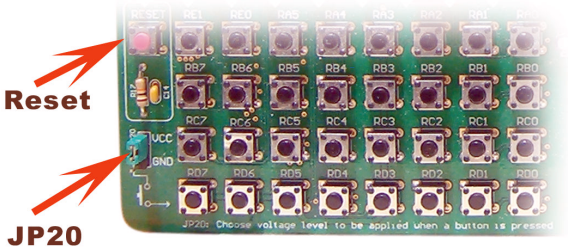
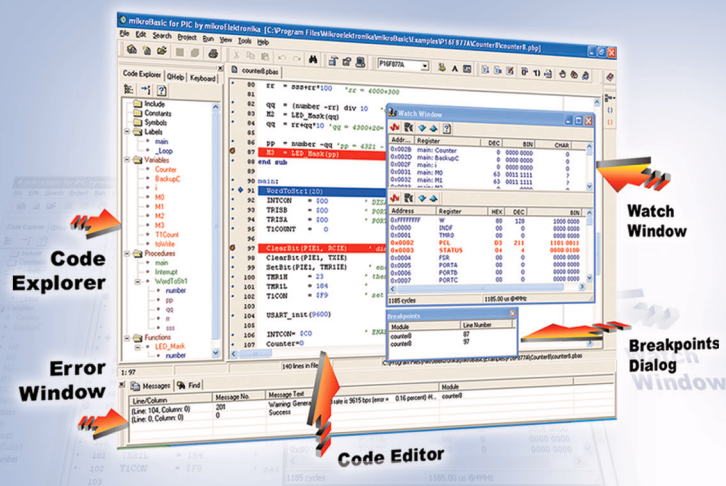


Figura 20 La zone della scheda riservata ai 32 pulsanti

Software Mikrobasic



Un potente compilatore Basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

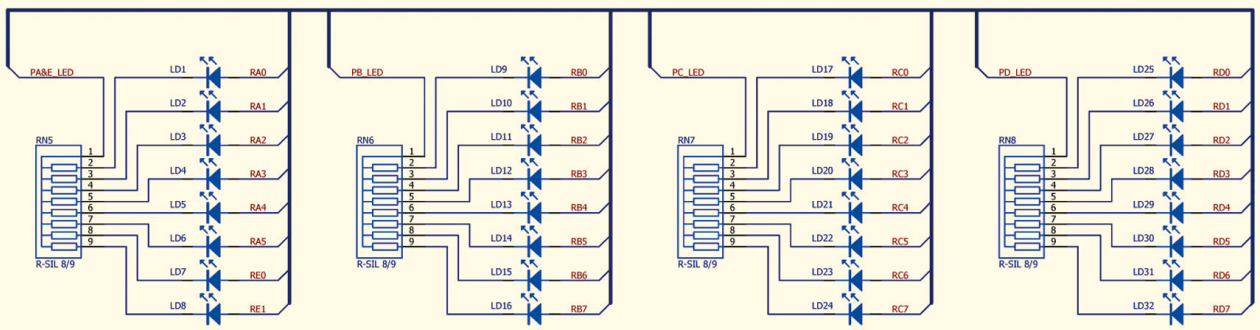


Figura 21 *Schema delle connessioni dei 32 Led*

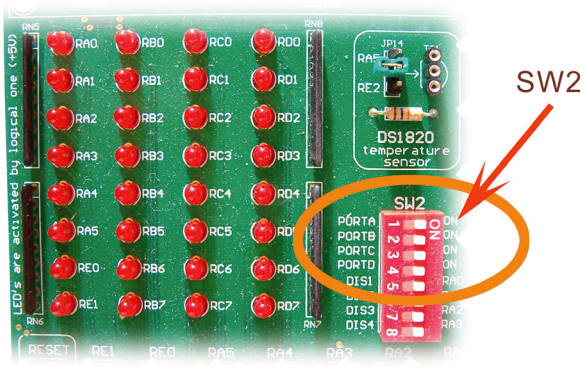


Figura 22 *La zone della scheda riservata ai 32 Led*

mente il principale.
Possono essere pertanto programmati (e testati) anche altri microcontrollori di tipo DIP40, DIP28, DIP18, DIP14 e DIP8, naturalmente bisogna assicurarsi che la disposizione dei piedini sia compatibile alle specifiche della scheda (figura 18).
Dal momento che tutti i dispositivi della scheda (Led, Display, pulsanti ed altro) sono collegati tra loro in parallelo a tutti gli zoccoli dei PIC, può essere montato **solamente** un microcontrollore per volta.

Ben 32 pulsanti

Con questa dotazione di pulsanti, si ha la possibilità del controllo di tutti i pin dei microcontrollore. In altre parole si può inviare un segnale per simulare, in ingresso, un qualsiasi dispositivo digitale, pertanto le porte a cui sono essi collegate devono essere configurate come ingressi.
Tutte le porte vengono considerate in questa circostanza e, alla pressione del relativo tasto, esse saranno collegate a Vcc oppure a massa (per la simulazione di un ingresso alto o basso) come illustrato in figura 19.
Tale distinzione è affidata al jumper **JP20**, che si incarica di scegliere, quale riferimento, 5 Volt oppure 0 Volt. Sulla scheda è presente un ulteriore pulsante, di colore rosso, che ha la funzione di resettare il microcontrollore e iniziare nuovamente l'esecuzione del software programmato (figura 20).

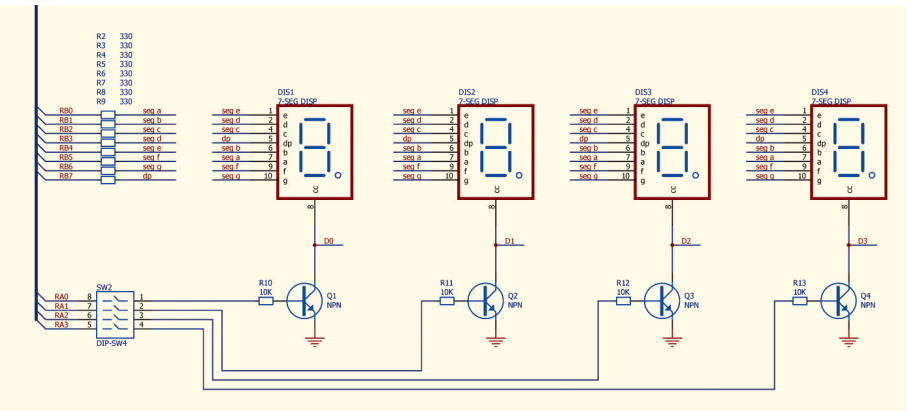


Figura 23 *Schema delle connessioni dei display a 7 segmenti*

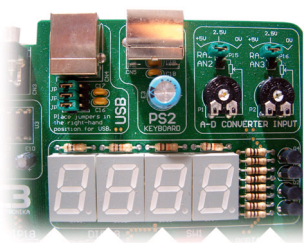


Figura 24 *I quattro display a 7 segmenti on-board*

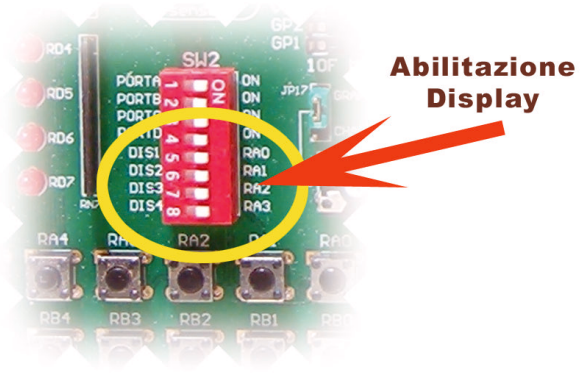


Figura 25 *Abilitazione dei display a 7 segmenti*

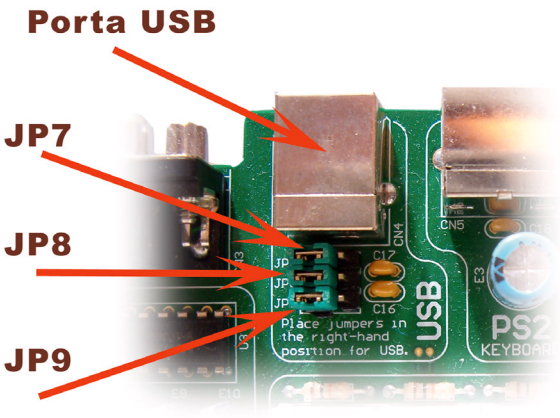


Figura 26 *La porta USB per la comunicazione con il PC*

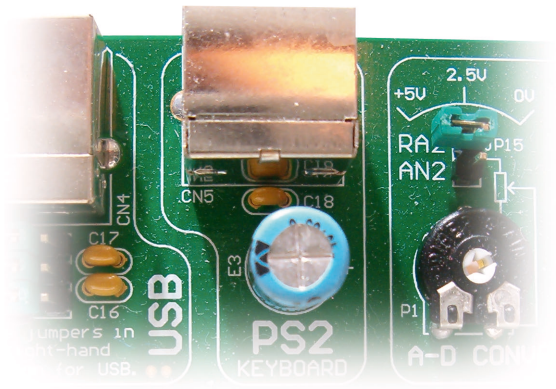


Figura 27 *Il connettore PS/2 per tastiera e mouse*

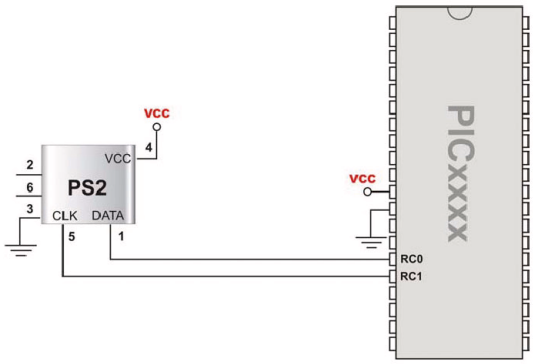


Figura 28 *Connessione PIC/PS2 (dal manuale originale*

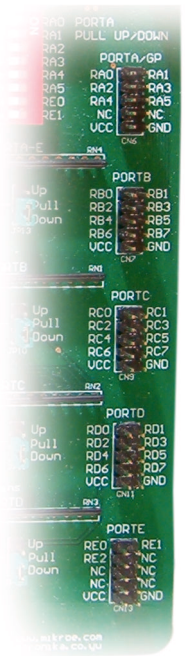


Figura 29 *Connettori esterni delle porte*

Monitoraggio dello stato delle porte (Led)

Tutte le porte dei microcontrollori da testare sono collegate ad un diodo led di colore rosso. Questo significa poter monitorare e controllare lo stato logico di ogni singola uscita, senza dover necessariamente utilizzare un tester. Pertanto, quando un determinato pin del PIC si trova a livello alto, esso illumina, attraverso la relativa resistenza di limitazione, un diodo led, mentre quando esso si trova a livello basso, ovviamente lo tiene spento (figura 21).
La scheda inoltre dà la possibilità di disabilitare un gruppo di led, se la porta stessa deve essere lasciata libera.
Tale opportunità avviene grazie al dip-switch SW2 che, in pratica, connette o meno la relativa rete a massa (figura 22).

Display a 7 segmenti

Il sistema è dotato di quattro display a 7 segmenti, molto utili per presentare numeri e risultati. La visualizzazione avviene con la tecnica del *multiplex* (vedi puntate precedenti). La linea del dato, ossia quella che abilita i sette singoli diodi led, è affidata alla PORT-B, mentre l'abilitazione, sequenziale, dei quattro *digit* è affidata alla PORT-A (figura 23).
I 4 mini interruttori posti nella parte bassa del dip-switch SW2 hanno la funzione di abilitare o meno la visualizzazione del display e, prati-

camente, collegano le basi dei quattro transistor di pilotaggio (BC546), attraverso le resistenze di polarizzazione da 10 Kohm, ai primi quattro pin della PORT-A.

Comunicazione USB

É possibile programmare un microcontrollore per la comunicazione **USB**. Non tutti i PIC supportano questo tipo di protocollo ma solo alcuni modelli, come ad esempio il

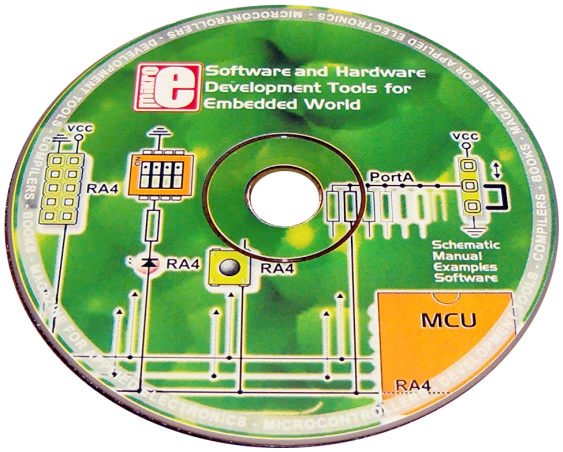


Figura 30 Il Cd-Rom a corredo

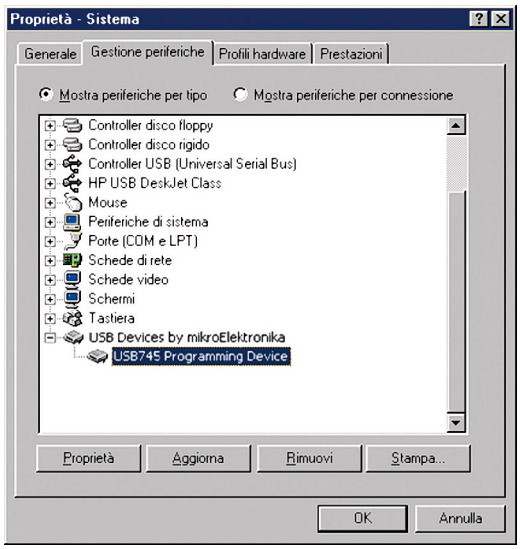


Figura 31 Il nuovo componente installato



Le so tutte!!!

Risposta al quesito codice LST24711

Il circuito è alimentato con una tensione continua per cui i condensatori possono essere considerati dei circuiti aperti, quindi essere rimossi dal circuito ai fini del calcolo della tensione. Rimuovendo i condensatori si nota subito che R5 non ha alcun effetto nel circuito e che R3 ed R4 sono in parallelo tra loro. Essendo di uguale valore, il parallelo tra R4 ed R3 dà come risultato una resistenza equivalente pari a Req=0,5K. La tensione ai capi di C2 (quindi anche di C3) è data dal rapporto di partizione del partitore costituito da R2 ed Req ovvero 12xReq/(R2+Req)=3V. Ovviamente la tensione ai capi di C1 è la differenza tra la tensione di alimentazione e quella ai capi di C2 quindi 9V.

Il vincitore di DICEMBRE 2005 (Vincitore LST24610)
I nostri complimenti a Diego Bellachioma di Villanova di Guidonia (RM)
che vince la Felpa di Fare Elettronica!



Figura 33 Dotazione della confezione



Figura 32 La scatola della EasyPic3

PIC18F2450 e il PIC18F4550. Attenti a non confondere il connettore USB di comunicazione (posto in alto a destra) con quello per la programmazione del PIC (posto in alto a sinistra).
In figura 26 possiamo vedere la porta USB dedicata alla comunicazione.
Per abilitare la comunicazione tra il microcontrollore e la porta USB del PC occorre che i tre jumper (JP7, JP8 e JP9) si trovino a destra.

Connettore per comunicazione PS/2

Il connettore **PS/2** (posto tra il connettore USB e i due trimmer per la simulazione del segnale analogico) permette il collegamento diretto tra la scheda di sviluppo e un dispositivo PS/2 per Personal Computer come, ad esempio, una tastiera o un mouse.
É possibile infatti connettere una tastiera per immettere dati nel microcontrollore, permettendo quindi una più ampia possibilità nella scelta dei caratteri Ascii. Il connettore è visibile in figura 27.

Il Mikrobasic è dotato di tre potenti funzioni che gestiscono interamente e con poco sforzo la comunicazione PS/2:

- **Ps2_Init**, che inizializza la porta del PIC per l'utilizzo della stessa.
- **Ps2_Config**, che inizializza la porta del PIC con maggiori opzioni.
- **Ps2_Key_Read**, che legge direttamente un tasto dalla tastiera fornendo in output un codice corrispondente.

Per il collegamento PIC/Tastiera bastano semplicemente quattro connessioni, come mostrato dallo schema applicativo in figura 28.

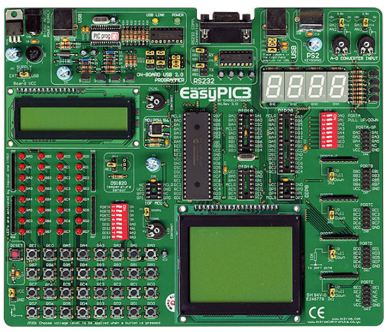


Figura 34 EasyPic3 versione "D"

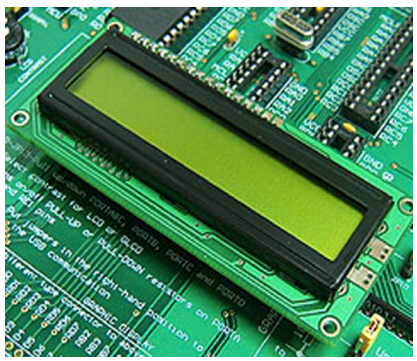


Figura 35 Particolare del display alfanumerico



Figura 36 Particolare del display grafico

Accesso diretto alle porte

Per un ulteriore e completo controllo delle porte (ma già la scheda è completamente autosufficiente) è possibile accedervi direttamente attraverso alcuni connettori per consentire un collegamento esterno semplice ed immediato.
Tali connettori, formati da 10 pin, sono situati sulla destra della scheda e consentono il collegamento anche a *Vcc* e *Gnd*.
Molto utili per permettere anche il collegamento delle uscite a transistor di potenza, voltmetri ed oscilloscopi, pulsanti esterni, relais, sensori situati all'esterno della scheda (figura 29).

CONTENUTO DEL CD

Il Cd-Rom a corredo della scheda (figura 30) è veramente ricco di materiale, informazioni ed esempi utili per il progettista, al quale si consiglia vivamente di consultarlo attentamente.
In dettaglio, ecco il contenuto:

- Documentazione delle caratteristiche generali della scheda.
- Ricchissimo manuale di tutte le funzionalità e schemi di connessione.
- Descrizione dettagliata delle varie componenti della scheda.
- Documentazione sull'installazione dei driver.
- Software MPLAB con relativo manuale.
- Tantissimi esempi scritti in MikroC, MikroPascal e MikroBasic.
- Software PIC Flash Programmer.
- Driver per sistemi operativi Windows 98, 2000, NT e XP.
- Tutorial su microcontrollori PIC ed in particolare sull'Assembler.
- Tutorial sul Basic per microcontrollori PIC.
- Ulteriori schemi applicativi con relativi esempi.
- Versioni demo e manuali del MikroC, MikroPascal e MikroBasic.

INSTALLAZIONE DEI DRIVER USB DEL PROGRAMMATORE

Per poter funzionare correttamente, il programmatore ha bisogno di un driver che ne permette l'interfacciamento con il sistema operativo del PC.
Fortunatamente la procedura d'installazione è estremamente semplice, dal momento che essa avviene in modalità guidata (wizard).
Occorre seguire solamente alcuni semplici passaggi e rispondere alle domande che il sistema propone:

1. Con il computer acceso occorre collegare la porta USB del PC al relativo connettore della scheda, in alto a sinistra (vedi figura 3), con il cavo in dotazione.
2. Il sistema si accorgerà del nuovo componente e chiederà l'installazione dei nuovi driver. Occorre selezionare la voce *"Install from a list or specific location (Advanced)"*, quindi cliccare su **Next**.
3. Selezionare pertanto la cartella del Cd-Rom nella quale sono collocati i driver per il proprio sistema operativo.
4. Confermare la successiva schermata rispondendo **Yes**.
5. Al termine dell'installazione occorre cliccare sul pulsante **Finish**.

I driver adesso si trovano correttamente instal-

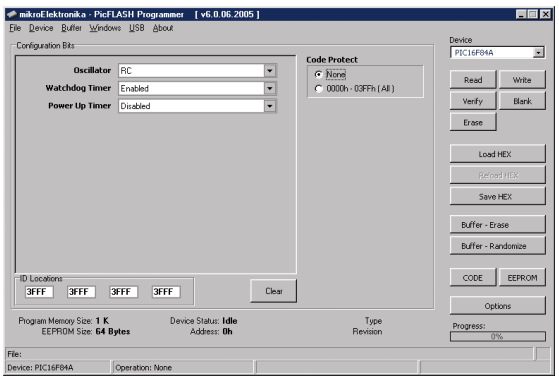


Figura 37 Il PIC Flash Programmer

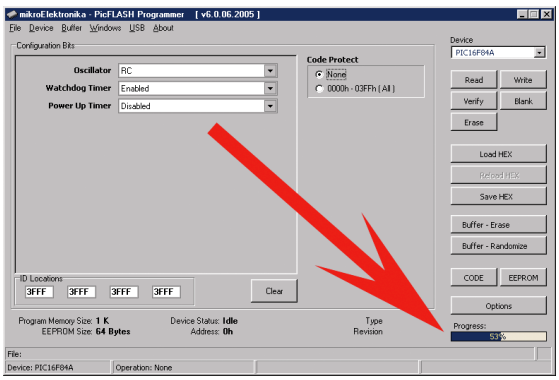


Figura 38 Programmazione in corso del PIC

lati ed il sistema operativo li riconosce.
Per verificare ciò occorre accedere alle risorse del sistema e controllare se effettivamente il nuovo componente risulta nella lista (figura 31).

LA DOTAZIONE

L'elegante scatola (figura 32) contiene tutto l'occorrente per la preparazione e l'uso della scheda. In dettaglio la dotazione (figura 33) è composta da:

- Scheda di sviluppo EasyPic3 con microcontrollore PIC16F877A.
- Cd-Rom con driver, software, esempi, ecc.
- Opuscolo con schemi di collegamento e connessioni.
- Cavo USB per collegamento al PC.
- Cavo seriale tipo 9 pin M/F.

EASYPIC3 VERSIONE D

Esiste in commercio anche la versione "D" della scheda, identica alla standard, ma con inclusi due moduli LCD, uno alfanumerico e l'altro grafico (figura 34).

UNA PROVA PRATICA

Bene, ora che abbiamo conosciuto bene il software e l'hardware che compongono la scheda EasyPic3, siamo pronti per una prova pratica guidata. Il passaggio più delicato è proprio quello della programmazione del microcontrollore ma, seguendo semplici regole, è possibile concludere questa fase nel migliore dei modi.
Elenchiamo i passi da seguire:

1. Bisogna avviare, come prima operazione, il compilatore Mikrobasic.
2. Il secondo passo da effettuare è naturalmente quello della scrittura del codice Basic del programma. Il lettore dovrebbe ormai aver familiarizzato abbastanza con questa procedura.
3. Occorre a questo punto compilare il programma, selezionando la voce **Build** dal menù **Project** o, in alternativa, cliccare sul pulsante raffigurante due ingranaggi, dalla barra degli strumenti.
4. Se la compilazione e il linking del programma vanno a buon fine, nella cartella di lavoro scelta dall'utente viene creato un nuovo file, con estensione **.HEX**, da copiare materialmente all'interno del microcontrollore. Si tratta del vero e proprio file eseguibile da caricare attraverso il programmatore.
5. Dal Cd-Rom allegato alla scheda EasyPic3, occorre lanciare il programma **Picflash2.exe** che si trova nella cartella *\picflash*. Si tratta del software di programmazione per microcontrollore aggiornato (versione 6.0.0), pertanto si consiglia questa soluzione (figura 37).
6. Si deve selezionare il tipo di microcontrollore utilizzato, aprendo il menù a tendina *device*, in alto a destra.
7. Occorre aprire il file .Hex generato dal compilatore, selezionando la voce **Load HEX** dal menù **File**.
8. Finalmente bisogna programmare il microcontrollore selezionando la voce **Write** dal menù **Device**. Con questa fase il file .HEX generato dal compilatore sarà fisicamente copiato all'interno del PIC.
9. Un'apposita barra mostrerà l'avanzamento del processo (figura 38).
10. Il microcontrollore è adesso programmato ed il flusso di esecuzione del software inizierà immediatamente.

Si ribadisce ancora una volta che tutto il lavoro di programmazione ed esecuzione codice avviene senza la necessità di rimuovere il microcontrollore dal proprio alloggiamento.

CONCLUSIONI

La EasyPic3 è sicuramente un utile tool di sviluppo che semplifica enormemente la progettazione e test dei propri prototipi. Il grande punto di forza, sta proprio *nell'integrazione modulare*: tutto ciò che serve per programmare, testare e simulare il microcontrollore si trova in un'unica scheda e non occorre rimuovere il chip programmato per altre operazioni. Uno strumento utilissimo pertanto che non può mancare assolutamente nel laboratorio del programmatore di microcontrollori.
Maggiori informazioni sulla EasyPIC3 le potete trovare sul sito di Fare Elettronica, nella sezione Shop, dove potrete acquistarla a condizioni vantaggiose e completa di tutte le sue espansioni.

Codice MIP 248044

Decima parte
n° 248 - Febbraio 2006
La scheda di sviluppo EasyPIC3

Undicesima parte
n° 249 - Marzo 2006
Conversionie analogico/digitale con il PIC

Dodicesima parte
n° 250 - Aprile 2006
I²C con il PIC

Mikrobasic per PICmicro

Aggiungiamo un importante tassello alla conoscenza dei microcontrollori, cioè lo studio della conversione analogico/digitale di un segnale. Un passo obbligatorio per poter gestire trasduttori e sensori che forniscono solamente tensioni di qualsiasi livello analogico.

Il PIC, lo abbiamo detto tante volte, accetta in input solamente due valori di tensione: 0 Volt rispetto a massa (*Falso digitale*) e +5 Volt rispetto a massa (*Vero digitale*).
Se tale gestione è sicuramente semplice per il trattamento di impulsi o segnali provenienti da apparecchiature digitali, è altrettanto vero che l'acquisizione di un segnale dall'ampiezza variabile o incerta sarebbe impossibile.
Dal momento che un microcontrollore è capace di "leggere" solamente queste due sole tensioni, è obbligatorio il trattamento di questi due livelli.
Esistono naturalmente piccole differenze che il PIC può tollerare, e questo vale anche per tutte le porte logiche esistenti.
Così, ad esempio, se in una porta si immette un valore di 4,6 Volt, il dato acquisito sarà **Alto**, digitalmente parlando. Al contrario, se si immette un valore minore di 0.6 Volt esso sarà considerato come **Basso**. Ma se il segnale da considerare è di 2,7 Volt il sistema si troverebbe in grande imbarazzo e non saprebbe come interpretarlo. Si definisce questa situazione come *stato di incertezza* che conviene sempre evitare in tutti i sistemi digitali.
Esistono invece casi in cui è indispensabile *processare* il valore reale del segnale, seguendo il suo andamento da un minimo a un massimo, ed elaborarlo per la sua quantità effettiva, senza perdita di informazioni.

LA CONVERSIONE ANALOGICO/DIGITALE (ADC)

È un metodo che consente di convertire un segnale analogico in uno digitale, interpretabile facilmente da un sistema digitale.
In altre parole la tensione proveniente da un trasduttore ideale, con andamento non limitato, probabilmente causale e imprevedibile, è convertito in una serie di valori (0 e 1) gestibili facilmente da un processore.
Fa parte di quella categoria di processi a *perdita di informazione*, poiché nella conversione si lascia sempre qualche informazione a discapito della precisione, anche per piccolissime entità.
Un segnale analogico infatti è molto più affida-

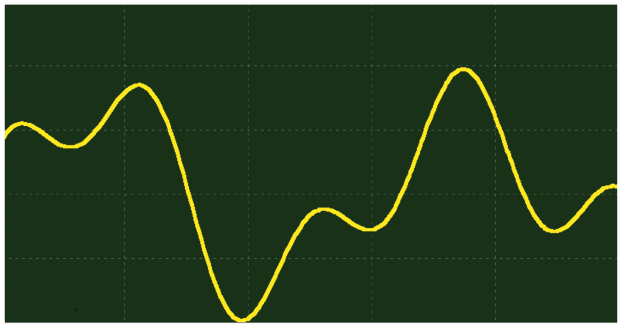


Figura 1 Tipico segnale analogico

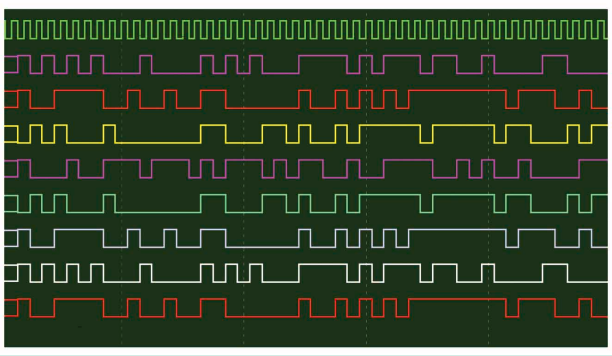


Figura 2 Tipico segnale digitale

Conversione analogico/digitale con il PIC



di Giovanni Di Maria

bile e completo di uno digitale e contiene tutto ciò che esso deve poter riprodurre e rappresentare e il fatto di limitarlo a poche o molte informazioni diversificate, implica gioco forza un abbassamento di qualità e di fedeltà. Volendo fare un paragone un po' spartano è come se si guardasse uno foto a colori (segnale analogico) in modalità bianco e nero (segnale digitale).

TEORIA

Se il microcontrollore è capace di "capire" solo due tipi di tensioni diverse, (0 Volt e 5 Volt) come si può far ad esso comprendere l'esistenza di un valore diverso e intermedio, come ad esempio 3,3 Volt?
Con le tecniche di *conversione* questo risultato è possibile, ma il prezzo che si paga è un aumento della difficoltà circuitale e della criticità del sistema.
Insomma, per garantire un'elevata precisione occorre utilizzare metodologie alquanto sofisticate che si traducono anche in un aumento del costo dell'intero sistema.
Elenchiamo alcune tecniche di conversione analogico/digitale, senza approfondirle in questa sede:

1. A conversione diretta (con molti comparatori in parallelo).
2. Ad approssimazioni successive.
3. A rampa semplice con integrazione.
4. A doppia rampa con integrazione.

CONVERSIONE PER APPROSSIMAZIONI SUCCESSIVE

Approfondiamo solamente questa tecnica di campionamento in quanto utilizzata dai microcontrollori PIC.
Si può anche chiamare *tracking A/D* oppure conversione *A/D ad inseguimento*.
In pratica viene utilizzato un convertitore D/A per fornire una base precisa di riferimento. Attraverso un comparatore analogico i due segnali vengono confrontati in maniera continua, pertanto se quello generato dal D/A è troppo basso, esso viene aumentato e viceversa, sino ad adeguarsi con il valore del segnale in ingresso.
Questo ripetersi di "inseguimenti" consente di ottenere delle conversioni molto precise in tempi davvero bassi (figura 3).

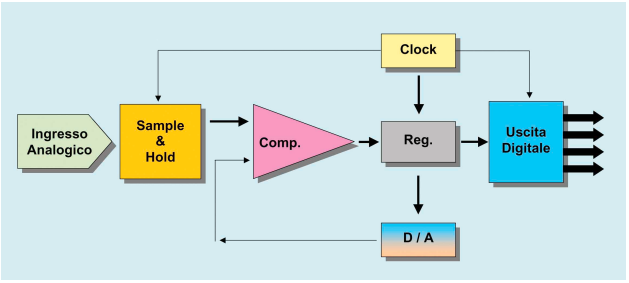


Figura 3 *Schema a blocchi della conversione per approssimazioni successive*

LA CONVERSIONE NEL PIC16F876

Per i nostri esperimenti utilizzeremo il fratello maggiore del **PIC16F84**, ovvero il modello **PIC16F876**, potente ed economico, ma soprattutto dotato di molte porte (anche di ingresso analogico).
Ecco come avviene la digitalizzazione in questo microcontrollore: il segnale da quantizzare viene applicato ad un piedino del PIC e, tramite un semplice condensatore, esso si mantiene stabile e senza fluttuazioni repentine, che renderebbero la lettura molto più critica (tale mantenimento di livello è definito *sample and hold*).
Il segnale viene così convertito attraverso la tecnica delle approssimazioni successive (spiegata prima) e resa disponibile all'utente in formato digitale, con una risoluzione di 10 bit.
Ciò vuol dire che la precisione del segnale (con V_{min}=0V e V_{max}=5V) equivale a $5/2^{10}$ ossia a 0,00488 Volt (4,88 mV), una risoluzione più che accettabile per la maggior parte delle applicazioni. Il PIC dispone inoltre del controllo dei riferimenti minimi e massimi del segnale, attraverso la combinazione di VDD, VSS, RA2 e RA3.

IMPORTANTE: La massima impedenza raccomandata dalla Microchip per la sorgente analogica è 10 Kohm.
Il PIC16F876 dispone di ben 5 ingressi analogici, con cui poter gestire fino cinque sensori analogici.
Un numero veramente elevato che consente in pratica di risolvere la maggior parte delle esigenze tecniche.

CARATTERISTICHE TECNICHE DEL PIC16F876

Per l'acquisizione di segnali analogici utilizziamo il microcontrollore PIC 16F876 (figura 7), poiché il 16F84 non dispone di convertitori adatti allo scopo.
È opportuno pertanto elencare le caratteristiche essenziali di questo chip veramente potente:

1. Frequenza del clock fino a 20 Mhz.
2. Basso consumo.
3. Contenitore da 28 pin DIL.
4. Memoria Flash per il programma da 8K.
5. Memoria dati da 368 bytes.
6. Eeprom da 256 bytes.
7. Porte I/O: A (6 bit), B (8 bit), C (8 bit).
8. 3 Timers.
9. 5 Canali analogici di input a 10 bit.

SCHEMA ELETTRICO APPLICATIVO

Le nostre prove saranno eseguite adottando uno schema generico (figura 4) idoneo alla maggior parte delle applicazioni tipiche.
Il lettore eventualmente personalizzerà tale schema adattandolo alle proprie esigenze.

Il numero di connessioni e componenti esterni è ridotto veramente al minimo e, almeno per queste prove, non è richiesto alcun circuito stampato ma basta semplicemente una basetta mille fori o la scheda **EasyPIC 3** presentata nella scorsa puntata.
Naturalmente merita una nota particolare il generatore di tensione, indicato con la lettera G e collegato alla porta RA0 del PIC.
Tale ingresso è di tipo analogico ed è capace di "leggere" una tensione analogica, con valore positivo variabile nel tempo.
Si comporta come se fosse una sorta di "tester" interpretato e decodificato dal microcontrollore.

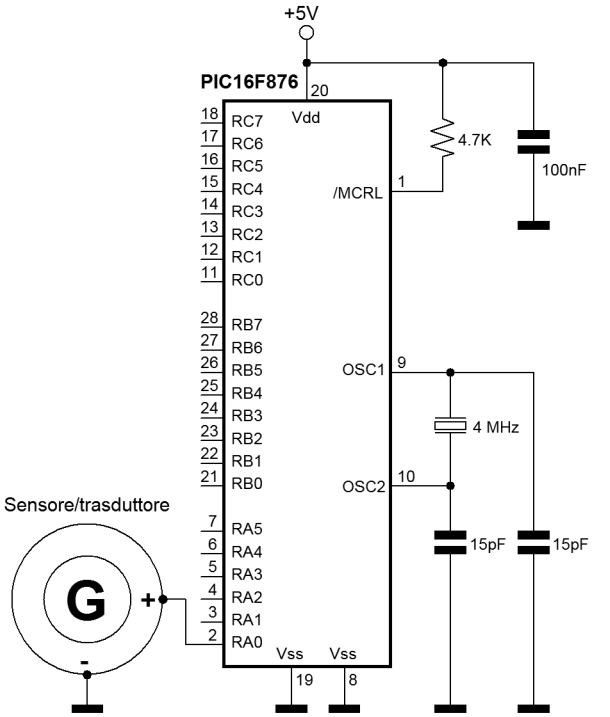


Figura 4 Schema elettrico di base per gli esperimenti

Cosa si può collegare a tale porta?
Praticamente quasi tutti i tipi di sensore (luce, temperatura, pressione, velocità, livello sonoro, eccetera) che sappiano fornire in uscita una tensione qualsiasi e tutti i tipi di generatori di tensione (pile, trasformatori, celle fotovoltaiche ed altri).
Addirittura potremmo collegare ad essa una semplice batteria da 4,5 Volt per monitorare la sua tensione attraverso dei Led colorati e controllare se questa è efficiente, mediamente scarica o è da gettare definitivamente (nel proseguo dell'articolo è descritto proprio questo progetto).
Naturalmente le specifiche del sensore/generatore dovranno seguire quelle del microcontrollore: in altre parole occorre rispettare i valori minimi e massimi tollerabili da quest'ultimo.

DIETRO LE QUINTE

Un microcontrollore preposto alla conversione analogico/digitale svolge, all'interno del suo sistema, numerose funzioni di *quantizzazione, calibrazione* e *raffronti,* funzioni sicuramente molto complesse e complicate, con codice molto esteso. Inoltre sono coinvolti numerosi *registri* interni del microcontrollore, ognuno dei quali preposto ad una precisa funzione per la conversione ed il controllo.

Naturalmente questo processo è "nascosto" dal compilatore Mikrobasic in quanto questo si occupa di gestire il tutto riducendone il controllo a poche istruzioni ad alto livello. Ciò costituisce un vantaggio dal punto di vista pratico, perché il programmatore in questo modo non ha bisogno di conoscere l'architettura del controllore. L'ideale sarebbe conoscere il microcontrollore e la sua costituzione a basso livello, saper programmare anche nel linguaggio Assembler ma sviluppare le applicazioni tramite compilatore Mikrobasic per ottenere una maggior velocità nello sviluppo.

LE FUNZIONI BASIC PER L'A/D

Fortunatamente il Mikrobasic è dotato di una sola funzione per il campionamento, che si occupa totalmente della gestione della conversione, generando all'interno del compilatore un codice Assembler davvero sofisticato, liberando ancora una volta l'uomo da questa fatica.
La funzione preposta al campionamento è **Adc_Read** che inizializza il modulo interno ADC e prevede la seguente sintassi:

```
sub function Adc_Read(dim channel as byte)
as word
```

dove:

- **sub function** indica l'utilizzo di una funzione, con un ritorno di valore al programma chiamante.
- **Adc_Read** è il nome della funzione vera e propria.
- **Channel** è un valore di tipo byte (0-255) che indica la porta dalla quale si effettua la lettura.
- La funzione ritorna una valore di tipo word, contenente il risultato del campionamento a 10 bit.

Esempio di utilizzo:

```
volt = Adc_Read(1)
```

Naturalmente, per eseguire una lettura da una porta, non basta scrivere questa funzione. Occorre anche commutarla in *input* e specificare che lo stesso input sia di tipo analogico e non digitale, attraverso il registro **Adcon1**. Gli esempi risulteranno tuttavia molto semplici ed esplicativi. Adesso andiamo a fare pratica con la conversione analogico/digitale attraverso degli esempi veramente utili e divertenti.

LA CONVERSIONE A/D IN PRATICA

Quanto il microcontrollore "legge" una tensione analogica, esegue con delle sofisticate operazioni una *conversione digitale*. Il valore ottenuto è rappresentato da un insieme di bit (solitamente 8, 10, 12, 16 o più). Naturalmente, più sono i bit in questione più preciso sarà il valore acquisito; in altri termini aumenta la risoluzione del dato campionato consentendo una lettura più dettagliata nei conteggi.
Ma come si "pesa" praticamente il valore in input e, soprattutto, come si confronta con altri di valore noto, al fine di far prendere al processore decisioni logiche e comportamenti conseguenti?
Ricorrendo sempre alla matematica, con dei pratici e semplici calcoli.
Il primo parametro da conoscere è la *risoluzione*, ossia a quanti bit il processore deve campionare il segnale. Supponiamo che esso sia di 8 bit. È noto che dalla numerazione binaria, con otto bit si possono formare e rappresentare 256 valori diversi, infatti 2^8 dà 256 combinazioni formate dai simboli 0 e 1.
Pertanto con questa risoluzione si possono ottenere 256 "divisioni" della tensione massima: non una in più.
Il secondo parametro da conoscere è il *range* della tensione da misurare: un valore V_{min} determina la tensione minima teorica possibile, anche negativa se necessario.
Solitamente è fissato a 0 Volt, ossia al potenziale di massa; un valore V_{max}, definito anche potenziale di riferimento, o V_{ref}, determina invece la tensione massima teorica possibile, solitamente fissata a 5 volt e, in ogni caso, compatibile con le specifiche del microcontrollore.
Si supponga che la tensione da misurare sia di 3,3 Volt. Quale rappresentazione digitale avrà questo valore? Basta applicare una semplice equazione:

$$V_{ref} : VDig_{Max} = V_{In} : x$$

dove:

- V_{ref} è il potenziale di riferimento o massimo, solitamente fissato a 5 Volt.
- $VDig_{Max}$ è il massimo valore digitale teorico possibile e con una risoluzione di 8 bit esso ammonta a 255.
- V_{In} è il potenziale di ingresso (proveniente dal trasduttore).

da cui si ricava:

$$5 : 255 = 3{,}3 : x$$

quindi:

$$x = 168$$

Pertanto il valore 168 (senza decimali) è il valore digitale corrispondente al valore analogico 3,3 Volt in ingresso. Il programmatore deve riferirsi ad esso per creare condizioni di scelta all'interno del programma scritto. In tabella 1 è presente una comparazione tra valori analogici e digitali con varie risoluzioni.

Tensione da acquisire (Volt)	Valore digitale decimale (8 bit)	Valore digitale decimale (10 bit)
0,0	0	0
0,5	26	102
1,0	51	205
1,5	77	307
2,0	102	409
2,5	128	512
3,0	153	614
3,5	179	716
4,0	204	818
4,5	230	921
5,0	255	1023

Tabella 1 *Corrispondenze tra valori analogici e digitali nella conversione A/D*

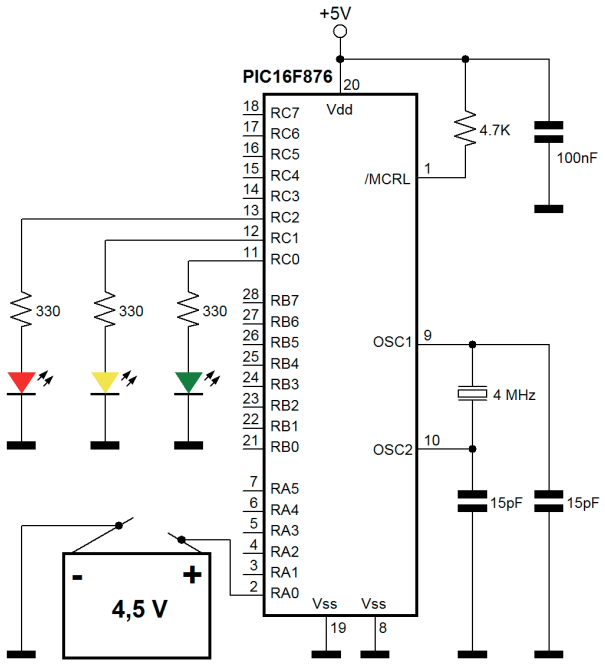


Figura 5 *Schema elettrico del monitor per batteria*

PRIMA ESPERIENZA: MONITOR DI LIVELLO PER BATTERIA

Un microcontrollore si presta ottimamente per questo tipo di applicazioni: un monitor che segnali lo stato di una batteria di 4,5 Volt fornendo la segnalazione attraverso tre diodi led colorati.
Senza la sua presenza occorrerebbe realizzare un sofisticato circuito dotato di almeno tre operazionali collegati a finestra multipla per comparare la tensione e fornire la risposta all'operatore.
Ma il nostro circuito fa molto di più.
In dettaglio ecco le caratteristiche che deve possedere il prototipo che ci accingiamo a realizzare:

- Possibilità di misurare le batterie di 4,5 Volt.
- Monitor composto da tre diodi Led diversamente colorati.
- Il Led verde si illumina se la batteria si trova in uno stato ottimale, cioè se il suo potenziale è compreso tra 3,9 Volt e 4,5 Volt.
- Il Led giallo si illumina se la batteria si trova in uno stato intermedio, cioè se il suo potenziale è compreso tra 3 Volt e 3,9 Volt.
- Il Led rosso si illumina se la batteria si trova in uno stato terminale, cioè se il suo potenziale è compreso tra 2 Volt e 3 Volt: in questo caso essa è sicuramente da gettare.
- Inoltre se la tensione della batteria si trova compresa tra 1 Volt e 2 Volt, il led rosso lampeggia velocemente indicando la condizione critica della stessa.
- Infine se la tensione della batteria supera i 4,5 Volt (evenienza possibile solo quando essa è nuova) il diodo led verde lampeggia velocemente indicando un esubero di energia.
- Nessun diodo si illumina se la batteria non è collegata o se la sua tensione è minore di 1 Volt.

Schema elettrico

Lo schema da realizzare è rappresentato in figura 5 e di seguito analizzato
Alla porta A.0 del PIC16F876 è collegata la sorgente analogica, nel nostro caso la batteria da misurare, che fornisce una tensione continua compresa tra 0 Volt e 4,7 Volt.
Alla porta C.0 è collegato il diodo led verde, attraverso la resistenza di limitazione da 330

Varie condizioni della tensione della batteria da 4,5 Volt in esame (Volt)	Corrispondente tensione espressa in un valore a 10 bit	Segnalazione del monitor
≥ 0 e < 1	≥ 0 e < 205	I led restano spenti
≥ 1 e < 2	≥ 205 e < 409	Il led rosso lampeggia velocemente
≥ 2 e < 3	≥ 409 e < 614	Il led rosso è acceso
≥ 3 e < 3,9	≥ 614 e < 798	Il led giallo è acceso
≥ 3,9 e ≤ 4,5	≥ 798 e ≤ 921	Il led verde è acceso
> 4,5	> 921	Il led verde lampeggia velocemente

Tabella 2 *Possibili misurazioni di una batteria di 4,5 Volt (V_{ref} = 5 Volt)*

ohm. Alla porta C.1 è collegato il diodo led giallo e alla porta C.2 è collegato il diodo led rosso. Utilizziamo tale configurazione hardware per predisporre correttamente il listato Basic.

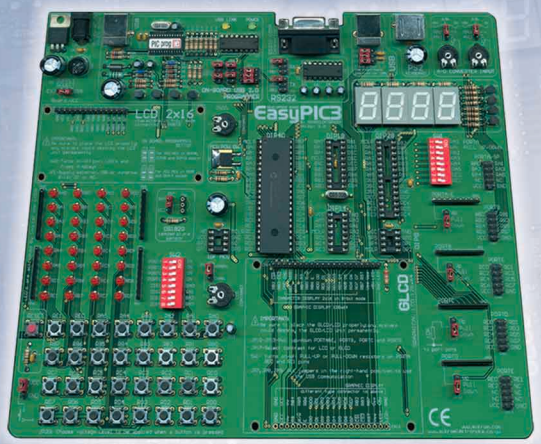
Analisi del software

Quando una batteria da 4,5 Volt si collega all'ingresso del monitor il microcontrollore deve valutare la differenza di potenziale presente tra i suoi capi. Grazie ad un opportuno software scritto allo scopo si produrrà un risultato dipendente proprio da questa tensione.

Vediamo pertanto di tracciare una sorta di elenco molto generale per dettare i passi più importanti da seguire:

- Inizializzazione delle porte e delle funzioni A/D.
- Lettura e memorizzazione della tensione esterna.
- Valutazione della stessa.
- Azione da intraprendere in conseguenza.

Scheda easyPIC3



La rivoluzionaria scheda di sviluppo per PICmicro

- Programmatore USB2.0 on-board
- Tastiera a 32 tasti
- 32 LED per il monitoraggio degli I/O
- 4 cifre LED a 7 segmenti
- Predisposizione per moduli LCD alfanumerici
- Predisposizione per moduli LCD grafici
- Predisposizione per comunicazione RS232
- Predisposizione per tastiera PS2
- Predisposizione per sensore di temperatura DS1820
- Supporto per tutte le famiglie PIC (anche PIC10F)*
- Predisposizione per comunicazione USB
- Alimentazione esterna o via USB
- Fornita con 16F877
- Disponibile con o senza display

Ordinala subito su www.farelettronica.com oppure telefona allo 02.66504794

[Listato 1]

```
program AD

rem Monitor di controllo per
rem batteria da 4,5 Volt
rem Le possibili condizioni e risposte
rem sono le seguenti:
rem Se volt >= 0 e < 1      i led restano spenti
rem Se volt >= 1 e < 2      il led rosso lampeggia veloce
rem Se volt >= 2 e < 3      il led rosso è acceso
rem Se volt >= 3 e < 3.9 il led giallo è acceso
rem Se volt >= 3.9 e <= 4.5 il led verde è acceso
rem Se volt > 4.5    il led verde lampeggia velocemente

dim volt as word 'Variabile che contiene i Volt letti

ADCON1=$80     'Configura la PORTA come input analogico
TRISA=%111111 'Tutti i bit della PORT-A sono in input
TRISC=0        'Tutti i bit della PORT-C sono in output
PORTC=0        'Azzera la PORT-C

while true         'Ciclo infinito
   volt=adc_read(0)  'LEGGE IL VALORE ANALOGICO DA RA.0
   delay_ms(10)    'Piccola pausa per non stressare l'ADC

   rem ------------Prima condizione-----------
   if (volt>=0) and (volt<205) then
      PORTC=0      'Spegne tutti i led
   end if

   rem ------------Seconda condizione-----------
   if (volt>=205) and (volt<409) then
      PORTC=0           'Spegne eventuali led accesi
      PORTC.2=1         'Accende led rosso
      delay_ms(100)     'Aspetta 1 decimo di secondo
      PORTC.2=0         'Spegne led rosso
      delay_ms(100)     'Aspetta 1 decimo di secondo
   end if

   rem ------------Terza condizione-----------
   if (volt>=409) and (volt<614) then
      PORTC=0           'Spegne eventuali led accesi
      PORTC.2=1         'Accende led rosso
   end if

   rem ------------Quarta condizione-----------
   if (volt>=614) and (volt<798) then
      PORTC=0           'Spegne eventuali led accesi
      PORTC.1=1         'Accende led giallo
   end if
```

```
    rem ------------Quinta condizione-----------
    if (volt>=798) and (volt<=921) then
       PORTC=0           'Spegne eventuali led accesi
       PORTC.0=1         'Accende led verde
    end if

    rem ------------Sesta condizione-----------
    if (volt>921) then
       PORTC=0           'Spegne eventuali led accesi
       PORTC.0=1         'Accende led verde
       delay_ms(100)     'Aspetta 1 decimo di secondo
       PORTC.0=0         'Spegne led verde
       delay_ms(100)     'Aspetta 1 decimo di secondo
    end if

wend

end.
```

Riassumiamo in tabella 2 le possibili situazioni che si possono verificare durante la misura di una batteria e la conseguente risposta del circuito da parte del microcontrollore.
Invitiamo il lettore a leggere e comprendere bene questa tabella, ponendo particolare attenzione agli **operatori di confronto**.
Tutto il programma itera attorno alla lettura esterna del valore della tensione (tramite conversione A/D) e relativa decisione di comportamento. L'intero algoritmo si trova all'interno di un ciclo infinito, permettendo così la ripetizione continua degli eventi e dei test ottenendo una risposta stabile e in tempo reale.

Il programma Basic

Il listato 1, riccamente commentato, risulta semplice ma nello stesso tempo molto didattico. L'intero ciclo nel quale è contenuto il programma si ripete all'infinito, per cui una variazione dello stato della batteria sottoposta al test è subito rilevata dal circuito.
Il suo funzionamento è quindi in tempo reale. Particolare importanza riveste il registro **ADCON1** che, contenendo il valore proposto nel listato, consente di configurare la PORT-A come digitalizzatore analogico.

Commenti all'utilizzo

Dal momento che una batteria si scarica dopo molti giorni si può simularla, al fine di velocizzare il processo di testing, connettendo direttamente al pin RA.0 il nodo centrale di un potenziometro da 10K utilizzato come partitore resistivo (e collegando gli altri due terminali rispettivamente a massa e Vcc) oppure usando un semplice alimentatore regolabile con continuità. In questa maniera si notano subito le diverse reazioni del circuito alle diverse tensioni della sorgente da analizzare.
Inoltre si ricorda che per poter valutare correttamente lo stato di una batteria è opportuno collegarla ad un carico fittizio poiché l'indicazione a vuoto (senza carico) non fornisce mai un riferimento utile.

Note per la programmazione

Nel creare il progetto, e di conseguenza il listato basic, il lettore non dimentichi di selezionare il *device* PIC16F876 nell'ambiente del Mikrobasic e naturalmente nel programmatore collegato al computer (figura 6).

SECONDA ESPERIENZA: TERMOMETRO SEMPLICE (VERSIONE BASE)

Cimentiamoci adesso nella realizzazione di un prototipo di un preciso termometro da utilizzare per misurare la temperatura ambiente. Quale elemento sensore alla temperatura si utilizza l'integrato LM 335.
Non è prevista, in questa versione, la visualizzazione delle misure rilevate poiché ci preme maggiormente l'aspetto pratico e didattico. Il

termometro in questione deve eseguire i seguenti compiti:

1. Rilevare la temperatura ogni ora.
2. Memorizzare la temperatura (o meglio il suo dato digitale corrispondente) in Eeprom.
3. Ad ogni memorizzazione deve lampeggiare il diodo led per 10 volte, molto velocemente.
4. Aspettare per un ora (ciclo di attesa).
5. Ripetere il ciclo per 24 ore.
6. Infine si deve illuminare un led (inizialmente spento) indicando il termine delle operazioni di rilevazione.

L'operatore umano, da parte sua, alla fine delle misurazioni (quindi dopo 24 ore) deve eseguire i seguenti passi, per ottenere il prospetto delle temperature giornaliere:

1. Leggere manualmente le 24 celle della Eeprom che contengono le 24 temperature, con un apposito hardware e programma.
2. Copiare questi valori su un foglio elettronico.
3. Convertire i valori digitali ottenuti in una corrispondente notazione analogica adatta a rappresentare correttamente le temperature.
4. Tracciare un grafico per fornire una valutazione visuale dell'intero processo.

Questa prima versione del termometro è puramente didattica, infatti le varie operazioni di decodifica e di calcolo sono alquanto macchinose, ma contribuiscono a fornire un'esatta visione d'insieme su tutto ciò che accade all'interno del microcontrollore.

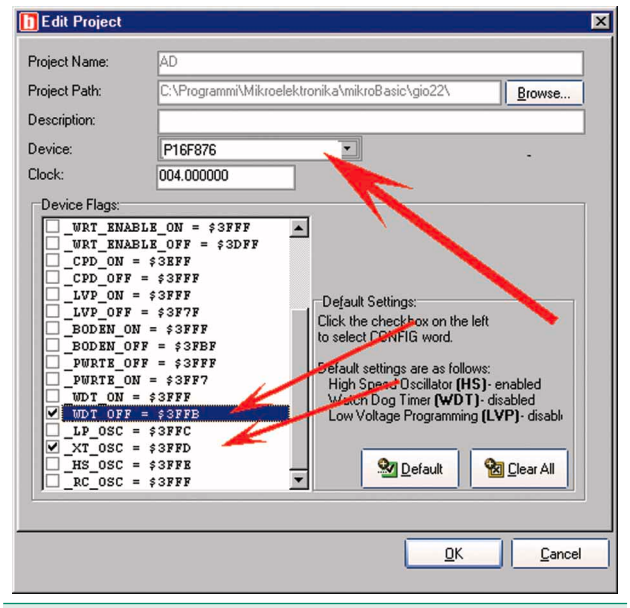


Figura 6 Le specifiche del progetto

Schema elettrico

Lo schema da realizzare è rappresentato in figura 8. Alla porta A.0 del PIC16F876 è collegata l'uscita del termometro (analogico).
Nel prossimo paragrafo viene approfondito l'utilizzo dello stesso.
Alla porta C.0 è collegato il diodo led verde, attraverso la resistenza di limitazione da 330 ohm., che ha il compito di visualizzare le operazioni di rilevamento in corso e di fine lavoro.
Il termometro è collegato a massa tramite il suo dell'anodo e a Vcc (5 Volt) tramite il suo catodo, attraverso la resistenza di limitazione da 2,2 K che assicura anche l'esatto flusso di corrente (circa 1mA) per un corretto funzionamento.

Il termometro LM335

Si presenta come un comune transistor al silicio, munito di tre piedini. Si tratta invece di un complesso circuito

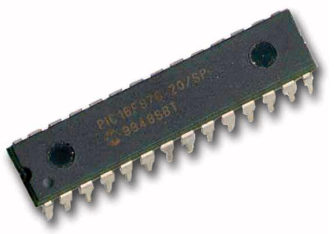


Figura 7 Il PIC16F876

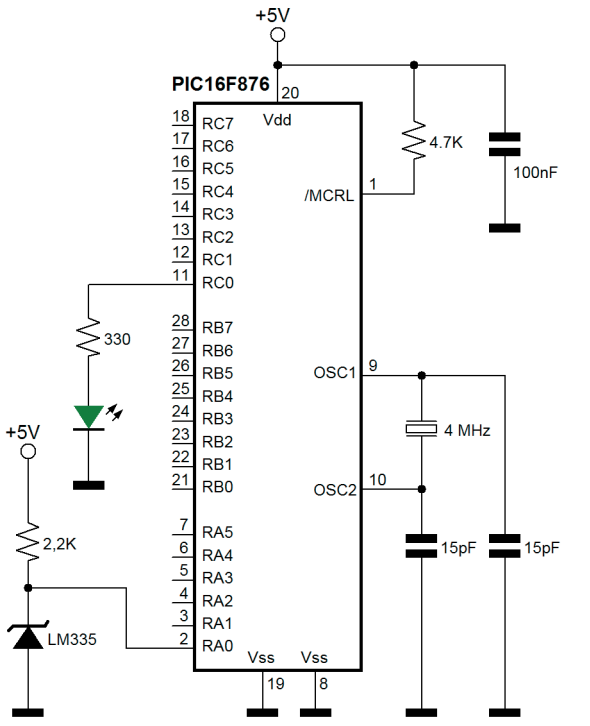


Figura 8 Schema elettrico del termometro didattico

integrato contenente un preciso termometro analogico calibrato e regolabile esternamente. Dal costo estremamente contenuto può essere utilizzato in qualsiasi sistema analogico ma anche essere adattato ad un prototipo digitale, con una difficoltà maggiore (come nel nostro caso).
Viene utilizzato come un diodo zener e la sua precisione è molto elevata, infatti assicura un margine di errore al di sotto dell'1%. Ma soprattutto l'andamento della misura è costantemente **lineare**.
Il suo unico neo (se di neo si può parlare) è quella di fornire le misura considerando la scala termometrica *Kelvin* e non *Celsius*, ma con alcune operazioni aritmetiche questo problema viene brillantemente risolto.
Il suo funzionamento tipico è il seguente: se tarato correttamente (ma solitamente non ha bisogno di questa operazione) la sua uscita fornisce una tensione di 2,98 V ad una temperatura di 25°C.
La sua variazione ammonta a 10 mV per ogni grado Kelvin di variazione. Per maggiori dettagli, e soprattutto per la sua taratura, si consulti il data-sheet fornito dalla casa produttrice che riporta anche alcuni schemi applicativi.

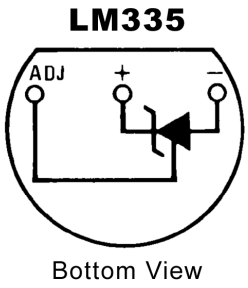


Figura 9 *Connessioni LM335 (viste da sotto)*

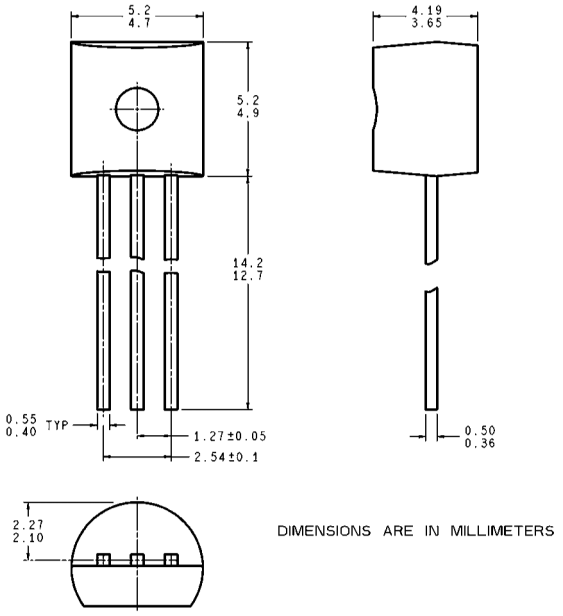


Figura 10 *Dimensioni fisiche LM335*

Gradi Kelvin e gradi Celsius

Fortunatamente, per convertire una misura da gradi Kelvin a Celsius, o viceversa, non occorre eseguire alcuna proporzione.
La scala Kelvin è solamente teorica in quanto non è utilizzata per misurare fenomeni metereologici ma solo fenomeni fisici (infatti 0 gradi Kelvin, ovvero lo zero assoluto, corrisponde a –273,15 gradi Celsius).
Per convertire le misure da un sistema ad un altro occorre semplicemente eseguire le seguenti operazioni:

1. Gradi Celsius = Gradi Kelvin – 273
2. Gradi Kelvin = Gradi Celsius + 273

Dal momento che il sensore possiede una variazione interna del potenziale equivalente a 10mV per grado (centigrado o Kelvin è la stessa cosa) ma lo stesso è tarato per la scala Kelvin, occorre sottrarre dal suo potenziale di risposta il valore 2,73 per ottenere la misura espressa in gradi centigradi.
Con un esempio chiariamo meglio il concetto. Si supponga di misurare, attraverso un tester, il potenziale di **3,12** Volt dall'uscita del sensore. Si comprenda come i decimali siano molto importanti ai fini della rilevazione per cui non si devono eliminare. Questo differenza di potenziale equivale esattamente a 312 gradi Kelvin (infatti 3,12 : 0,01 dà esattamente 312).
Per ottenere la misura in gradi centigradi basta eseguire la sottrazione 312-273 per ottenere il risultato più utile in Celsius (ossia 39° C).

Diagramma di flusso

In figura 11 è raffigurato un diagramma di flusso che facilita la comprensione delle varie fasi eseguite dal microcontrollore mediante il programma scritto in Basic.

Il programma

Quanto detto sino ad ora fornisce le basi per la costruzione di un rudimentale ma preciso termometro digitale. Da notare che il software non effettua nessun tipo di conversione termometrica ma salva su Eeprom il dato grezzo, così come lo riceve dal sensore. Sarà cura poi dell'operato-

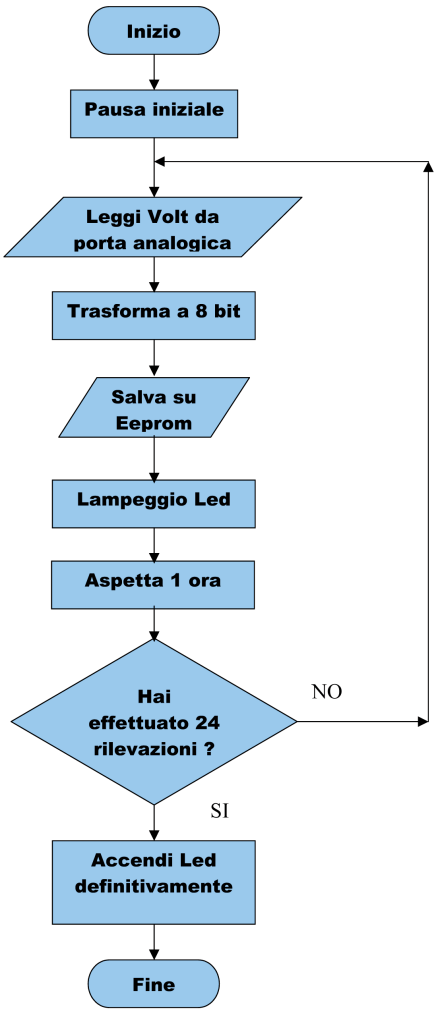


Figura 11 Diagramma di flusso del progetto termometro

re eseguire i calcoli per la conversione attraverso un foglio elettronico.
Il listato 2 riporta il software da trascrivere nell'editor del compilatore. Studiatelo molto attentamente per comprendere la funzione di tutte le variabili coinvolte e delle varie routine utilizzate. Particolare attenzione meritano due blocchi del programma: la parte che si occupa della rilevazione della temperatura e quella della memorizzazione su Eeprom.
Si esamini la prima: dopo la lettura dell'ingresso analogico, il valore (campionato a 10 bit) è memorizzato nella variabile volt. Occorre però sottolineare che questo dato è troppo grande per essere ospitato nella Eeprom (che accetta invece solamente valori di 8 bit). Occorre pertanto adeguare la misura a tale grandezza semplicemente dividendo il numero per quattro. In questa maniera il dato rilevato avrà un valore massimo di 255 contro i 1023 iniziali ma in questo modo può essere correttamente memorizzato sulla Eeprom. Si effettua in altre parole uno shift a destra, ossia uno slittamento di bit verso destra di due posizioni con la conseguenza di dividere per quattro l'informazione. Unico neo di questa operazione è una perdita di risoluzione del 75%.
La seconda parte invece si occupa della memorizzazione permanente sulla memoria interna del PIC, ossia la sua Eeprom. Il dato è memorizzato partendo dalla prima cella (indirizzata con zero) sino all'ultima (indirizzata con 23). Si spiega pertanto la costruzione della variabile *indirizzo* attraverso la clausola *ciclo-1*, proprio per non lasciare vuota la prima locazione.
Infine l'attesa di un'ora (per scandire il tempo intercorrente tra le due misurazioni) è affidata ad un ciclo che esegue per 3600 volte la pausa di un secondo (equivalente appunto a un'ora). Questo perché, allo stato attuale, il compilatore non accetta funzioni del tipo:
delay_ms(3600000)

Lettura e rappresentazione delle temperature

Bene, trascorse 24 ore dall'esecuzione del programma su PIC, sono stati registrati 24 valori digitali, ognuno dei quali "depositato" nella propria cella e rappresentante il valore letto della tensione del sensore.
Con un software di lettura/scrittura MCU si effettui la lettura della memoria, come rappresentato in figura 12.
Come si può notare, le 24 temperature sono state correttamente registrate nella zona gialla contrassegnata dal cerchio rosso.
Ma quale è la procedura (manuale) per convertire tali valori in corretta rappresentazione di gradi centigradi? Seguiamo questo iter di calcoli, facendoci aiutare eventualmente da un foglio elettronico ed esaminando il primo valore nell'indirizzo 0 della Eeprom (8D).
Questo valore rappresenta, come detto prima, la tensione misurata all'ingresso analogico e depositato qui senza alcuna conversione, se non quella analogico/digitale. Pertanto esso ha una risoluzione di 8 bit.
Si eseguano i seguenti passi:

1. Convertire il valore esadecimale letto (**8D**) in equivalente notazione decimale: $8D_{16}$ = **141**$_{10}$

2. Adeguare tale valore al range 0-5 con una proporzione: 141 x 5 : 255 = **2,76** (questa è la reale tensione misurata).
3. Moltiplicare tale tensione per 100 per convertirla in gradi Kelvin: 2,76 x 100 = **276**° K.
4. Sottrarre da tale valore 273 per ottenere il corrispondente grado Celsius: 275 – 273 = **3**° C.

Sembra una procedura complicata ma automatizzandola con opportuni software di calcolo si eviteranno perdite di tempo ed eventuali errori. La procedura sopra elencata serve anche a determinare i gradi sotto lo zero.
Guardiamo la tabella 3 dove sono riportati i calcoli effettuati.

[Listato 2]

```
rem Semplice termometro

dim volt as word  'Variabile che contiene i Volt letti
dim ciclo as byte  'Conta numero di rilevazioni
dim indirizzo as byte  'Contiene indirizzo Eeprom
dim k as word       'Variabile di comodo

ADCON1=$80     'Configura la PORTA come input analogico
TRISA=%111111 'Tutti i bit della PORT-A sono in input
TRISC=0        'Tutti i bit della PORT-C sono in output
PORTC=0        'Azzera la PORT-C

delay_ms(1000)       'Pausa iniziale 1 secondo
for ciclo=1 to 24    'Esegue per 24 volte

   rem ---------Lettura da ingresso analogico------
   volt=adc_read(0) 'LEGGE IL VALORE ANALOGICO DA RA.0
   volt=volt/4      'Trasforma da 10 bit a 8 bit (byte)

   rem ---Memorizza il dato acquisito su Eeprom---
   indirizzo=ciclo-1
   Eeprom_Write(indirizzo,volt)

   rem ---Il led Lampeggia 10 volte velocemente---
   for k=1 to 10
      PORTC.0=1      'Accende il led
      delay_ms(250) 'Piccola pausa
      PORTC.0=0      'Spegne il led
      delay_ms(250) 'Piccola pausa
   next k

   rem -----Attesa di un ora (3600 secondi)
   for k=1 to 3600
      delay_ms(1000)    'Pausa di attesa
   next k

next ciclo

PORTC.0=1     'Accende definitivamente il LED

end.
```

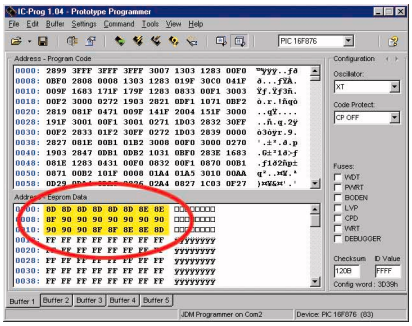


Figura 12 *La lettura della Eeprom del PIC*



Figura 13 *Protezione del sensore LM335*

Note

L'unica nota dolente, utilizzando questo metodo, è la perdita di risoluzione del digitalizzatore, infatti si è considerata l'informazione a 8 bit e non più a 10 bit, per adeguare la misura alla capacità della Eeprom.
Tuttavia esistono metodi matematici più complessi per evitare tale problema.
Si consiglia di inserire il sensore di temperatura in una idonea copertura attraverso una guaina in gomma termo-restringente, per aumentarne la protezione contro agenti atmosferici (figura 13).

COMPENDIO E CONSIGLI

Bene, ci sarebbero ancora tantissime cose da dire, ma lo spazio purtroppo è tiranno. In ogni caso il materiale presentato è sufficiente per iniziare a sperimentare con le conversioni analogico-digitali.
Elenchiamo pertanto una serie di punti su cui il lettore dovrebbe meditare e soffermarsi durante lo sviluppo delle sue applicazioni:

1. Il PIC16F876 fornisce un'accurata rappresentazione della tensione in ingresso a 10 Bit; ciò vuol dire che la variabile utilizzata contiene un valore disponibile su un range da 0 a

Indirizzo Eeprom	Valore esadecimale letto dalla Eeprom	Equivalente notazione decimale	Adeguamento al range 0-5V	x 100 (gradi Kelvin)	K - 273 (gradi Celsius)
0	8D	141	2,76	276	3
1	8D	141	2,76	276	3
2	8D	141	2,76	276	3
3	8D	141	2,76	276	3
4	8D	141	2,76	276	3
5	8D	141	2,76	276	3
6	8E	142	2,78	278	5
7	8E	142	2,78	278	5
8	8F	143	2,80	280	7
9	90	144	2,82	282	9
10	90	144	2,82	282	9
11	90	144	2,82	282	9
12	90	144	2,82	282	9
13	90	144	2,82	282	9
14	90	144	2,82	282	9
15	90	144	2,82	282	9
16	90	144	2,82	282	9
17	90	144	2,82	282	9
18	90	144	2,82	282	9
19	8F	143	2,80	280	7
20	8F	143	2,80	280	7
21	8E	142	2,78	278	5
22	8E	142	2,78	278	5
23	8D	141	2,76	276	3

Tabella 3 *Procedura di conversione da segnale LM355 a gradi Centigradi*

1023, equivalente ad una tensione variabile da 0 a 5 Volt. Ci si riferisca pertanto a questo valore per decisioni logiche ed elaborazioni in genere.

2. Nel trattare un grandezza analogica, ad esempio la temperatura, conviene rilevare più valori in più unità di tempo ed effettuarne la media aritmetica. Ciò contribuisce ad aumentare la precisione della lettura.
3. Se si deve campionare un segnale con tensione superiore a 5 Volt, lo stesso deve essere adeguato al range sopportato dal PIC attraverso un partitore resistivo variabile di precisione.
4. È possibile campionare più segnali contemporaneamente dalla PORT.A anche se ciò comporta una maggiore difficoltà di programmazione.

CONCLUSIONI

In passato l'operazione di campionamento era una delle più complicate. Con l'avvento dei circuiti integrati, e quindi dei comparatori, tale compito è diventato sempre più semplice.
Oggi, con la comparsa dei microcontrollori, ma ancor di più, dei compilatori che rendono semplice la vita del programmatore, la conversione di un segnale da analogico a digitale è proprio un gioco da ragazzi. Basta solamente organizzare e pianificare il lavoro di analisi.
Buona sperimentazione a tutti.

Codice MIP 249048

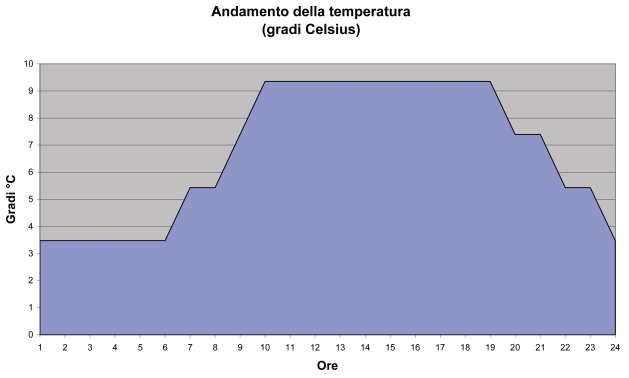
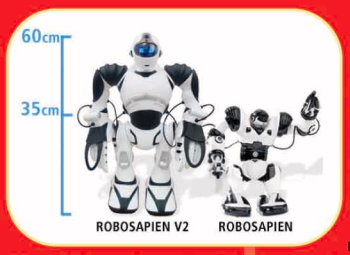


Figura 14 *Il grafico dell'andamento della temperatura*

IDEA ELETTRONICA: accendiamo le tue idee!

ROBOSAPIEN 2



Il giocattolo si evolve parla, canta, balla, gioca, raccoglie e lancia oggetti, e fa persino mosse di kung fu. Si chiama Robosapien V2, e rappresenta la nuova generazione di robot per il divertimento domestico, presentati in anteprima al CES di Las Vegas. Dotato di videocamere e radar a infrarossi, il robottino alto 60 cm è in grado anche di riconoscere le persone, interagendo con loro.

€ 299,00

MINI ROBOSAPIEN - edizione limitata -



Fratello più piccolo del Robosapien! Questo piccolo robot ha funzionalità limitate rispetto al grande Robosapien, ma è fantastico per gli appassionati di Robosapien. Misura appena 18 centimetri di altezza e può camminare. I suoi caratteristici occhi rossi si illumineranno mentre cammina. Le pinze di presa funzionano con i tasti meccanici ed i relativi bracci possono essere spostati su e giù usando gli scorrevoli sulla parte posteriore delle sue spalle. Il mini Robosapien utilizza due batterie AAA.

BIANCO € 20,00 - ARGENTO/BLU/ORO € 23,00



ROBOSAPIEN COLORATI

- edizione limitata -

Robotsapien è un mini Robot Androide progettato da Mark Tilden, creatore dei Robots B.e.a.m. e degli stupefacenti B.I.O. Bugs. Robosapien compie Movimenti e gesti fluidi: camminata veloce e dinamica a due velocità; braccia completamente funzionali con due tipi di presa delle mani, 67 funzioni pre-programmate: presa, lancio, calcio, danza, kung-fu, aerofagia, eruttazione, rap e molte altre ancora, 4 modalità di programmazione, 3 modalità dimostrative.

€ 125,00

MINI ROBOT PICAXE



Il Microrobot Picaxe consiste in una base in plastica di 120 X 80mm che include 2 motoriduttori (rapporto 42:1) con relative ruote in gomma e ospita 4 batterie stilo, una scheda che e' composta da un PICAXE 18 (espandibile con un PICAXE18X) un chip che genera automaticamente gli impulsi PWM per comandare i motori e un chip di potenza che pilota i due motori (L293D). La funzione PWM e' molto interessante in quanto permette di modulare la velocita' dei motori in modo molto graduale, senza impegnare il PICAXE, lasciando quindi spazio al programma di gestione dei sensori e della logica del Robot. Sono disponibili ben 5 Input per collegare vari sensori e 4 Output. La scheda e', inoltre, provvista del comodo connettore da 3.5mm stereo, per la programmazione seriale, in questo modo tutte le operazioni di programmazione e debug avvengono in modo rapido e sicuro.

€ 84,00

Codice MIP 249065

Tutti i prezzi si intendono IVA compresa. Per ordini e informazioni:
IDEA ELETTRONICA - Via San Vittore n°24/A - 21040 Oggiona con S. Stefano - Varese - ITALY - Tel./Fax 0331/212723
Visitate il nostro sito: WWW.IDEAELETTRONICA.IT

Decima parte
n° 248 - Febbraio 2006
La scheda di sviluppo EasyPIC3

Undicesima parte
n° 249 - Marzo 2006
Conversionie analogico/digitale con il PIC

Dodicesima parte
n° 250 - Aprile 2006
I²C con il PIC

Mikrobasic per PICmicro

Questa puntata conclude il corso sul Mikrobasic. Affronteremo l'implementazione di una Eeprom esterna, con bus I2C. Un passo obbligato per aumentare il numero delle informazioni da memorizzare in modo permanente.

In elettronica esistono circuiti che svolgono i compiti più disparati, dai più semplici ai più complessi. Elaborazioni di vario genere che, con l'acquisizione di dati, producono risultati e soluzioni compatibili con le esigenze del programmatore.
Tutta questa grande tecnologia, fatta di sforzi e analisi anche impegnative, non avrebbe nessun valore pratico se le informazioni elaborate non venissero in qualche modo memorizzate da qualche parte, per la successiva elaborazione.
Pensiamo ad esempio alle missioni spaziali: durante le "gite" fuori dalla nostra atmosfera, i satelliti e le sonde dispongono di tantissimi sensori che "registrano" continuamente informazioni di carattere tecnico, fisico, chimico. Se i risultati non fossero raccolti in opportune memorie, l'intera missione avrebbe solo motivo di perdita di tempo e denaro.
La memorizzazione perenne dei dati si può considerare dunque il penultimo stadio di un intero sistema automatico intelligente, cui segue naturalmente quello della raccolta ed elaborazione degli stessi.

LE EEPROM ESTERNE

Le Eeprom che trattiamo in quest'ultima puntata vengono prodotte dalla Microchip e per caratteristiche fisiche sono del tutto simili a quelle presenti all'interno del PIC (figura 1).
Sono paragonabili a piccolissimi hard disk, ma senza una struttura magnetica, e la loro implementazione in un sistema a microcontrollore non comporta problemi di sorta, essendo necessarie solamente 2 linee logiche di comunicazione digitale. I dati sono mantenuti per molto tempo anche senza alimentazione elettrica.
Esse sono contrassegnate con una sigla che inizia per 24, seguita dalla capacità in Kilobit (almeno per la maggior parte di questi modelli). In tabella 1 sono riportate le caratteristiche generali delle Eeprom di questa famiglia.

Basso consumo (tipico 1 mA)
Interfaccia I²C a due sole connessioni
Input triggerato per la soppressione di rumore
1.000.000 di cicli scrittura/cancellazioni garantiti
Mantenimento della memoria senza alimentazione per più di 200 anni
Funzionamento in un range esteso di temperatura
Basso costo

Tabella 1 *Caratteristiche generali delle Eeprom modello 24Cxx*

LA EEPROM 24C16

Per i nostri esperimenti si utilizzerà la Eeprom **24C16**. Si tratta di una *Elettrically Erasable PROM* prodotta dalla Microchip Tecnology Inc. La sua capacità è di 16K bit (equivalenti a 2048 bytes). Si possono pertanto memorizzare, in modo permanente, ben 2048 dati in formato byte, superando di gran lunga la capacità della Eeprom interna del PIC.
La memoria è organizzata in 8 blocchi da 256 byte ognuno.



Figura 1 *Tipica Eeprom esterna al PIC*

I²C con il PIC



di Giovanni Di Maria

Questa suddivisione serve per indirizzare la corretta pagina corrente, come si vedrà fra poco con degli esempi.
Come tutti i modelli della sua famiglia, il "colloquio" con il sistema *Master* ad essa collegato avviene attraverso due fili soltanto, semplificandone enormemente la gestione.

Caratteristiche

In tabella 2 sono riportate le caratteristiche principali del modello 24C16. Si tratta, come si può vedere, di dati importanti che ne decretano sicuramente l'affidabilità e la robustezza. Per maggiori dettagli si consulti comunque il *datasheet* della stessa casa madre.

Basso consumo (tipico 1 mA)
Organizzazione in 8 blocchi da 256 bytes
Interfaccia I²C a due sole connessioni
Input triggerato per la soppressione di rumore
Possibilità di proteggere da scritture accidentali
Protezione contro l'elettricità statica fino a 4000 Volt
1.000.000 di cicli scrittura/cancellazioni garantiti
Mantenimento della memoria senza alimentazione per più di 200 anni
Funzionamento in un range esteso di temperatura
Disponibile anche in DIL 8 pin
Basso costo

Tabella 2 *Caratteristiche della Eeprom modello 24C16*

Funzionamento dei pin

In figura 2 è riportata la disposizione dei pin del chip. In dettaglio ecco la funzione di ciascuno di essi:

1. Il pin **A0** non è utilizzato nel modello 24C16 pertanto può essere indifferentemente collegato a massa, a Vcc o addirittura lasciato libero, in quanto risulta internamente non connesso.
2. Anche Il pin **A1** è inutilizzato nel modello 24C16 pertanto può essere indifferentemente collegato a massa, a Vcc o lasciato libero.
3. Anche Il pin **A2** è inutilizzato nel modello 24C16 pertanto può essere indifferentemente collegato a massa, a Vcc o lasciato libero.
4. Il pin **VSS** o **GND** va collegato alla massa comune del circuito (ground).
5. Il pin **SDA** (*Serial Data Address*) ha il compito di ricevere dal Master, bit dopo bit, il dato vero e proprio, ossia l'informazione o il comando da gestire.
6. Il pin **SCL** (*Serial Clock*) ha il compito di acquisire gli impulsi di clock.
7. Il pin **WP** (*Write Protect*) abilita o meno la protezione da scrittura dei dati. Se è collegato a massa, il micro può liberamente scrivere sulla Eeprom. Se è collegato a Vcc, quest'ultima è protetta da scrittura e nessun dato può essere memorizzato; in questo caso il suo utilizzo è assimilabile ad una normale Rom. Non viene invece intaccata la funzione di lettura.
8. Il pin **VCC** infine deve essere connesso sempre all'alimentazione del circuito entro un *range* consentito di +2.5V minimi contro +5.5V massimi.

Gli altri fratelli

Sono elencate, in tabella 3, le capacità di memoria degli altri modelli. Tutte le caratteristi-

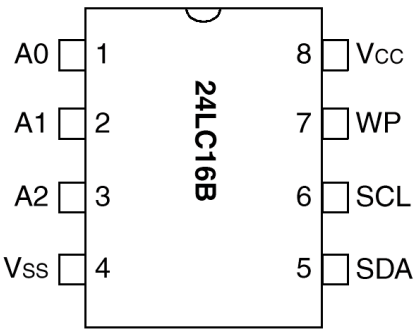


Figura 2 *Piedinatura del modello 24C16*

Modello	Capacità in bit	Capacità in bytes
24C01	1K (1024)	128 bytes
24C02	2K (2048)	256 bytes
24C04	4K (4096)	512 bytes
24C08	8K (8192)	1024 bytes
24C16	16K (16384)	2048 bytes
24C32	32K (36768)	4096 bytes
24C64	64K (65536)	8192 bytes
24C128	128K (131072)	16384 bytes
24C256	256K (262144)	32768 bytes
24C512	512K (524288)	65536 bytes

Tabella 3 *I vari modelli di Eeprom della famiglia 24Cxx*

che sono compatibili con i montaggi effettuati e per la sostituzione con altro tipo non occorre modificare la circuiteria del prototipo, ma solamente il software applicativo. Si tratta di capacità di memorizzazione sicuramente irrisorie se paragonate a quelle di un personal computer, ma per molte applicazioni embedded sono più che sufficienti.

IL PROTOCOLLO I²C

É un metodo di comunicazione dati, messo a punto dalla Philips. Il suo funzionamento si basa su un bus seriale sincrono, che permette la comunicazione tra più periferiche. Si utilizza molto per collegare due sistemi allo scopo di scambiare dati in formato seriale (generalmente si tratta di unità *Master/Slave*).
Solitamente il microcontrollore funge da *Master* mentre la Eeprom da *Slave*. Il punto di forza di questo protocollo è quello di utilizzare, come detto precedentemente, due sole connessioni: una per il clock e l'altra per i dati.
Il protocollo è supportato, oltre che dalle Eeprom, anche da molti altri dispositivi tra cui: display LCD, convertitori ADC e DAC, eccetera. I dispositivi possono comunicare tra loro ad una velocità di 100 Kbit/s o 400 Kbit/s.
Ultimamente con alcuni modelli è possibile ottenere anche un rate di 1000 Kbit/s.
La transazione inizia quando il bus è libero, cioè quando i pin SDA e SCL sono a livello logico alto. In questo caso il Master (cioè il PIC) può inviare un segnale di Start. Quindi spedisce il byte di controllo (che identifica anche lo slave e la direzione dei dati).
A questo punto si avvia la trasmissione dei dati vera e propria. Quando il trasferimento è completo, il master deve inviare un segnale di Stop e il bus diventa nuovamente libero.

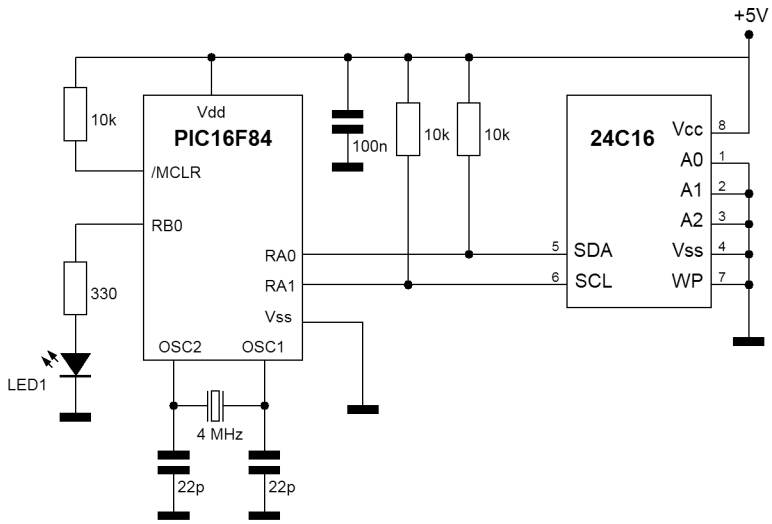


Figura 3 *Schema elettrico di configurazione Master-Slav*

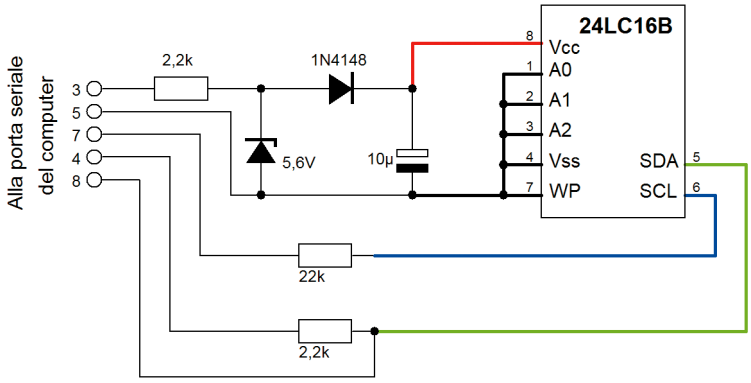


Figura 4 *Schema elettrico dell'interfaccia PC-Eeprom*

SCHEMA APPLICATIVO

Le connessioni da effettuare sono veramente semplici. La configurazione adottata è quella del *Master* e *Slave*, proposto in figura 3.
Si utilizza anche un diodo led collegato alla porta RB0 con funzione di monitor per le operazioni di lettura e scrittura. Esso, inizialmente acceso, si spegne al termine delle transazione con la Eeprom. La resistenza di limitazione ne assicura un corretto funzionamento.
Ribadiamo che il *datasheet* originale della Microchip lascia l'utente libero di collegare i pin 1, 2, 3 a massa, a Vcc ovvero di non connetterli del tutto, in quanto internamente non fanno capo a nessuna parte del circuito.

CIRCUITO PER LETTURA E SCRITTURA DI UNA EEPROM DA PC

Per permettere successivamente la lettura e la verifica dei dati occorre una particolare interfaccia che, tramite personal computer e software dedicato, visualizzi l'intero contenuto della Eeprom.
In figura 4 è riprodotto lo schema elettrico, tra l'altro di semplicissima concezione. La connessione al personal computer avviene utilizzando l'apposito connettore a 9 pin (tipo femmina) inserito nella porta COM del PC.

I COMANDI DEL MIKROBASIC

Il Mikrobasic dispone di molte funzioni e procedure per la gestione completa del protocollo I_2C. Alcune di esse prevedono la connessione della memoria esclusivamente sulla PORT.B o PORT.C. Ci occupiamo invece di quelle che prevedono il collegamento delle Eeprom su qualsiasi porta, fornendo così al programmatore più libertà di scelta.
Da sottolineare che le routine che ci accingiamo a spiegare sono completamente indipendenti dall'hardware utilizzato, pertanto possono essere utilizzate con qualunque tipo di microcontrollore.
Ecco l'elenco delle funzioni e delle procedure che offre il compilatore Mikrobasic:

- Soft_I2C_Config
- Soft_I2C_Start
- Soft_I2C_Read
- Soft_I2C_Write
- Soft_I2C_Stop

Si descrive adesso l'utilizzo e la sintassi di ciascuna di esse, approfondendole in seguito con degli esempi pratici.

Soft_I2C_Config

Si tratta di una procedura, pertanto non ritorna alcun valore alla routine chiamante. É utilizzata per configurare e specificare la porta del PIC a cui è collegata la memoria esterna in esame. Occorre invocarla sempre all'inizio delle procedure di gestione della Eeprom.
La sintassi di utilizzo è la seguente:
Soft_I2C_Config(porta, numero porta SDA, numero porta SCL)
dove:

- **porta** indica la porta fisica, passata indirettamente per indirizzo (ma del tutto trasparente al programmatore);
- **numero porta SDA** è semplicemente il numero del bit della porta su cui è collegato fisicamente il pin SDA della Eeprom (linea dei dati);
- **numero porta SCL** è semplicemente il numero del bit della porta su cui è collegato fisicamente il pin SCL della Eeprom (linea del clock).

Esempio di utilizzo:

```
Soft_I2C_Config(PORTA, 0, 1)
```

Nell'esempio si è collegata la Eeprom sulla PORT.A del PIC e in particolare la linea dati al pin RA.0 e la linea clock al pin RA.1.
É assolutamente vietato collegare le due linee della memoria al medesimo pin, per ovvi motivi logici.

Soft_I2C_Start

É una procedura che si occupa di inviare un segnale di Start alla Eeprom. Occorre invocarla prima di ricevere o inviare dati alla memoria esterna.
La sintassi di utilizzo, molto semplice, è la seguente:

```
Soft_I2C_Start
```

Soft_I2C_Read

É una funzione che ritorna il valore di un byte come risultato della lettura dalla Eeprom. Il valore di ritorno è assegnato ad una variabile di tipo byte, precedentemente dichiarata. La sintassi di

utilizzo è la seguente:

```
x=Soft_I2C_Read(ack)
```

Dove:

- **ack** è un dato di tipo byte che autorizza l'invio di un segnale di *acknowledge* se il suo valore è maggiore di zero, altrimenti lo impedisce.

Esempio di utilizzo:

```
dato = Soft_I2C_Read(0)
```

Nell'esempio il dato viene letto dalla memoria Eeprom senza l'invio di un segnale di acknowledge e il risultato è memorizzato (assegnato) alla variabile *dato*.

Soft_I2C_Write

É una funzione che ha il compito di scrivere e memorizzare un dato sulla Eeprom, ma contemporaneamente restituisce un valore indicante l'esito dell'operazione stessa.
Se la sua esecuzione va a buon fine, il byte passato come parametro viene memorizzato correttamente su memoria esterna e nel contempo ritorna il valore 0 (zero). Se invece la risposta è diversa da zero, probabilmente si è verificato un errore di scrittura e il dato non viene memorizzato (ad esempio se la Eeprom è protetta).
La sintassi di utilizzo è la seguente:

```
Soft_I2C_Write(dato)
```

Dove:

- **dato** è un parametro di tipo byte contenente il valore da passare alla Eeprom (si può anche trattare di uno pseudo comando).

Esempio di utilizzo:

```
Soft_I2C_Write($A0)
```

In quest'esempio il valore esadecimale $A0_{16}$ è inviato direttamente alla memoria esterna.

Soft_I2C_Stop

Si tratta di una semplice procedura, che invia un segnale di Stop alla memoria Eeprom.
Si utilizza per chiudere la comunicazione ed è invocata alla fine delle operazioni di connessione.
La sintassi di utilizzo è la seguente:

```
Soft_I2C_Stop
```

Altre routines

Il compilatore dispone anche di altre routine molto simili a quelle spiegate, ma il loro funzionamento avviene esclusivamente sulla PORT.B e PORT.C, pertanto ci limitiamo semplicemente ad elencarle senza approfondirle. Chi volesse maggiori dettagli in merito può consultare la guida in linea del Mikrobasic (tasto F1) ovvero il manuale digitale in formato PDF disponibile liberamente per il download sul sito di Fare Elettronica.

- I2C_Init
- I2C_Start
- I2C_Repeated_Start
- I2C_Is_Idle
- I2C_Rd
- I2C_Wr
- I2C_Stop

MEMORIZZAZIONE SU 24C16 IN PRATICA

Vediamo adesso in pratica che cosa succede al bus di comunicazione quando le informazioni devono essere memorizzate all'interno della Eeprom.
In Mikrobasic la procedura è sicuramente molto meno "indolore" della medesima creata in linguaggio Assembler. In ogni modo ecco a grandi linee le operazioni da effettuare per la corretta scrittura su una Eeprom:

- Configurazione della porta del PIC.
- Invio di un segnale di Start.
- Invio di un byte comando (controllo).
- Invio di un byte indirizzo da cui iniziare a scrivere.

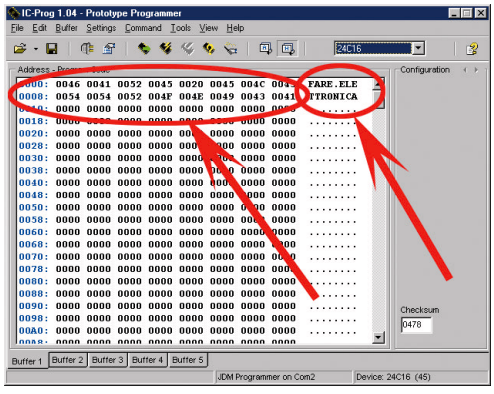


Figura 5 La mappa della Eeprom dopo l'esecuzione del programma n. 1

- Invio di un byte dato.
- Eventuale ripetizione dell'invio dei dati.
- Invio di un segnale di Stop.

Questi passi, rappresentati da altrettanti comandi in Mikrobasic, assicurano la scrittura dei dati per il successivo recupero.

Memorizziamo la nostra prima frase

Il primo programma da provare (listato 1) memorizza semplicemente la frase "FARE ELETTRONICA" all'interno della Eeprom. L'intera frase viene salvata a partire dalla locazione zero (in pratica la prima della pila) e le varie lettere sono accodate sequenzialmente fino a completamento dell'informazione.
Vengono inviati alla memoria esterna i rispettivi codici *Ascii* dei vari caratteri.
Si faccia riferimento allo schema proposto in figura 3 per la realizzazione del prototipo sperimentale.
Dopo aver compilato e scaricato il programma su PIC, si esegua lo stesso.
Il diodo led monitor, inizialmente illuminato per quattro secondi, si spegne dopo la memorizzazione, informando l'operatore dell'avvenuto processo.
Il software realizzato è a *ciclo finito*, ossia dopo la sua esecuzione "logica" esso si arresta e non reinizia più; non sono presenti infatti istruzioni di salto o di ripetizione di codice.
I tempi di attesa sono stati volutamente allungati per fornire una indicazione chiara su ciò che il microcontrollore esegue.
In effetti lo spegnimento del diodo led conferma l'avvenuto salvataggio dei dati, ma per verificare praticamente quanto accaduto si deve leggere il contenuto della Eeprom con un apposito programmatore e l'interfaccia sopra descritta. La figura 5 mostra la schermata della mappa di memoria con i caratteri memorizzati a partire dalla prima locazione.

Il programma basic è ampiamente commentato [listato 1] e ogni statement descrive in dettaglio l'operazioni eseguita. Da notare che il medesimo programma funziona perfettamente anche con gli altri modelli "più piccoli" di Eeprom, vale a dire 24C01, 24C02, 24C04 e 24C08, senza la necessità di apportare modifiche al software.

[Listato 1]

```
program i2c01

rem         LISTATO 1
rem     Questo programma memorizza la
rem     frase FARE ELETTRONICA sulla
rem     Eeprom fornendo anche indicazione
rem     luminosa su diodo Led

TRISB=0       'Definisce la porta B come uscita
PORTB=0       'Azzera tutta la porta B
PORTB.0=1     'Accende il Led connesso su RB.0

delay_ms(2000) 'Pausa per dare tempo all'operatore

Soft_I2C_Config(PORTA, 0, 1)       'Configura Eeprom
                                   'su PORT.A
                                   'SDA su RA0 e SCL su RA1

Soft_I2C_Start              'Invia segnale di START via I2C

Soft_I2C_Write($A0)         'Invia un comando al 24C16
```

```
Soft_I2C_Write(0)       'Invia indirizzo di memorizzazione

Soft_I2C_Write(70)      'Invia lettera F
Soft_I2C_Write(65)      'Invia lettera A
Soft_I2C_Write(82)      'Invia lettera R
Soft_I2C_Write(69)      'Invia lettera E
Soft_I2C_Write(32)      'Invia spazio
Soft_I2C_Write(69)      'Invia lettera E
Soft_I2C_Write(76)      'Invia lettera L
Soft_I2C_Write(69)      'Invia lettera E
Soft_I2C_Write(84)      'Invia lettera T
Soft_I2C_Write(84)      'Invia lettera T
Soft_I2C_Write(82)      'Invia lettera R
Soft_I2C_Write(79)      'Invia lettera O
Soft_I2C_Write(78)      'Invia lettera N
Soft_I2C_Write(73)      'Invia lettera I
Soft_I2C_Write(67)      'Invia lettera C
Soft_I2C_Write(65)      'Invia lettera A

Soft_I2C_Stop           'Invia segnale di STOP via I2C

delay_ms(2000)          'Pausa finale

PORTB.0=0               'Spegne il diodo Led
                        'Il programma termina qui
                        'e l'esecuzione si ferma

end.
```

Commenti al listato

Alla configurazione della porta a cui è collegata la Eeprom e l'invio del segnale di Start segue lo statement:

```
Soft_I2C_Write($A0) 'Invia un comando al 24C16
```

In effetti anche le successive linee di programma contengono l'invio di un byte. Ma questa "prima spedizione" è del tutto speciale perché informa la memoria esterna delle nostre intenzioni. Cosa indica il valore esadecimale tra parentesi ($A0)? Osserviamolo in dettaglio, convertendolo nella notazione binaria corrispondente: $A0_{16} = 10100000_2$

Questo primo byte inviato è chiamato **byte di controllo** e descrive le successive operazioni da compiere verso la memoria. É formato da più elementi (gruppi di bit) che andiamo ad esaminare, spezzandolo in tre parti fondamentali: `1010  000  0`

- I primi quattro bit (1010) determinano il codice identificativo della Eeprom 24C16.
- I seguenti tre bit (000) definiscono ed indirizzano la pagina di memoria (ampia 256 bytes) tra le otto disponibili nel device (infatti con 3 bit si possono rappresentare 8 combinazioni possibili: 000, 001, 010, 011, 100, 101, 110, 111).
- L'ultimo bit (0) indica un'operazione di lettura o di scrittura. Nella fattispecie, un valore di 0

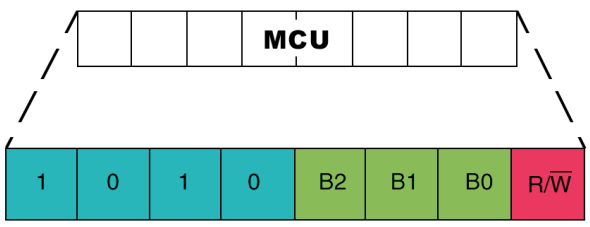


Figura 6 Significato dei singoli bit nel byte di controllo

permette la scrittura sulla Eeprom, mentre un valore di 1 ne permette la lettura.

In figura 6 è rappresentata la disposizione schematica dei vari bit all'interno del byte di controllo mentre in tabella 4 è riportato il byte di controllo nei vari modelli di memoria esterna.

Modello	Byte di controllo
24C01	1010xxx0
24C02	1010xxx0
24C04	1010xxb0
24C08	1010xbb0
24C16	1010bbb0
bbb = pagina di memoria	
xxx = non ha effetto	

Tabella 4 *Byte di controllo nei vari modelli di Eeprom*

Dopo l'invio del byte di controllo ne segue immediatamente un altro; esso indica semplicemente la locazione di memoria dalla quale far partire la memorizzazione dei dati successivi. Nel nostro caso lo statement:

```
Soft_I2C_Write(0)   'Invia indirizzo di
memorizzazione
```

indica al compilatore che i dati saranno memorizzati a partire dalla locazione zero (prima cella della pagina prescelta con 000) e, a seguire, in tutte le successive locazioni, sequenzialmente.

Le successive scritture dei bytes confermano proprio tale concetto, andando a memorizzare, uno dopo l'altro, i caratteri che compongono la frase "FARE ELETTRONICA".
La procedura si conclude con l'invio del segnale di Stop, che conclude definitivamente la comunicazione con la Eeprom esterna. La figura 7 illustra il flusso di trasmissione all'interno del bus I²C.

Scrivere su indirizzi diversi

Vediamo adesso quali operazioni si devono eseguire se occorre salvare i dati in indirizzi non adiacenti.
Il programma seguente memorizza tre parole in celle distanti tra loro e in particolare:

- La parola "FARE" a partire dall'indirizzo 8.
- La parola "ELETTRO" a partire dall'indirizzo 24.
- La parola "NICA" a partire dall'indirizzo 40.

Il listato è molto commentato [listato 2] e non ha bisogno di ulteriori spiegazioni. Occorre dire solamente che è obbligatorio inserire, dopo l'invio del segnale di Stop, una piccola pausa di almeno 10 mS, per dare alla Eeprom il tempo di memorizzare i dati. Diversamente, i successivi accessi verrebbero ignorati.

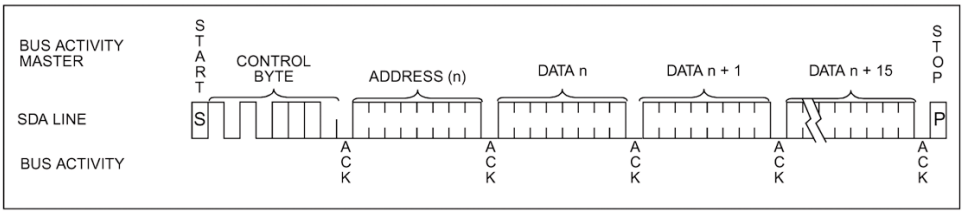


Figura 7 *Il flusso dati per la scrittura nel bus I²C*

[Listato 2]

```
program i2c02

rem         LISTATO 2
rem     Questo programma memorizza la
rem     parola              'FARE'
rem     a partire dalla cella n. 8 in poi,
rem     la parola          'ELETTRO'
rem     a partire dalla cella n. 24 in poi
```

```
rem    e la parola         'NICA'
rem    a partire dalla cella n. 40 in poi
rem    fornendo anche indicazione
rem    luminosa su diodo Led

TRISB=0       'Definisce la porta B come uscita
PORTB=0       'Azzera tutta la porta B
PORTB.0=1     'Accende il Led connesso su RB.0

delay_ms(2000) 'Pausa per dare tempo all'operatore

Soft_I2C_Config(PORTA, 0, 1)       'Configura Eeprom
                                   'su PORT.A
                                   'SDA su RA0 e SCL su RA1

Soft_I2C_Start       'Invia segnale di START via I2C
Soft_I2C_Write($A0)  'Invia un comando al 24C16
Soft_I2C_Write(8)    'Invia indirizzo 8
Soft_I2C_Write(70)   'Invia lettera F
Soft_I2C_Write(65)   'Invia lettera A
Soft_I2C_Write(82)   'Invia lettera R
Soft_I2C_Write(69)   'Invia lettera E
Soft_I2C_Stop        'Invia segnale di STOP via I2C

delay_ms(100)  'PICcola Pausa

Soft_I2C_Start       'Invia segnale di START via I2C
Soft_I2C_Write($A0)  'Invia un comando al 24C16
Soft_I2C_Write(24)   'Invia indirizzo 24
Soft_I2C_Write(69)   'Invia lettera E
Soft_I2C_Write(76)   'Invia lettera L
Soft_I2C_Write(69)   'Invia lettera E
Soft_I2C_Write(84)   'Invia lettera T
Soft_I2C_Write(84)   'Invia lettera T
Soft_I2C_Write(82)   'Invia lettera R
Soft_I2C_Write(79)   'Invia lettera O
Soft_I2C_Stop        'Invia segnale di STOP via I2C

delay_ms(100)  'PICcola Pausa

Soft_I2C_Start       'Invia segnale di START via I2C
Soft_I2C_Write($A0)  'Invia un comando al 24C16
Soft_I2C_Write(40)   'Invia indirizzo 40
Soft_I2C_Write(78)   'Invia lettera N
Soft_I2C_Write(73)   'Invia lettera I
Soft_I2C_Write(67)   'Invia lettera C
Soft_I2C_Write(65)   'Invia lettera A
Soft_I2C_Stop        'Invia segnale di STOP via I2C

delay_ms(2000)  'Pausa finale
```

```
PORTB.0=0          'Spegne il diodo Led
                   'Il programma termina qui
                   'e l'esecuzione si ferma

end.
```

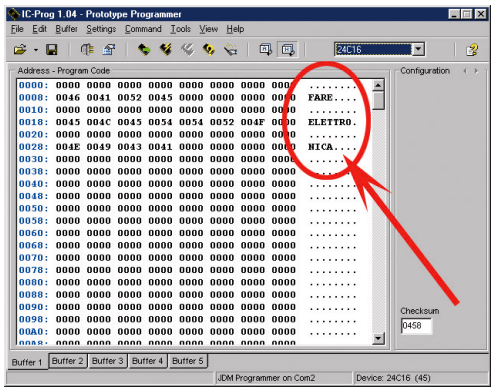


Figura 8 *La mappa della Eeprom dopo l'esecuzione del programma n. 2*

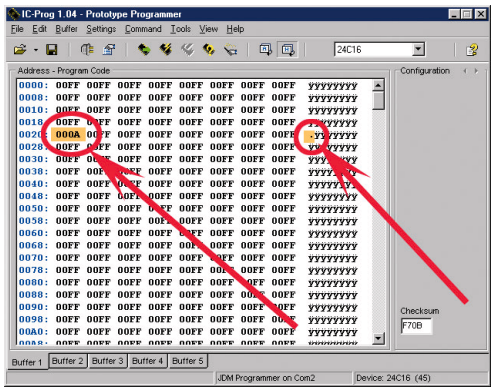
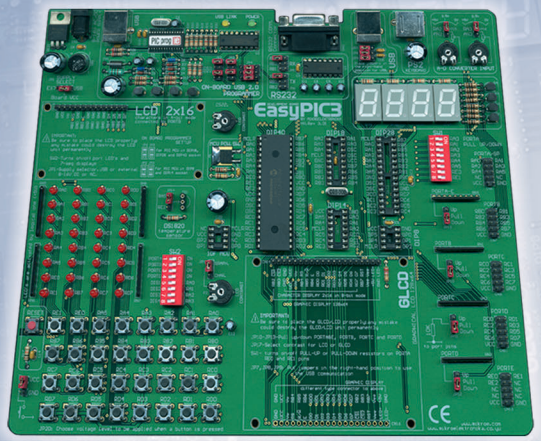


Figura 9 *Il numero dei lampeggi memorizzato nella locazione 32 della Eeprom*

Scheda easyPIC3



La rivoluzionaria scheda di sviluppo per PICmicro

- ✔ Programmatore USB2.0 on-board
- ✔ Tastiera a 32 tasti
- ✔ 32 LED per il monitoraggio degli I/O
- ✔ 4 cifre LED a 7 segmenti
- ✔ Predisposizione per moduli LCD alfanumerici
- ✔ Predisposizione per moduli LCD grafici
- ✔ Predisposizione per comunicazione RS232
- ✔ Predisposizione per tastiera PS2
- ✔ Predisposizione per sensore di temperatura DS1820
- ✔ Supporto per tutte le famiglie PIC (anche PIC10F)*
- ✔ Predisposizione per comunicazione USB
- ✔ Alimentazione esterna o via USB
- ✔ Fornita con 16F877
- ✔ Disponibile con o senza display

Ordinala subito su www.farelettronica.com oppure telefona allo 02.66504755

Un'ultima nota: tra un esempio e il successivo conviene azzerare manualmente il contenuto della Eeprom per evitare che i risultati dei vari programmi si sovrappongano. La figura 8 mostra la mappa della memoria dopo l'esecuzione del programma numero 2.

LETTURA DA 24C16 IN PRATICA

La procedura di lettura è molto simile a quella di scrittura. Occorre infatti programmare più o meno le stesse operazioni. Ecco i passi standard da seguire durante questa fase:

- Configurazione della porta del PIC (se non lo si è fatto prima).
- Invio di un segnale di Start.
- Invio di un byte di comando (controllo).
- Invio di un byte indirizzo da cui iniziare a leggere.
- Invio di un altro segnale di Start.
- Invio di un byte comando di richiesta dati.
- Lettura del byte dato.
- Ripetizione eventuale della lettura dei dati.
- Invio di un segnale di Stop.

Esempio pratico

Si supponga che nella cella 32 della Eeprom sia memorizzato un byte che indichi quanti lampeggi deve effettuare il diodo led collegato al circuito. Tale byte è stato precedentemente memorizzato tramite procedura di scrittura (vedi sopra) oppure tramite programmatore apposito. La situazione all'interno della Eeprom è pertanto quella visibile in figura 9. Si noti il byte inizializzato all'indirizzo specificato, mentre tutte le altre locazioni sono vuote (o meglio contengono altri valori). Con questo valore salvato ($0A_{16}$), il led eseguirà dieci lampeggi, prima di spegnersi definitivamente.

Il listato

Come si vede nel programma basic riprodotto nel listato 3, la gestione somiglia molto a quella del trattamento di un file in un normale programma per computer. Si ha infatti il comando di apertura, di lettura e di chiusura della comunicazione. Dopo la definizione della configurazione del circuito e la spedizione del segnale di Start il master, tramite il comando di invio ($A0), manda il byte 32 per indicare l'indirizzo iniziale della lettura. Segue pertanto un ulteriore invio del segnale di Start seguito stavolta dal byte $A1_{16}$. Si tratta, anche in questo caso, di un byte di controllo che abilita la lettura anziché la scrittura. Traducendo il valore $A1_{16}$ in 10100001_2 e dividendolo in tre parti fondamentali si ottiene: `1010 000 1`

- I primi quattro bit (1010) determinano il codice identificativo della Eeprom 24C16.
- I seguenti tre bit (000) definiscono ed indirizza-

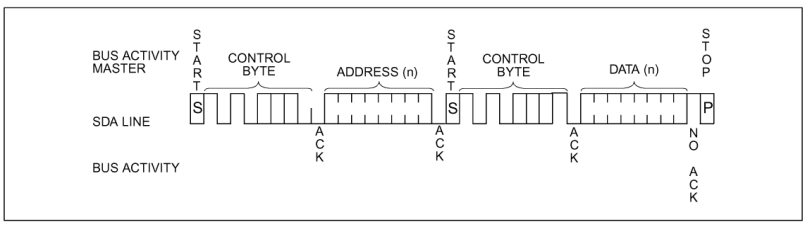


Figura 10 *Il flusso dati per la lettura dal bus I²C*

[Listato 3]

```
program i2c03

rem         LISTATO 3
rem     Questo programma legge un valore
rem     numerico dalla Eeprom.
rem     Il led lampeggia tante volte in base
rem     al valore memorizzato.

dim k as byte                'Variabile per il conteggio
dim lampeggi as byte         'Variabile che definisce il
```

```
                          'numero di lampeggi

TRISB=0    'Definisce la porta B come uscita
PORTB=0    'Azzera tutta la porta B

delay_ms(1000) 'Pausa iniziale di un secondo

Soft_I2C_Config(PORTA, 0, 1)       'Configura Eeprom
                                   'su PORT.A
                                   'SDA su RA0 e SCL su RA1

Soft_I2C_Start              'Invia segnale di START via I2C
Soft_I2C_Write($A0)         'Invia comando scrittura al 24C16
Soft_I2C_Write(32)          'Invia indirizzo 32

Soft_I2C_Start              'Invia segnale di START via I2C
Soft_I2C_Write($A1)         'Invia comando lettura al 24C16
lampeggi=Soft_I2C_Read(0)   'Legge numero lampeggi da Eeprom

Soft_I2C_Stop               'Invia segnale di STOP via I2C
                            'e chiude la comunicazione con la Eeprom

for k=1 to lampeggi         ' Ripete ciclo n volte
   PORTB.0=1                'Accende led
   delay_ms(500)            'Aspetta mezzo secondo
   PORTB.0=0                'Spegne led
   delay_ms(500)            'Aspetta mezzo secondo
next k

end.
```

no la pagina di memoria (ampia 256 bytes) tra le otto disponibili nel device (infatti con 3 bit possiamo rappresentare 8 combinazioni possibili: 000, 001, 010, 011, 100, 101, 110, 111).

- L'ultimo bit (1) indica un'operazione di lettura.

Dopo questa operazione avviene la lettura vera e propria del dato da Eeprom e la conseguente memorizzazione sulla variabile *lampeggi* (in Ram). É molto semplice a questo punto creare un ciclo ripetitivo che effettui un numero stabilito di accensioni e spegnimenti del diodo led collegato alla porta RB0.
Un appunto sulla funzione Soft_I2C_Read: essa accetta, come parametro, un valore numerico tra parentesi. Esso indica la possibilità di inviare o meno un segnale di acknowledge allo *slave*.
Se è settato a 0 nessun segnale viene inviato, diversamente il master manda un segnale di acknowledge. In figura 10 si può osservare il flusso dei dati e gli indirizzamenti necessari per la lettura delle informazioni.

CONCLUSIONE E RINGRAZIAMENTI

Bene, con la puntata si conclude il corso che ci ha dato la possibilità di conoscere il mondo dei microcontrollori in maniera "indolore" grazie all'utilissimo compilatore Mikrobasic. Ringrazio la redazione di Fare Elettronica che mi ha supportato (e sopportato) in tutte le mie richieste tecniche e di consigli e con la quale ho instaurato un bellissimo rapporto di lavoro e collaborazione.
Ma soprattutto ringrazio Voi, cari lettori, che con le vostre richieste, suggerimenti e domande mi avete dato lo spunto e la possibilità di preparare gli articoli nel migliore dei modi.
Auguro a tutti Voi un buon lavoro e una buona sperimentazione.

Codice MIP 250050

Eventi casuali

La generazione di eventi casuali ed imprevisti è talvolta un'esigenza indispensabile per molte applicazioni a microcontrollore. Vediamo come fare.

In questo articolo parliamo di eventi casuali. Molte applicazioni li utilizzano per ottenere numeri e sequenze casuali e pseudo-casuali. Basti pensare ai videogiochi e alle apparecchiature ludiche nelle quali è necessario simulare comportamenti sempre imprevedibili per l'utente finale.

IL CASO

Esistono tre tipi di eventi che possono caratterizzare la vita di tutti i giorni (anche quella informatica):

1. **Eventi certi**, che si verificano *certamente* e trovano il loro riscontro in fatti pratici e concreti.
2. **Eventi impossibili**, che non hanno *alcuna possibilità* di un riscontro concreto.
3. **Eventi possibili**, che *potrebbero* verificarsi al sorgere di talune condizioni.

Della prima categoria fanno parte tutti gli eventi che, per qualunque motivo, si verificano sicuramente e trovano un campo applicativo certo. Ad esempio *è sicuro* che il numero estratto a sorte, in seguito ad un lancio di un dado, sia compreso tra 1 e 6. Della seconda categoria fanno parte tutti gli eventi che non hanno alcuna probabilità di realizzarsi. Così, ad esempio, è impossibile che si verifichi il passaggio del pianeta Giove davanti il Sole (visto dalla Terra), per ovvii motivi astronomici.
Della terza categoria fanno infine parte tutti gli eventi che hanno una certa probabilità di verificarsi, con esito incerto ed imprevedibile. Ad esempio, vincere un ambo al gioco del Lotto non è un evento sicuro ma nemmeno impossibile (non per l'autore che non ha mai vinto…). Sono pertanto questi ultimi, oggetto del presente articolo.

Il Caso è solo apparente

Il caso è legato quindi a tutti quegli eventi di cui non si conosce il verificarsi o l'esito a priori. Ma è errato definire un evento casuale e non preoccuparsi più del suo andamento fisico.
Il non conoscere il risultato finale non fa parte di un misterioso arcano nascosto all'uomo o addirittura dell'intervento di oscure forze regolatrici degli andamenti coinvolti. Se ci fosse la piena conoscenza di molti fattori che intervengono sull'evento se ne potrebbe prevedere il risultato finale. Con un esempio cerchiamo di essere più chiari: si supponga di gettare un dado a sei facce sul tavolo: ovviamente non potremo sapere a priori il punteggio che ne verrà fuori.
Ma se prendessimo in considerazione TUTTI i parametri coinvolti nel lancio (parametri fisici, chimici, elettronici, astronomici, umani, ecc) e le relative relazioni fisiche che li legano, sicuramente il risultato potrebbe essere noto sin dall'inizio. Relativamente al lancio del dato, se l'uomo venisse a conoscenza del peso e massa del dado, della forma che assume nello spazio, dell'aspetto aerodinamico dello stesso, del materiale utilizzato, dell'attrito che offre l'aria, della sua velocità angolare, della sua velocità lineare, della forza di gravità, della sua elasticità all'impatto con il tavolo, della forza applicata al suo lancio, dell'altezza da cui avviene il lancio, della durezza e dell'elasticità del supporto di collisione, di eventuali correnti d'aria, di potenziali di elettricità statica generati e molti altri ancora, si saprebbe a priori effettivamente il punteggio di sortita come risultato parametrico di tutti quei fenomeni coinvolti nell'evento. Parametri che naturalmente sono sconosciuti

con Mikrobasic



di Giovanni Di Maria

all'uomo, ma ammesso che siano noti, occorrerebbe eseguire tantissime operazioni matematiche che renderebbero del tutto inadeguato il lancio stesso. E il gioco dei dadi è forse uno dei casi più semplici in cui interviene un modesto numero di parametri e fatti esterni che lo condizionano.

LA PROBABILITÀ

Dal momento che ogni evento possibile può o non può verificarsi (secondo certe condizioni), se ne può misurare la possibilità con cui esso può trovare svolgimento pratico.
Tale misura è definita *probabilità* ed è il rapporto esistente tra i casi favorevoli ed il numero dei casi possibili che si presentano. In pratica essa rappresenta un numero che è tanto più alto quanto è l'opportunità che l'evento accada.
Ad esempio, la probabilità di indovinare il numero cinque al lancio di un dado è dato da:

1:6 = 0,166 (oppure 16,6%)

in cui il numeratore indica il numero dei casi interessati mentre il denominatore indica il numero dei casi possibili.
Questa breve premessa è necessaria per intraprendere l'applicazione del concetto del caso alla programmazione pratica e sua implementazione su microcontrollore.



Figura 1 *Dati, Carte da gioco, numeri della Tombola e del Lotto: chiari esempi di generazione casuale*

IL CASO NEI MICROCONTROLLORI

Un microcontrollore è una macchina capace di eseguire e produrre risultati programmati e scontati (sistema deterministico). Generalmente esso non è capace di produrre risultati assolutamente casuali o, meglio, sconosciuti ed imprevedibili per l'uomo. Ci sono occasioni in cui occorre la generazione di eventi o di numeri casuali, per poter dare un senso di *realtà* e di "umanità" alle applicazioni informatiche create. La macchina non è assolutamente in grado di generare un evento casuale, ed il più delle volte occorre affiancare questa funzione ad un **evento esterno** che agevoli il raggiungimento del risultato. Per generare numeri o sequenze di numeri casuali, possiamo seguire alcuni metodi. Approfondiamoli di seguito.

PRIMO METODO: UTILIZZO DEL REGISTRO TMR0 DEL PIC

Il **TMR0** è un registro interno a 8 bit che funge da timer e da contatore. Esso è incrementato automaticamente ad ogni ciclo macchina. Siccome il suo avanzamento è del tutto trasparente all'operatore, si può efficacemente utilizzarlo quale generatore di numeri casuali.
Purtroppo l'incremento di tale registro deve essere opportunamente programmato.
Fortunatamente il Mikrobasic dà l'accesso, a livello di bit (basso livello), a molti registri interni della MCU. A tale scopo si osservi lo schema a blocchi in figura 2. Si consideri anche il registro **OPTION** raffigurato in figura 3. La frequenza di conteggio del registro TMR0 dipende dall'impostazione dei blocchi che lo precedono. Tale registro può determinare la modalità di conteggio attraverso il contatore interno oppure con impulsi esterni. Nel primo caso è lo stesso clock del Pic a fornire l'impulso di avanzamento, con frequenza divisa per quattro. Per questo motivo il bit **T0CS** (t, zero, C, S) del registro OPTION (raffigurato in rosso) deve essere

settato a 1 logico. Se non si ha la necessità di suddividere ulteriormente la frequenza di conteggio, non occorre coinvolgere il *Prescaler* . Tale scopo si ottiene settando a 1 logico il bit **PSA** del registro OPTION (raffigurato in giallo). In questo modo il registro TMR0 avanza di un'unita alla frequenza di 1 Mhz, se il clock selezionato è di 4 Mhz.
Tale conteggio avviene sequenzialmente, e al limite massimo raggiunto (255) riparte nuovamente da zero. In effetti le cadenze generate (sequenziali, scontate e con un intervallo 0-255) sono ben lungi da essere considerate casuali, ma con un piccolo artifizio esterno si può rendere imprevedibile la sequenza.

Come si programma il conteggio

Adesso si può configurare il registro OPTION attraverso una normale assegnazione binaria in Mikrobasic, con la seguente linea di comando `OPTION_REG=%00001000` che setta i *flag* come specificato sopra.

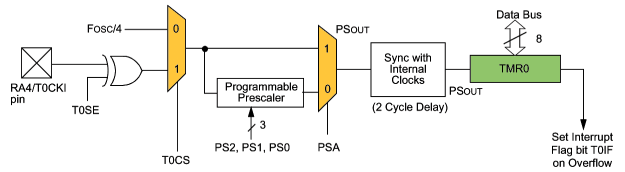


Figura 2 *Funzionamento del registro TMR0*

OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Figura 3 *I bit del registro OPTION*

Visualizzare i numeri generati

Per la visualizzazione delle sequenze numeriche generate si può utilizzare un display LCD collegato al microcontrollore. Si consideri pertanto lo schema elettrico di cui alla figura 4.

Il programma dei numeri NON casuali

Nel primo esempio il prototipo visualizza sul display, per ogni secondo trascorso, un numero "pseudo-casuale" da 0 a 255. Il valore mostrato è prelevato dal registro TMR0 che, come detto prima, avanza ciclicamente ad una velocità molto alta, quindi imprevedibile e continuamente in progressione.
L'unico problema di questo approccio sta nel fatto che ad ogni riavvio o reset del sistema, la sequenza proposta è sempre la stessa. Si digiti sull'editor del Mikrobasic il listato 1.
Eseguendo il programma su microcontrollore, si osserverà la visualizzazione cadenzata di numeri, apparentemente casuali, sul display ma se si spegne il circuito e si riavvia nuovamente, la progressione sarà sempre la stessa. Questo perché il PIC, quale macchina deterministica, esegue sempre la stessa sequenza di azioni e operazioni.

Il programma dei numeri CASUALI

Volendo generare sequenze casuali, o per lo meno non prevedibili, occorre deviare il preciso

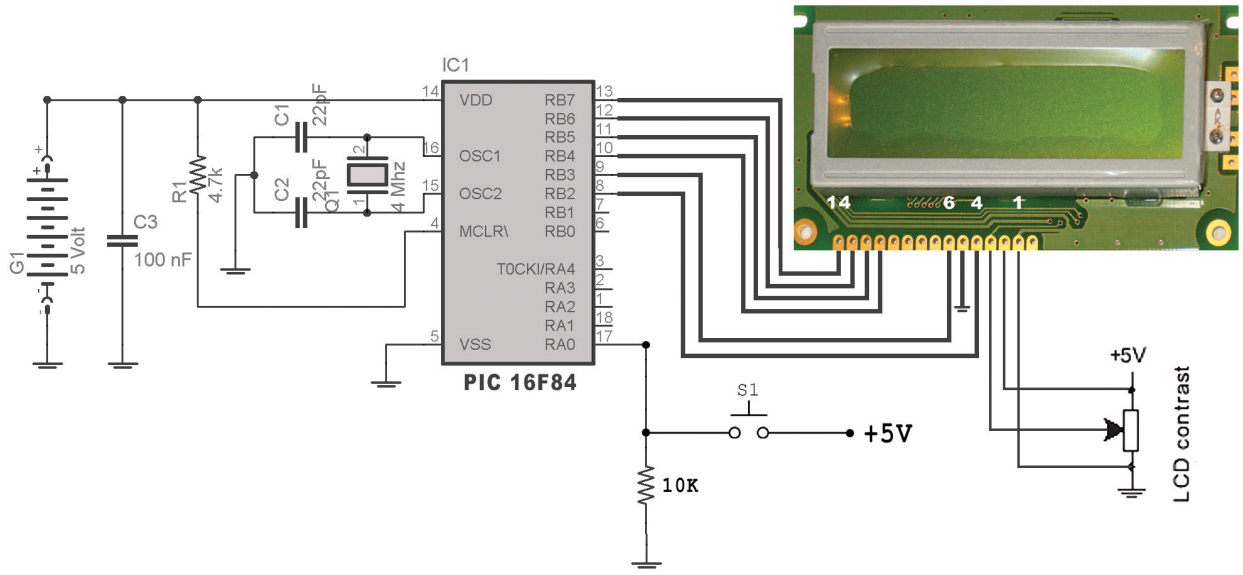


Figura 4 *Schema elettrico del primo prototipo*

percorso dell'esecuzione mediante un evento esterno. Tale comportamento, estrinseco al microcontrollore, deve essere:

- Inatteso.
- Imprevisto.
- Possibilmente ripetuto.
- Variabile nel tempo.
- Improvviso.
- Incerto.

Per ottenere un simile evento si può ricorrere ad un pulsante normalmente aperto, da premere casualmente all'occorrenza. Tale pressione sarà valutata internamente dal software attraverso alcune istruzioni del tutto "inutili" ai fini della programmazione, ma che implicano un cambio di percorso del software e la variazione dei tempi di esecuzione con la conseguente modifica imprevista del registro TMR0.
Seguendo sempre lo schema elettrico della figura 4, si scriva il listato 2. Questa volta, eseguendo il programma, si può notare come i numeri e le sequenze siano sempre diverse; occorre ogni tanto premere il tasto, per assicurare degli eventi casuali e sempre diversi. La pressione del pulsante, assieme alle relative istruzioni di controllo, assicurano un'esecuzio-

[Listato 1]

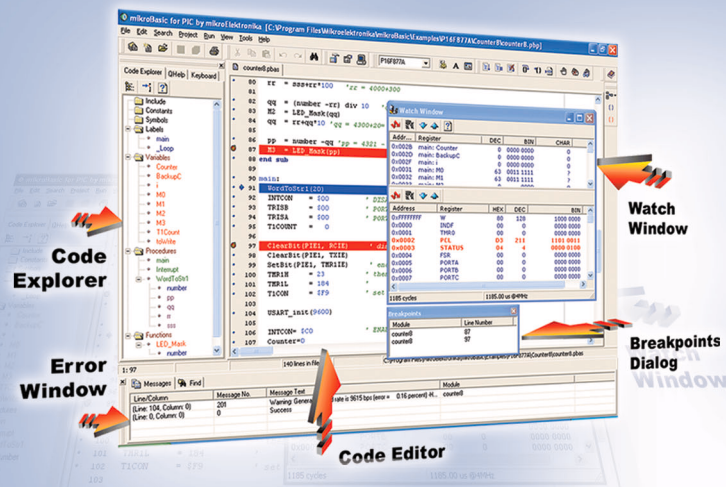
```
program caso1

dim st as string[3] 'Stringa per il Display
OPTION_REG=%00001000'Setta registro OPTION
trisb=0             'PORTB in output

lcd_init(portb) 'Prepara Lcd su PORTB
lcd_cmd(Lcd_Cursor_Off) 'Disabilita cursore
while true      'Ciclo infinito
   delay_ms(1000)  'Attesa di un secondo
   bytetostr(TMR0,st) 'TMR0 in stringa
   lcd_out(1,1,"Numero estratto:")
   lcd_out(2,1,st)    'Stampa risultato
wend

end.
```


Software Mikrobasic



Un potente compilatore Basic per PICmicro

- ✔ Code Editor
- ✔ Code Explorer
- ✔ Debugger
- ✔ Statistiche

Tutto in un ambiente Windows facile ed intuitivo

Un set di strumenti veramente indispensabili per sviluppare applicazioni con i PICmicro

Codice MIP 251081

Ordinalo subito su www.farelettronica.com oppure telefona allo 02.66504794

ne del software (a livello macchina) sempre diversa, pertanto il registro TMR0 assumerà differenti valori non prevedibili. Si ribadisce che i numeri visualizzati in questo caso sono compresi tra 0 e 255.

[Listato 2]

```
program caso2

dim st as string[3] 'Stringa per il Display
dim k as byte
OPTION_REG=%00001000 'Setta registro OPTION
trisa=%00001    'PORTA.0 in INPUT
trisb=0         'PORTB in output

lcd_init(portb) 'Prepara Lcd su PORTB
lcd_cmd(Lcd_Cursor_Off) 'Disabilita cursore
while true      'Ciclo infinito
   for k=1 to 100
      delay_ms(10)  'Attesa 10 millisecondi
      if PORTA.0=1 then  'Se premo tasto
         delay_ms(10)  'Ulteriore attesa
      end if
   next k
   bytetostr(TMR0,st) 'TMR0 in stringa
   lcd_out(1,1,"Numero estratto:")
   lcd_out(2,1,st)    'Stampa risultato
wend

end.
```

SECONDO METODO: UTILIZZO DI UNA RETE RC

Il secondo metodo presuppone sempre l'intervento esterno dell'uomo per rendere casuali i numeri generati. Esso si basa sostanzialmente sulla scarica



Figura 5 Il prototipo funzionante

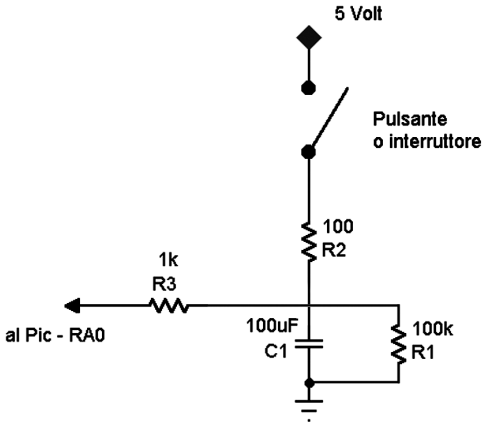


Figura 6 Schema di principio del generatore casuale

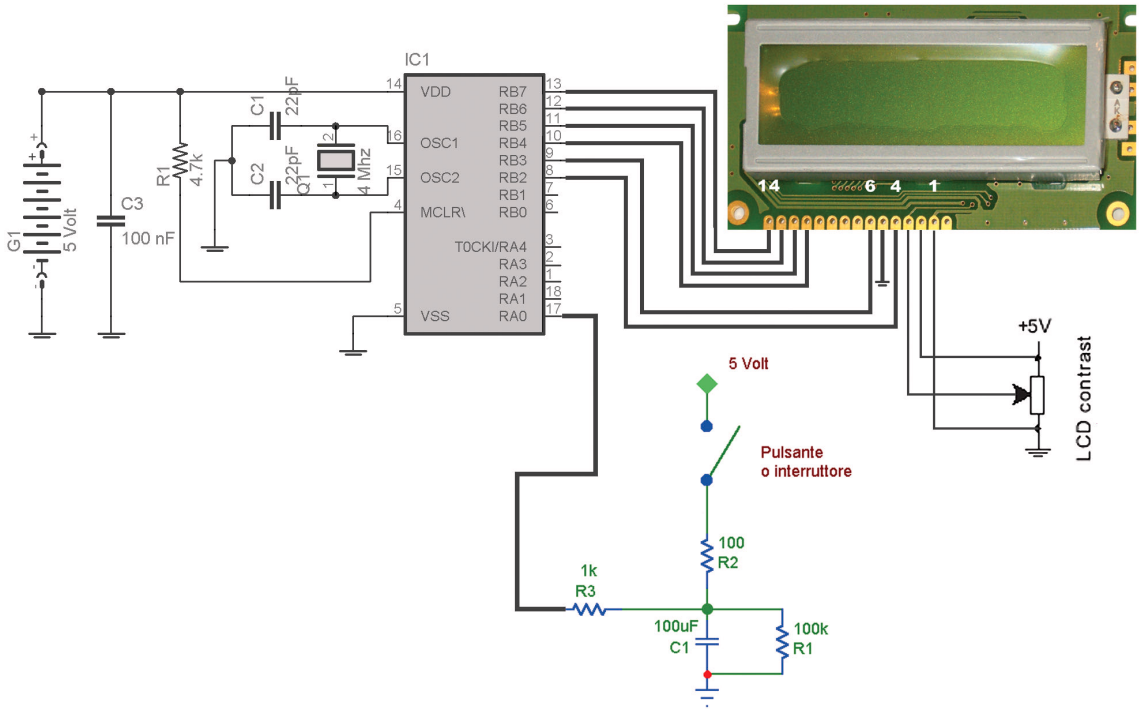


Figura 7 Schema elettrico completo del secondo prototipo

di un condensatore elettrolitico e relativa rilevazione digitale da parte del microcontrollore.

In dettaglio

Questo sistema prevede l'utilizzo di un condensatore, quale erogatore graduale di tensione discendente, in parallelo ad una resistenza, che ha la funzione di scaricarlo poco alla volta.
Attraverso un interruttore, che collega il circuito all'alimentazione, è possibile caricare istantaneamente il condensatore. Il circuito applicativo è raffigurato in figura 6. Vediamone i dettagli.
Inizialmente il pulsante è aperto, nessuna corrente fluisce nel circuito, ai capi del condensatore non c'è tensione e all'uscita, tramite la resistenza da 1 Kohm, troviamo il potenziale di 0 Volt. Tale livello costituisce un valore logico BASSO che l'ingresso del Pic (PORTA.0) interpreta come tale.
Appena il pulsante viene premuto e rilasciato, il condensatore si carica istantaneamente attraverso il resistore da 100 ohm, portandosi al potenziale di 5 Volt. Tale stato perdura anche dopo il rilascio del pulsante, per un periodo proporzionale alla costante RC, infatti il resistore in parallelo da 100 Kilohm, scarica molto lentamente il condensatore. In questo caso, all'uscita troviamo un potenziale che determina un valore digitale ALTO, sin quando la tensione raggiunge circa 1,3 Volt. Dopo di che torna nuovamente al livello basso.
Ciò che è importante comprendere è il fatto della imprevedibilità della durata del livello logico alto, determinata dalla pressione del tasto e dalla relativa "imprecisione" intrinseca del condensatore.
In figura 8 è riportato l'andamento analogico e digitale del condensatore. La traccia blu mostra la scarica inesorabile del condensatore, mentre la traccia rossa segue l'andamento digitale dello stesso componente, visto dal microcontrollore.
Il transitorio, come si vede, si conclude dopo circa 14 secondi. Se tale tempo dovesse risultare eccessivo si può abbassare la capacità del condensatore o diminuire il valore della resistenza R1. La figura 7 mostra lo schema pratico completo dell'intero prototipo.

La funzione dei componenti

Riguardiamo nuovamente lo schema di principio di figura 6 per fornire la funzione specifica

circuiti stampati in 24 ore

garantiamo il tempo di consegna: 24 ore o i circuiti sono gratis

Potrete scegliere tra singola e doppia faccia con foro metallizzato. Con solder e serigrafie per uno stampato di alta qualità o solo piste stagnate per un prototipo a basso costo.
Prezzi a partire da* € 14,38 (doppia faccia foro metallizzato 7,50x7,50 cm) e da € 9,13 (singola faccia 7,50x7,50 cm) per FR4 1,6 mm con rame 35 µm, **tutti comprensivi di attrezzatura.**
Nessuna limitazione sul numero dei fori, sul numero degli utensili (diametri) e sul tipo di scontornatura (anche tondeggiante).
Distanza minima tra le piste e pista minima 8 mils (0,20 mm).

PREVENTIVO ANONIMO, GRATUITO ED IMMEDIATO con il nostro calcolatore online.



Codice MIP 251083

visita il nostro sito per il dettaglio delle note tecniche

www.mdsrl.it

millennium md dataware

millennium dataware srl
parco scientifico e tecnologico
15050 rivalta scrivia - tortona (al)
tel. 0131 860.254 fax 0131 860157
www.mdsrl.it info@mdsrl.it

* i prezzi si intendono iva esclusa e calcolati sul singolo pezzo - ordine minimo 2 pezzi

per ogni componente elettrico utilizzato:

- Pulsante: ha il compito di caricare il condensatore, simulando il lancio della moneta o dei dadi.
- R2 (100 ohm): ha il compito di caricare istantaneamente il condensatore e, per il suo ridotto valore, determina una costante di tempo molto bassa.
- C1 (100 µF): costituisce il "serbatoio " della procedura, implicando un tempo di attesa casuale per la determinazione dei valori.
- R1 (100K): ha l'importante funzione di scaricare lentamente ed inesorabilmente il condensatore, permettendo di tracciare un andamento discendente progressivo.
- R3 (1K): permette il collegamento al pin d'ingresso del microcontrollore. Se ne potrebbe fare a meno, data l'alta impedenza di quest'ultimo.

Il Programma

Il software è diverso dal precedente. Si consideri il listato 3. Dopo l'inizio del ciclo infinito (While – Wend) il Pic invita a premere il tasto. Tale ciclo di attesa è racchiuso all'interno di un blocco (senza istruzioni interne) che si protrae per tutto il tempo in cui l'ingresso PORTA.0 rimane a livello logico basso (tasto non premuto). Se tale livello cambia, vuol dire che il pulsante è stato sottoposto a pressione, per cui l'esecuzione del programma continua con le istruzioni successive.
Con il condensatore completamente carico, il successivo ciclo controlla il livello logico alto dell'ingresso. All'interno si trova la variabile caso che incrementa ad ogni iterazione, generando in questo modo una sorta di numero casuale. Numero che non può superare il valore di 90 grazie alla condizione di controllo.

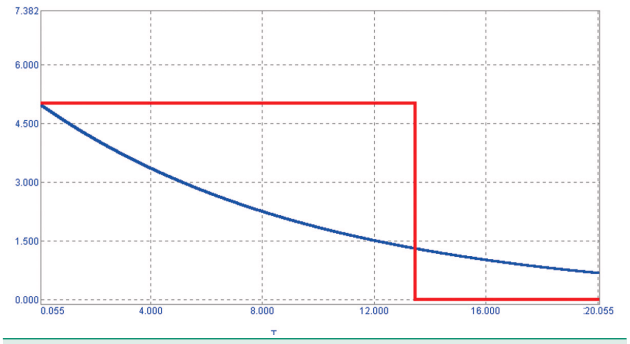


Figura 8 Grafico dell'andamento del condensatore e suo livello "logico"



Figura 9 Estrazione in corso

[Listato 3]

```
program caso3

dim st as string[3] 'Stringa per il Display
dim caso as byte
trisa=%00001    'PORTA.0 in INPUT
trisb=0         'PORTB in output

delay_ms(1000) 'Pausa iniziale
lcd_init(portb) 'Prepara Lcd su PORTB
lcd_cmd(Lcd_Cursor_Off) 'Disabilita cursore
while true       'Ciclo infinito
   caso=0      'Azzera variabile caso
   rem ----Invita a premere il tasto----
   lcd_out(1,1,"Premere il tasto")
   while PORTA.0=0
      rem Il programma attende il tasto
   wend
   lcd_cmd(Lcd_Clear) 'Cancella display
   lcd_out(1,1,"ESTRAZIONE ...")
   rem --Il tasto e' stato premuto---
   while PORTA.0=1    'Finche C e' carico...
      caso=caso+1     'Incrementa caso
      if caso>90 then
         caso=1
      end if
   wend
   rem ----Stampa risultato----
   bytetostr(caso,st) 'Converte in stringa
   lcd_cmd(Lcd_Clear) 'Cancella display
   lcd_out(1,1,"Estratto:")
   lcd_out(2,1,st)
   delay_ms(5000) 'Attende 5 secondi
   lcd_cmd(Lcd_Clear) 'Cancella display
wend

end.
```

Non appena il condensatore, scaricandosi, raggiunge il valore logico basso (dopo qualche secondo), il ciclo cessa di iterare è viene mostrato sull'Lcd il valore della variabile caso, quale numero generato. Dopo una pausa di 5 secondi, l'intero processo ricomincia da capo.

TERZO METODO: CLOCK CASUALE ESTERNO

Probabilmente il terzo metodo è il più efficace di quelli descritti, poiché si basa effettivamente su eventi esterni variabili ed imprevisti, di cui naturalmente l'uomo non può fare ipotesi di sorta.

Il generatore di impulsi variabile

Guardiamo lo schema di principio in figura 10. Si tratta di un classico generatore di onda quadra con periodo molto breve. E' formato da due porte NOT *bufferate* (integrato CD40106 – figura 15). La prima si occupa della generazione effettiva di un treno di impulsi, con frequenza dipendente dal condensatore e dalla resistenza ad esso in parallelo. La seconda invece "squadra" esattamente l'impulso, isolandolo e rendendolo utilizzabile direttamente dal PIC (figure 13 e 14). La particolarità sta nella generazione "casuale" del periodo, in quanto esso è determinato soprattutto dalla fotoresistenza che rende variabile la frequenza generata. Basta una minima variazione di luce ambientale per modificare enormemente il periodo, rendendo così imprevedibile il risultato finale.

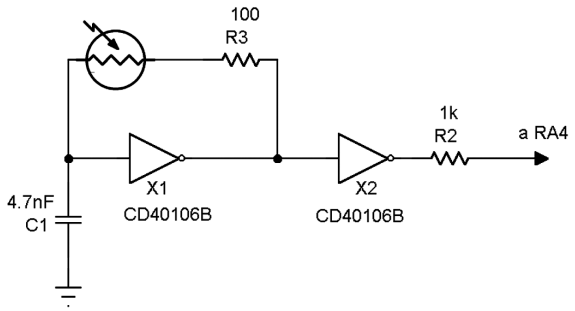


Figura 10 Schema elettrico del generatore di impulsi variabile

In dettaglio

Vediamo adesso come opera l'intero circuito, visibile in figura 11. La PORTA.4 del microcontrollore è adibita alla ricezione del treno di impulsi. Lo scopo principale del software è pertanto quello di "contare" ciclicamente tali segnali, attraverso il registro TMR0.
A differenza però del primo metodo esposto, l'avanzamento del conteggio stavolta è determinato da un evento esterno variabile, e non dallo stesso microcontrollore.
Dato che il conteggio avviene a grande velocità, data la frequenza in gioco, non è possibile stabilire a priori il risultato finale. Esaminiamo nuovamente le figure 4 e 5.
Questa volta il segnale da conteggiare (esterno) è prelevato attraverso l'ingresso RA4, l'unico predisposto all'avanzamento del registro TMR0. Per ottenere questo scopo è necessario porre a 1 logico il bit **T0CS** del registro OPTION. Il bit **T0SE**

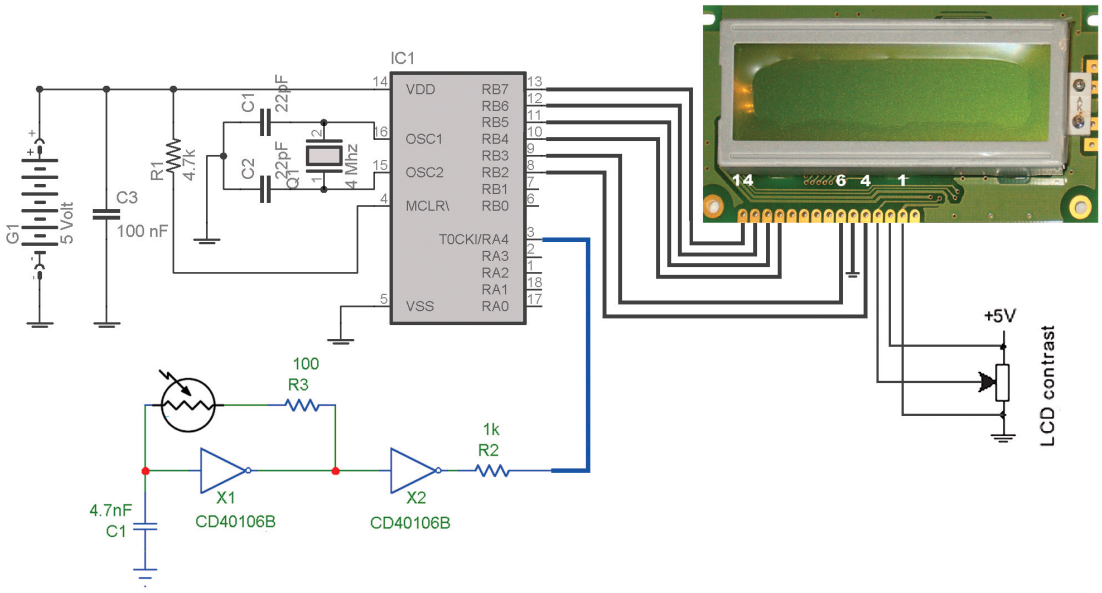


Figura 11 Schema elettrico completo del terzo prototipo

è invece posto a 0 logico per far si che il conteggio avvenga in prossimità dei *fronti di salita* del segnale. Inoltre il bit **PSA** deve essere messo a 1 logico per evitare l'utilizzo del *prescaler* ed avere così l'incremento unitario del registro TMR0 ad ogni clock esterno.
Il valore binario da assegnare al registro OPTION è il seguente: 00101000.



Figura 12 La fotoresistenza

Parametri di frequenza

La frequenza dell'oscillatore esterno ad onda quadra è determinata dal condensatore, che collega l'ingresso della prima porta NOT a massa, e dalla resistenza che collega l'ingresso della stessa alla sua uscita. Più basso è il valore di questa, più elevata sarà la frequenza prodotta. In questo caso l'elemento variabile è rappresentato dalla fotoresistenza che, con minime variazioni di luce ambientale, modificherà l'intero andamento del circuito e, quindi, la casualità della sortita. Tipicamente una fotoresistenza prevede un valore resistivo di circa 25 ohm in piena luce e di circa 4 Megaohm al buio completo. Ma utilizzando il prototipo in una stanza "normalmente" illuminata, essa può assumere dei valori medi nell'intervallo 500 ohm – 10 Kohm. Con questa estensione il circuito genera un'onda quadra con frequenza compresa tra 25 Khz e 250 Khz, frequenza che viene elaborata dal software del PIC.

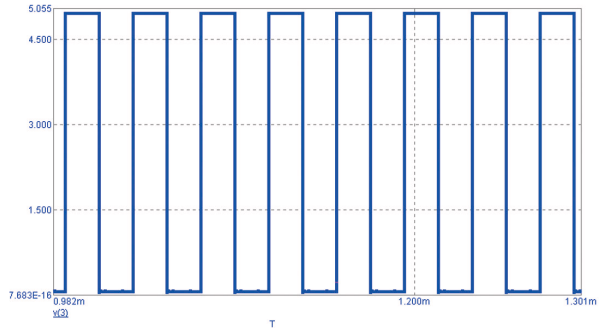


Figura 13 L'onda quadra generata dal circuito

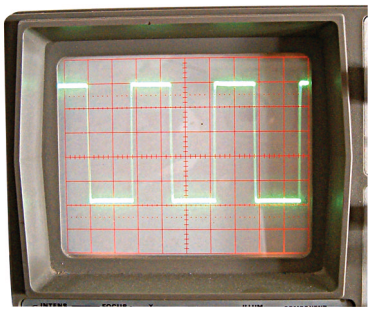


Figura 14 Oscillogramma del treno di impulsi

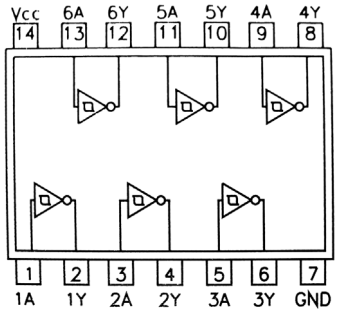


Figura 15 Pinout CD 40106

Il programma

Il software in questione (listato 4) genera ogni secondo un numero casuale compreso tra 0 e 255, secondo la tecnica descritta.

COME OTTENERE L'INTERVALLO DESIDERATO

Se si desidera ottenere un insieme di valori specifici occorre eseguire alcuni semplici calcoli per filtrare in questo modo i risultati ottenuti. Può capitare infatti di aver bisogno dei numeri da 1 a 6 per realizzare un dado elettronico oppure

[Listato 4]

```
program caso4

dim st as string[3] 'Stringa per il Display
trisa=%10000    'PORTA.4 in INPUT
trisb=0         'PORTB in output

delay_ms(1000) 'Pausa iniziale
lcd_init(portb) 'Prepara Lcd su PORTB
lcd_cmd(Lcd_Cursor_Off) 'Disabilita cursore
OPTION_REG=%00101000    'Registro OPTION
while true       'Ciclo infinito
   bytetostr(TMR0,st) 'TMR0 in stringa
   lcd_cmd(Lcd_Clear) 'Cancella display
   lcd_out(1,1,"Estratto:")
   lcd_out(2,1,st)
   delay_ms(1000)
wend

end.
```

dei numeri da 1 a 90 per approntare una tombola elettronica. Per raggiungere lo scopo basta semplicemente calcolare il resto della divisione tra il numero generato ed il numero di elementi del *range* desiderato, aggiungendo al risultato il valore uno, secondo la formula:

$$n=(p \bmod e)+1$$

dove: "**n**" è il numero da generare, "**p**" è il numero generato dal PIC, "**e**" è il numero di elementi dell'intervallo desiderato.
Esempio: si vuole simulare il lancio di un dado (1-6) adeguando il valore del registro TMR0 (0-255) con la formula data.

$$n=(TMR0 \bmod 6)+1$$

Il software pertanto deve prevedere tale "conversione" e tagliare tutti i valori non desiderati (tabella 1).

Il programma

La generazione del numero casuale segue esattamente lo procedura del metodo 3. Si differenzia solo per la riduzione del valore generato a cifre utili. Il programma provvede a "lanciare" il dado ogni secondo, visualizzando l'esito sul display LCD. Si scriva pertanto il listato 5 e si compili attraverso le opportune procedure.

[Listato 5]

```
program caso5

dim st as string[3] 'Stringa per il Display
dim dado as byte    'Variabile per il lancio
trisa=%10000    'PORTA.4 in INPUT
trisb=0         'PORTB in output

delay_ms(1000) 'Pausa iniziale
lcd_init(portb) 'Prepara Lcd su PORTB
lcd_cmd(Lcd_Cursor_Off) 'Disabilita cursore
OPTION_REG=%00101000    'Registro OPTION
while true       'Ciclo infinito
   lcd_cmd(Lcd_Clear) 'Cancella display
   lcd_out(1,1,"Lancio del DADO:")
   dado=(TMR0 mod 6)+1  'Genera da 1 a 6
   bytetostr(dado,st) 'Converte in stringa
   lcd_out(2,1,st)    'Stampa il risultato
   delay_ms(1000)
wend

end.
```

LA NON RIPETIZIONE DEI NUMERI

L'ultimo paragrafo dell'articolo illustra un metodo per evitare la ripetizione dei numeri già estratti, procedura molto utile per simulare il gioco della Tombola, nella quale ogni numero è sorteggiato una volta sola.
Nell'esempio che segue verranno estratti i

[Listato 6]

```
program caso6

dim st as string[3] 'Stringa per il Display
dim uscito as byte[20]   'vettore di 20 flag
dim k as byte         'Variabile di servizio
dim caso as byte  'Contiene il numero casuale
trisa=%10000    'PORTA.4 in INPUT
trisb=0         'PORTB in output

delay_ms(1000) 'Pausa iniziale
lcd_init(portb) 'Prepara Lcd su PORTB
lcd_cmd(Lcd_Cursor_Off) 'Disabilita cursore
OPTION_REG=%00101000    'Registro OPTION
rem ------Azzeramento vettore-------
for k=0 to 19  'Processa il vettore
    uscito[k]=0    'numero non ancora uscito
next k
for k=1 to 20
   lcd_cmd(Lcd_Clear) 'Cancella display
   lcd_out(1,1,"Giocatore:")
lancio:              'LABEL
   caso=(TMR0 mod 20)+1  'Genera da 1 a 20
   if uscito[caso-1]=1 then
      'Se e' stato estratto prima
      goto lancio       'RITENTA il lancio
   end if
   uscito[caso-1]=1    'Lo MARCA come USCITO
   bytetostr(caso,st) 'Converte in stringa
   lcd_out(1,12,st)    'Stampa il risultato
   delay_ms(5000)      'Attende 5 secondi
next k
lcd_out(2,1,"FINE ESTRAZIONE")

end.
```

TMR0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	249	250	251	252	253	254	255
(TMR0 mod 6)+1	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	...	4	5	6	1	2	3	4

Tabella 1 *Conversione di TMR0 in un range utile*



Figura 16 Terno Secco in funzione

[Listato 7]

```
program caso7

dim st as string[3] 'Stringa per il Display
dim n1 as byte      'Primo numero estratto
dim n2 as byte      'Secondo numero estratto
dim n3 as byte      'Terzo numero estratto
trisa=%10000    'PORTA.4 in INPUT
trisb=0         'PORTB in output

delay_ms(1000) 'Pausa iniziale
lcd_init(portb) 'Prepara Lcd su PORTB
lcd_cmd(Lcd_Cursor_Off) 'Disabilita cursore
OPTION_REG=%00101000    'Setta Registro OPTION
lcd_cmd(Lcd_Clear) 'Cancella display
lcd_out(1,1,"Gioca il Terno:")
while true
    n1=(TMR0 mod 90)+1   'Genera da 1 a 90
    delay_ms(100)
    n2=(TMR0 mod 90)+1   'Genera da 1 a 90
    delay_ms(100)
    n3=(TMR0 mod 90)+1   'Genera da 1 a 90
    delay_ms(100)
    rem ---Controlla se ci sono DOPPIONI----
    if (n1=n2) or (n1=n3) or (n2=n3) then
       continue    'Ripete estrazione
    end if
    rem ----Stampa i tre numeri-----
    bytetostr(n1,st) 'Converte n1 in stringa
    delay_ms(1000)
    lcd_out(2,1,st)    'Stampa n1
    bytetostr(n2,st) 'Converte n2 in stringa
    delay_ms(1000)
    lcd_out(2,5,st)    'Stampa n2
    bytetostr(n3,st) 'Converte n3 in stringa
    delay_ms(1000)
    lcd_out(2,9,st)    'Stampa n3
    break              'Esce dal ciclo
wend

end.
```

numeri di 20 giocatori. Pertanto ogni cifra deve uscire casualmente ma UNA volta sola!
Il metodo si basa sull'utilizzo di un vettore unidimensionale, che funge da *flag* per indicare se un valore è già stato estratto oppure no. Inizialmente tutti i venti elementi del vettore saranno posti a zero, e man mano che l'estrazione procede, verranno rimpiazzati dal valore uno. Esso permette la non ripetizione di un numero se esso è stato già estratto.
La generazione dei venti numeri prosegue al ritmo di 5 secondi. Il programma è mostrato nel listato 6.

TERNO SECCO!

Come ultimo esempio proponiamo un generatore di tre numeri compresi tra 1 e 90 da giocare al gioco del Lotto quale terno secco su una ruota a scelta.
Il metodo è sempre lo stesso. Si tratta di generare, uno alla volta, tre numeri compresi tra 1 e 90. Se questi vengono ripetuti il software, contenuto nel listato 7, provvederà a effettuare nuovamente l'estrazione. Non si utilizza nell'esempio il vettore poiché avviene una verifica immediata di uguaglianza tra i tre numeri estratti.
Il terno viene generato in sequenza, mostrando una cifra ogni secondo, aumentando in questo modo la suspance del gioco.
Troviamo due nuove istruzioni all'interno del blocco *while – wend*:

- **continue**, che provvede a riprendere l'esecuzione del programma dall'inizio del blocco in cui essa è contenuta.
- **break**, che termina la "vita" del blocco stesso e continua l'esecuzione del programma dall'istruzione immediatamente successiva a *wend*.

CONCLUSIONI

Come si vede, non esiste limite alla programmazione. Occorre sempre analizzare a fondo qualsiasi problema e cercare ogni strategia che lo possa determinare. Con le dovute tecniche ed espedienti vari si riesce sempre a trovare la soluzione a qualsiasi tipo di algoritmo, anche quello esaminato nell'articolo, apparentemente irrisolvibile.

Codice MIP 251078

Gestire gli interrupt

Una puntata particolare nella quale si affronta un argomento molto delicato: la gestione degli interrupt con il compilatore Mikrobasic ed il microcontrollore PIC16F84. Un'operazione riservata ai linguaggi a basso livello che adesso è realizzabile anche con quelli ad alto livello.

La programmazione degli *interrupt* suscita sempre qualche perplessità e, diciamolo pure, anche un po' di paura. In effetti l'argomentazione è un po' delicata e se affrontata con superficialità, e senza una buona dotazione di strumentazione sofisticata, il progetto è destinato, se non a fallire, a non soddisfare in pieno le esigenze del programmatore. Nell'articolo si vedrà come rendere "amichevole" una delle più complicate procedure della teoria e pratica della programmazione informatica.

LA PROGRAMMAZIONE SEQUENZIALE

Il Basic, come altri dialetti ad "alto livello", è un linguaggio ad esecuzione prettamente "sequenziale". Ciò significa che ogni istruzione viene elaborata e concretizzata una dopo l'altra, dalla CPU. Se vogliamo essere più precisi, l'intero codice ad alto livello è tradotto in istruzioni più "piccole", attraverso il compilatore, in linguaggio macchina, ma anche queste risultano, alla fine, attuate sempre in maniera progressiva.
Qualcuno potrebbe obiettare, dicendo che esistono istruzioni e comandi che permettono di "modificare" il normale percorso del software, considerando ad esempio il salto condizionato oppure i cicli che consentono di rendere imprevedibile il normale svolgimento del processo. Ma se si approfondisce bene il concetto, si vede che, alla fine, l'iter discorsivo e graduale è pur sempre rispettato.

COSA È UN INTERRUPT

Cerchiamo adesso di avvicinarci sempre più all'argomento, cercando di familiarizzarne il concetto. Un **Interrupt** è una modalità operativa gestibile da un microprocessore o da un microcontrollore. Consiste nella possibilità di poter "variare" il normale processo o flusso di esecuzione di un programma, "deviando" l'attenzione della CPU o MCU verso un altro insieme di istruzioni. L'attivazione di esso è causata da un evento ben preciso esterno o interno al sistema. Somiglia, in linea di principio, ad una *subroutine* infatti, in presenza di un interrupt, l'esecuzione del programma principale si arresta immediatamente ed il controllo passa ad altri comandi. Al termine il normale flusso riprende, continuando così l'azione interrotta precedentemente. Praticamente esso riesce ad intercettare un evento, solitamente non previsto e deter-

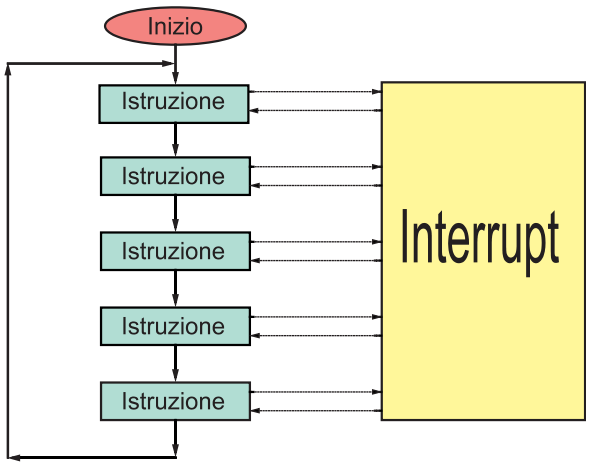


Figura 1 Flusso di principio di un Interrupt

con Mikrobasic



di Giovanni Di Maria

minato da fatti esterni, per poi eseguire una porzione di codice "alternativo" e infine tornare al punto del programma prima interrotto.
In teoria si potrebbe evitare l'utilizzo degli interrupt, ma questo implicherebbe una maggiore difficoltà nella codificazione, in quanto occorrerebbe implementare la rilevazione dell'evento esterno con un software appositamente scritto, che controlli continuamente la presenza di un ipotetico segnale (polling).
La gestione degli interrupt è estremamente complessa in un Personal Computer, infatti in un Microprocessore è veramente notevole il numero di controlli e interruzioni.
Fortunatamente il nostro MCU 16F84 semplifica la vita al programmatore, limitando il suo raggio d'azione a soli quattro tipi di eventi esterni ed imprevisti.
Come si vede in figura 1, il normale flusso di un programma può essere modificato da una "chiamata" di interrupt. Con la doppia linea tratteggiata, si evidenzia che la chiamata non è un fatto sicuro e certo, ma può avvenire in maniera del tutto imprevista e con percentuali di attuazione estremamente variabili. Si noti infine che l'interruzione può avvenire in qualsiasi momento, senza preavviso, con il conseguente nomale ritorno alla routine di partenza.

GLI INTERRUPT GESTIBILI

Come detto prima, il PIC permette solamente quattro tipi di "rilevazione" automatica durante l'esecuzione di un programma. Due di questi sono di tipo "esterno", gli altri due di tipo "interno". Vediamo quali:

- Interrupt causato dalla transizione di stato della porta RB0/INT (evento esterno).
- Interrupt causato dalla transizione di stato delle porte RB4, RB5, RB6, RB7, considerate naturalmente come input (evento esterno).
- Interrupt causato dall'azzeramento sequenziale del registro TMR0 (evento interno).
- Interrupt causato dall'esecuzione di un completo ciclo di scrittura sulla Eeprom del Pic (evento interno).

Cambiamento di stato di RB0/INT

Questo interrupt viene invocato quando lo stato logico della porta RB0 del Pic (pin 6) cambia il suo livello. Si tratta ovviamente di un evento esterno, per il quale occorre il verificarsi di una variazione del potenziale digitale della porta in questione (0 Volt – 5 Volt).
Quando avviene la transizione, il programma corrente viene abbandonato con il conseguente salto alla routine di interrupt. Questo è un classico interrupt esterno.

Cambiamento di stato di RB4, RB5, RB6 e RB7

Anche la modifica dello stato logico di queste quattro porte è considerata "scatenante" ai fini dell'attivazione dell'interrupt. Basta infatti una rapida variazione della tensione sulle porte in questione per richiamare direttamente l'interrupt. Anche quest'altro tipo di evento costituisce un classico interrupt esterno.

Azzeramento di TMR0

Il registro TMR0 è un particolare byte che avanza automaticamente durante l'esecuzione di un programma (anche se esso è temporaneamente "congelato" da una lunga istruzione di pausa). Il microcontrollore invoca un segnale di interrupt non appena il valore di tale byte supera il limite massimo consentito dagli 8 bit (255), ossia al verificarsi del relativo *overflow*. Si tratta di un interrupt interno, poiché provocato da fatti appartenenti implicitamente alla MCU.

Ciclo completo di scrittura su Eeprom

Non appena un programma termina un ciclo normale di memorizzazione su Eeprom, viene

invocato un interrupt. Anche in questo caso si tratta di un evento interno.

IL REGISTRO INTCON

Come si è visto, è sufficiente che uno dei quattro fatti si verifichi, per causare immediatamente un interrupt. Esistono però casi in cui tale rilevazione non serve assolutamente. A tale scopo è possibile abilitare o disabilitare un singolo interrupt, in modo da gestirlo autonomamente o in gruppo con altri tipi di eventi. Tale scelta è realizzabile intervenendo direttamente, e a "basso livello", sul registro **INTCON**, registro composto da 8 bit e che costituisce un *byte* indirizzabile anche direttamente da un opportuno valore decimale. La figura 2 mostra l'organizzazione di tale registro, nonché i nomi dei suoi singoli elementi. È formato innanzitutto da due componenti fondamentali:

- 5 bit che devono essere programmati per consentire o meno il verificarsi di un interrupt.
- 3 bit che possono essere "letti" per controllare il tipo di evento verificato.

Abilitazione o meno di un interrupt

Sono cinque i *flag* o bit che il programmatore può impostare a piacere per programmare e gestire il tipo di evento che si vuol considerare. Sono bit da "settare" preventivamente, prima del verificarsi di un interrupt. In particolare:

- **RBIE** (bit 3): se tale bit è impostato a 1 l'interrupt che gestisce l'evento esterno sulle porte RB4, RB5, RB6 e RB7 viene abilitato. Viceversa, anche al cambiamento di stato logico su queste quattro porte, non accadrà la rilevazione da parte del microcontrollore.
- **INTE** (bit 4): se tale bit è impostato a 1, l'interrupt che gestisce l'evento esterno sulla porta RB0/INT viene abilitato. Viceversa, anche al cambiamento di stato logico su questa porta, non accadrà nulla.
  Si può decidere se l'interrupt deve essere invocato durante il fronte di discesa o di salita, agendo sul flag **INTEDG** appartenente al registro **OPTION_REG** (bit 6).
- **T0IE** (bit 5): se tale bit è impostato a 1 l'interrupt che gestisce l'evento interno di "trabboccamento" del registro TMR0 è abilitato.
- **EEIE** (bit 6): se tale bit è impostato a 1 l'interrupt che gestisce l'evento interno che segnala la corretta fine scrittura su Eeprom è abilitato.
- **GIE** (bit 7): se tale bit è impostato a 1, abilita TUTTI gli altri quattro interrupt. È dunque un flag generale che si comporta da interruttore principale.

Flag che determinano l'evento accaduto

Gli altri tre bit (FLAG) sono riservati al controllo del tipo di interrupt effettivamente chiamato. Il loro cambio di stato avviene dopo il verificarsi di un evento, e ciò consente di determinare se e quale routine di input è stata coinvolta durante l'esecuzione del programma. Essi sono:

- **RBIF** (bit 0): se questo flag viene a trovarsi al valore logico 1, vuol dire che è avvenuta una modifica dello stato digitale alle porte esterne RB4, RB5, RB6 o RB7.
- **INTF** (bit 1): se questo flag viene a trovarsi al valore logico 1, vuol dire che la porta RB0/INT ha modificato il proprio stato logico.
- **T0IF** (bit 2): se questo *flag* viene a trovarsi al valore logico 1, il time interno TMR0 ha rag-

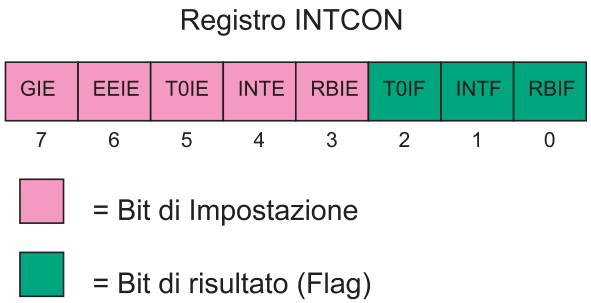


Figura 2 Il registro INTCON

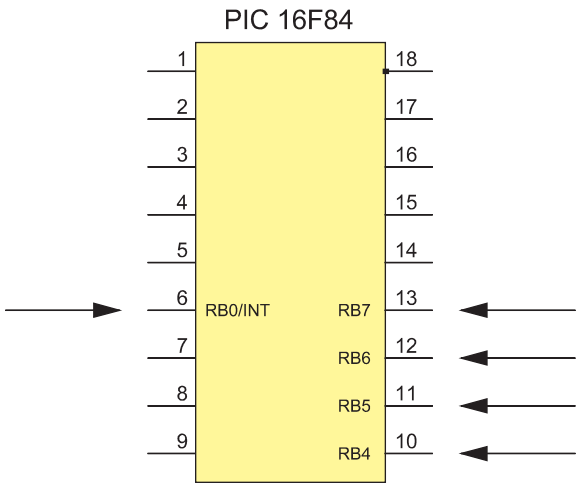


Figura 3 Le porte preposte ad un Interrupt (eventi esterni)

giunto il limite massimo e superato il valore 255.
- Se infine la routine di interrupt è stata invocata ma i tre flag si trovano a livello logico basso, evidentemente è stata conclusa correttamente la procedura di scrittura su Eeprom.

QUANDO È UTILE USARE GLI INTERRUPT?

Teoricamente, un programma potrebbe fare a meno della gestione di un interrupt. Infatti organizzando il software in modo che esso controlli continuamente (e velocemente) gli eventi interni ed esterni (polling) si raggiungerebbe lo stesso scopo. Il prezzo da pagare è però una maggior difficoltà nella stesura del codice con il conseguente incremento della sua lunghezza.
Un altro problema legato al non utilizzo degli Interrupt sta nel fatto che una determinata routine potrebbe subire un drastico rallentamento durante la rilevazione di un evento esterno. Nel proseguo dell'articolo approfondiremo proprio questo fattore. L'implementazione di una gestione ottimizzata di un interrupt, in genere, può essere effettuata in tutte quelle applicazioni con tempi di attesa molto alti, con la presenza di routine di ritardo che "congelano" l'esecuzione del programma. Ebbene, sono proprio questi i casi in cui si possono utilizzare con successo le interruzioni. Con degli esempi pratici si chiarirà meglio il concetto.

COSA OFFRE IL MIKROBASIC

Sebbene il Mikrobasic sia un linguaggio ad alto livello, esso prevede l'utilizzo degli interrupt. In effetti l'uso delle interruzioni dovrebbe essere una prerogativa dei linguaggi a basso livello, quale l'Assembler o, al massimo, il linguaggio C. Da questo punto di vista infatti si potrebbe pensare ad una gestione anomala. In ogni caso la presenza di questa possibilità permette al progettista di espandere notevolmente i campi di applicazione delle proprie creazioni, aumentando di molto la flessibilità offerta da queste ultime.
Il nostro compilatore offre una **procedura** che, invocata automaticamente al verificarsi di un segnale d'interrupt, permette di eseguire una *routine* in risposta ad un evento esterno o interno. La procedura, in tutto simile a quelle utente (UDF e UDP) deve chiamarsi obbligatoriamente "**interrupt**". Vediamo adesso le caratteristiche che deve possedere il sorgente basic per prevedere l'utilizzo delle interruzioni.
Il programma inizia SEMPRE con la parola chiave "*program*", che sancisce l'inizio dell'esecuzione dei comandi. Segue poi la procedura chiamata in caso di interrupt, di esecuzione probabilmente incerta. Essa inizia con il comando "*sub procedure*" e termina col comando "*end sub*". Infine segue il programma vero e proprio, delimitato dalla label "*main*" e dal comando "*end.*". Questa è l'organizzazione di principio da adottare in un programma Mikrobasic.
Ricordate sempre di utilizzare il nome corretto della procedura. Inoltre la "label" **main** in questo caso è obbligatoria in quanto costituisce una sorta di punto d'entrata del programma principale.

```
program interrupt_demo

sub procedure interrupt
     . . .
     . . .
     codice da eseguire in caso di interrupt
     . . .
     . . .
end sub

main:
     . . .
     . . .
     programma principale
     . . .
     . . .
end.
```

LE FASI NECESSARIE

Vediamo adesso quale deve essere l'approccio, per meglio gestire un segnale d'interrupt nel proprio software.
Occorre infatti eseguire alcune fasi preliminari per la gestione ottimale delle interruzioni.
Ecco a grandi linee le fasi principali da seguire, nel programma principale, per gestire un interrupt:
- Configurare (eventualmente) come input i pin della PORT-B sottoposti a segnale d'attivazione.
- Abilitare il corrispondente flag per scegliere il tipo di interrupt da trattare (tra RBIE, INTE, T0IE e EEIE).

- Abilitare il flag generale GIE.

D'altro canto, nella procedura d'interrupt, occorre:

- Azzerare, dopo averlo processato, il flag di RBIF, INTF o T0IF, dal momento che esso non si resetta automaticamente.
- Gestire la routine.
- Uscire con il comando "**End Sub**". Concretamente esso crea, a basso livello, l'istruzione Assembler **RETFIE**, che consente di "riagganciare" la routine chiamante e ripristinare automaticamente lo stato del registro GIE.

Nota: durante l'esecuzione di una routine di interrupt, il registro GIE è automaticamente azzerato, per evitare di creare una sorta di "nidificazione" di altre interruzioni, ovvero una chiamata ricorsiva che potrebbe riempire lo *stack* e bloccare il sistema.

PRIMO ESEMPIO: SEMAFORO SEMI-AUTOMATICO

Come primo esempio proponiamo la realizzazione di un semaforo particolare. Il lavoro sequenziale dell'illuminazione delle luci avviene in maniera del tutto automatica, ma esso può essere, per così dire, integrato, secondo le esigenze di illuminazione locale.

Caratteristiche del prototipo

Il semaforo (teorico) del nostro esempio possiede le seguenti caratteristiche di temporizzazione:

- Luce verde accesa per 50 secondi.
- Luce gialla (col verde) accesa per 10 secondi.
- Luce rossa accesa per 50 secondi.

La sequenza è, naturalmente, a ciclo infinito. Il problema sarebbe facilmente risolvibile se non fosse per un'ulteriore caratteristica richiesta nella progettazione del prototipo: la presenza di un controllo manuale.

Il controllo manuale

Il vigile urbano, che si trova in prossimità del semaforo, dispone di un controllo (pulsante o relè automatico) che permette di "accendere" direttamente una potente luce per illuminare

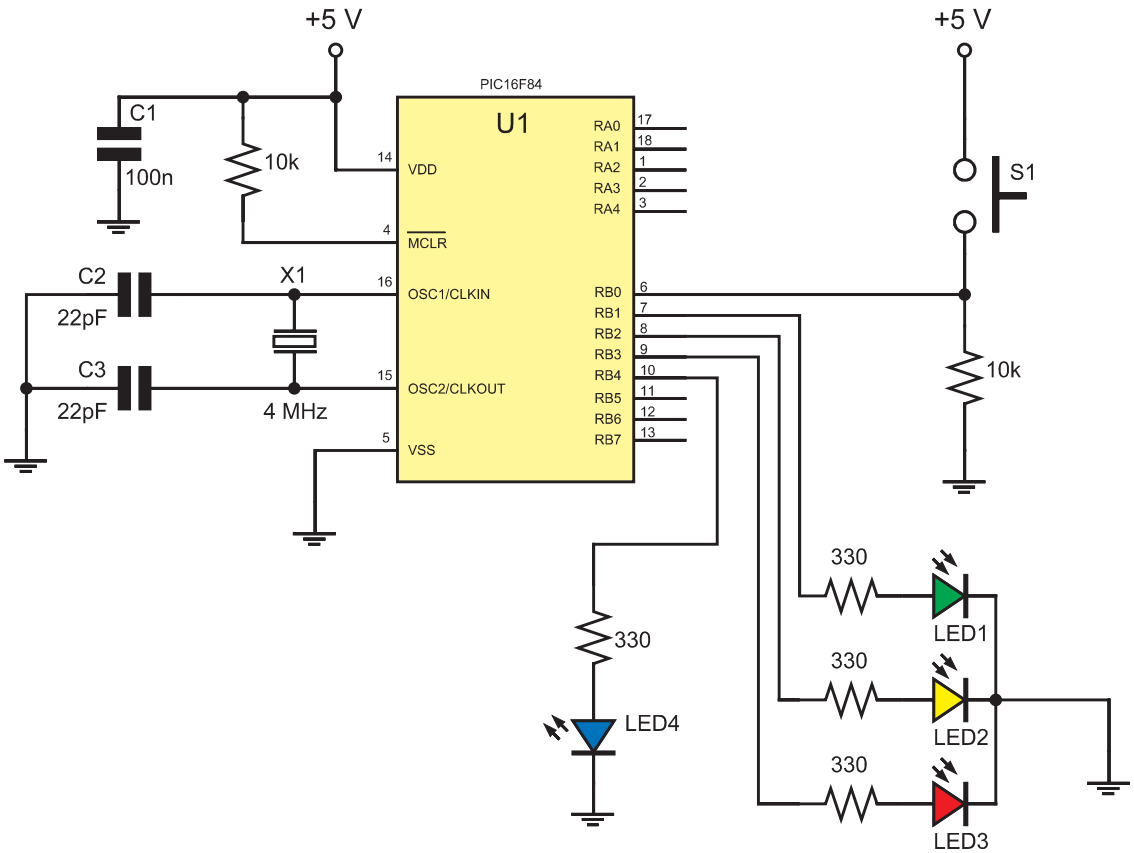


Figura 4 Schema elettrico del semaforo

l'incrocio stradale. Tale controllo può essere attivato **in qualsiasi momento**, sia a semaforo rosso, giallo o verde e, soprattutto, senza aspettare la fine di un ciclo di attesa (cambio del colore) che, nel nostro esempio, risulta alquanto prolungato (50 secondi).

Se non si utilizza l'interrupt

Se l'algoritmo non prevedesse l'uso degli interrupt, il controllo funzionerebbe male, senza una procedura attentamente valutata e prodotta. Infatti, la maggior parte del programma "itera" attorno a stati di attesa molto lunghi, che "congelano" il microcontrollore e non gli consentono di operare in altre attività. Il comando "*delay_ms*" infatti blocca l'esecuzione del programma e toglie la possibilità di eseguire altre istruzioni. Non ci troviamo infatti dinanzi ad un sistema *multi-tasking*!
In questi casi, l'unica alternativa è quella di controllare ciclicamente e molto velocemente l'ingresso, stressandolo, con una tecnica di rilevazione, denominata *polling*. Se non si ha effettivamente questa esigenza, la strategia non è consigliata.

Se si utilizza l'interrupt

Grazie all'uso dell'interrupt è invece possibile semplificare drasticamente le procedure, a patto però che si lavori "a regola d'arte". Basta infatti un "niente" per far saltare l'applicazione e bloccarla nel punto più disparato. Si ricordi che il controllo di un'applicazione e relativo *debug* è un'operazione tutt'altro che semplice e necessita di strumentazione hardware molto sofisticata, come analizzatori e *tracker* digitali.

Schema elettrico

In figura 4 è rappresentato lo schema elettrico "base" del semaforo. Si noti l'esistenza dei componenti che permettono il funzionamento del microcontrollore, quali i condensatori ed il cristallo da 4 Mhz, nonché la resistenza di *pull-up* che fissa a livello logico alto il piedino 4 di reset.
I collegamenti con il mondo esterno sono rappresentati da:

- RB0 che funziona da input, essendo collegato al pulsante. Proprio su questa porta è effettuato il controllo di interrupt.
- RB1 che funziona da output, ed è collegato al diodo led verde attraverso la resistenza limitatrice.
- RB2 che funziona da output, ed è collegato al diodo led giallo attraverso la resistenza limitatrice.
- RB3 che funziona da output, ed è collegato al diodo led rosso attraverso la resistenza limitatrice.
- RB4 che funziona da output, ed è collegato al diodo led blu ad alta luminosità.

Il programma

Il Listato 1 riporta il primo programma applicativo da trascrivere sull'editor del Mikrobasic. Commentiamo, come sempre, le parti salienti, invitando il lettore a comprendere bene i metodi e gli effetti dell'algoritmo.
Osserviamo innanzitutto la procedura riportata subito dopo la clausola *program*. Essa, racchiusa tra gli statement *sub procedure* e *end sub* contiene una parte di codice che è eseguito se avviene una chiamata di interrupt. In esso si verifica

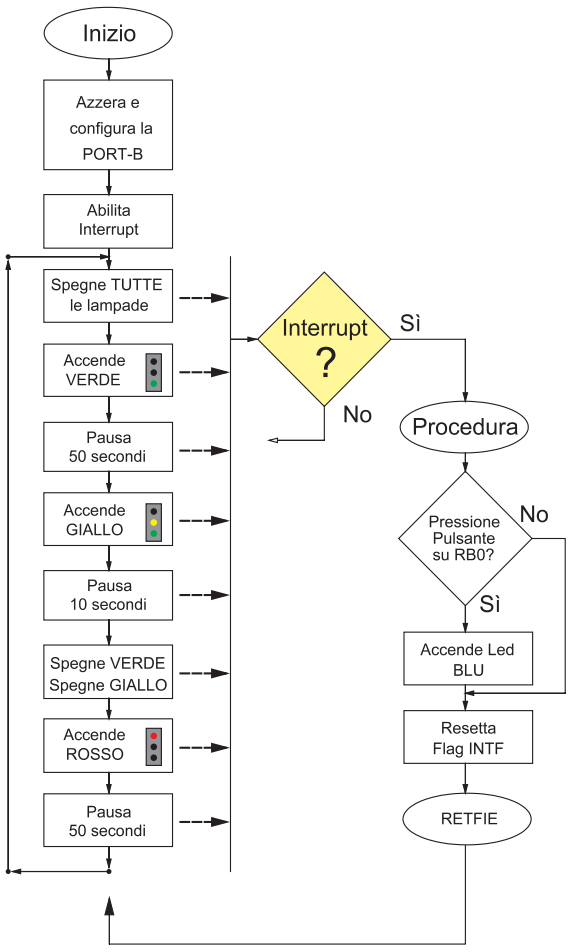


Figura 5 Il Flow-chart del semaforo

l'accensione del led blu, collegato alla porta RB4, mediante attivazione a livello logico alto del relativo piedino. Tale evento, oltre a verificarsi naturalmente in presenza di segnale di interrupt, avviene solo se il bit 1 del registro INTCON è a livello alto, in altre parole se un segnale ha coinvolto la porta RB0/INT del PIC.
Continuando, incontriamo la label *main*, obbligatoria in questo caso perché stabilisce il punto di ingresso del programma vero e proprio. Se viene omessa, il compilatore non completerà la creazione del file eseguibile, generando addirittura un messaggio d'errore.
Seguono l'azzeramento e la definizione della funzione della PORTB. Il valore binario %00000001 (equivalente decimale 1) configura tale porta in modo che il bit 0 sia trattato come ingresso, mentre i rimanenti come uscita.

[Listato 1]

```
program Semaforo

sub procedure interrupt 'Inizio PROCEDURA INTERRUPT
 if testbit(INTCON,1) then 'Se si verifica l'interrupt
 portb.4=1 'Accende il led BLU
 end if
 clearbit(INTCON,1) 'Ripristina il flag di controllo
end sub

main:
portb=0 'Azzera PORTB
trisb=%00000001 'Configura pin 0 come ingresso
 'e tutti gli altri come uscita
setbit(INTCON,4) 'Pone a 1 il bit 4 di INTCON (INTE)
 'abilitando l'evento su RB0
setbit(INTCON,7) 'Pone a 1 il bit 7 di INTCON (GIE)
 'abilitando la rilevazione generale

while true 'Programma a ciclo INFINITO
 portb.1=0 'Spegnimento iniziale VERDE
 portb.2=0 'Spegnimento iniziale GIALLO
 portb.3=0 'Spegnimento iniziale ROSSO
 portb.1=1 'Accende la luce VERDE
 delay_ms(50000) 'Attesa di 50 secondi
 portb.2=1 'Accende ANCHE la luce GIALLA
 delay_ms(10000) 'Attesa di 10 secondi
 portb.1=0 'Spegne la luce verde
 portb.2=0 'Spegne la luce gialla
 portb.3=1 'Accende la luce ROSSA
 delay_ms(50000) 'Attesa di 50 secondi
wend

end.
```

Analizziamo le istruzioni che gestiscono gli interrupt. La prima, **setbit(INTCON,4)**, pone a 1 il bit 4 del registro INTCON, abilitando in questo modo la "sola" rilevazione sulla RB0/INT. La seconda, **setbit(INTCON,7)**, assicura il controllo complessivo degli interrupt, abilitandone il registro generale (GIE). Segue infine, all'interno del ciclo infinito *while / wend*, la sequenza di istruzioni che determinano l'accensione e lo spegnimento dei led del semaforo. Il lettore noti come, in questa fase, **non esista alcun controllo** sullo stato della porta RB0/INT come ingresso, essendo questo demandato alla gestione degli interrupt. Adottando questo primo metodo, abbiamo utilizzato alcune funzioni e procedure che operano a livello di bit. Esse sono:

- *Setbit(byte,posizione)* che pone a 1 il bit specificato.
- *Clearbit(byte,posizione)* che pone a 0 il bit specificato.
- *Testbit(byte,posizione)* che controlla lo stato del bit specificato.

Per maggiori informazioni si consulti il manuale on-line (F1) del Mikrobasic.

SECONDO ESEMPIO: DUE PULSANTI E DUE LED

Come secondo esempio esaminiamo un prototipo dotato di due pulsanti normalmente aperti e due diodi led. Naturalmente si tratta di un esempio teorico che il lettore potrà applicare ad eventuali modelli pratici, con o senza modifiche.

Analisi della procedura

Il primo diodo led (verde) lampeggia al ritmo di 1 secondo, mentre inizialmente il secondo led (rosso) è spento. Se l'utente preme il primo pulsante, il led rosso si illumina. Viceversa, se preme il secondo pulsante, il led rosso si spegne. Il primo led continua "imperterrito" a lampeggiare, come se nulla fosse suc-

cesso. Grazie all'utilizzo dell'interrupt, la pressione dei tasti è "processata" in tempo reale, senza aspettare la fine dei tempi morti e, soprattutto, senza causare un sostanziale rallentamento alla frequenza di lampeggio del primo diodo led.

Schema elettrico

In figura 7 è riportato lo schema elettrico del prototipo. Come si può vedere, le porte RB0 e RB1 sono collegate ad altrettanti diodi Led, pertanto fungono da Output. Le porte RB4 e RB5 sono invece collegate a due pulsanti n/a. Il loro funzionamento è pertanto in ingresso. In stato di riposo, tali porte si trovano al potenziale di 0 Volt (livello logico basso), grazie alle resistenze di *pull-down* da 10K (valore non critico). Se il pulsante viene premuto, il relativo livello sarà immediatamente alto.

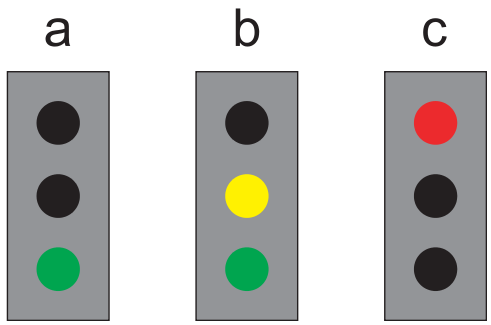


Figura 6 Le tre fasi operative del semaforo

Il programma CON GLI Interrupt

Esaminiamo il relativo software, che utilizza al meglio la gestione degli interrupt, prodotto nel listato 2. Guardiamo il codice che segue la *label* "main". Come si può vedere, attraverso il comando TRISB, le due porte RB4 e RB5 sono configurate quali ingressi, le restanti invece come uscite. Inoltre i bit 3 e 7 del registro INTCON sono posti ad uno logico, permettendo la rilevazione dell'interrupt, in caso di evento su RB4, RB5, RB6 o RB7. Il ciclo infinito seguente si occupa SOLAMENTE di accendere e spegnere il led verde alla cadenza di un secondo. Nulla più!

Se si verifica un interrupt, l'esecuzione del codice passa immediatamente alla procedura *interrupt*. Il primo controllo è quello di verificare se l'interruzione è stata determinata da un cambiamento di stato degli ultimi 4 bit della PORTB (test sul bit 0 del registro INTCON). Se tale condizione ha avuto esito positivo, viene verificato quale pulsante ha determinato la variazione, attraverso il controllo vero e proprio della PORTB. A tale verifica segue l'accensione o spegnimento del diodo led rosso.

È importante sottolineare che il test di interrupt

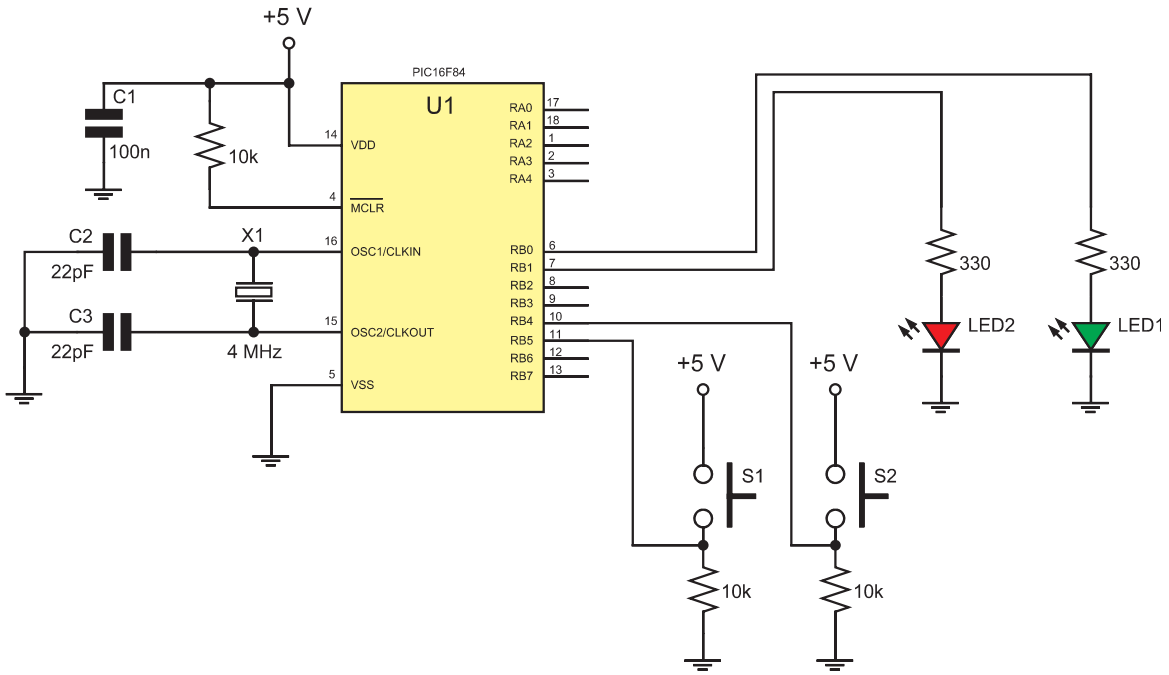


Figura 7 Schema elettrico del semaforo

[Listato 2]

```
.program Led

sub procedure interrupt 'Inizio PROCEDURA INTERRUPT
 if testbit(INTCON,0) then 'Se si verifica l'interrupt
 if portb.4=1 then 'Se premuto primo pulsante
 portb.1=1 'Accende Led Rosso
 end if
 if portb.5=1 then 'Se premuto secondo pulsante
 portb.1=0 'Spegne Led Rosso
 end if
 end if
 clearbit(INTCON,0) 'Ripristina il flag di controllo
end sub

main:

portb=0 'Azzera PORTB
trisb=%00110000 'Configura pin 4-5 come ingresso
 'e tutti gli altri come uscita
setbit(INTCON,3) 'Pone a 1 il bit n. 3 di INTCON (RBIE)
 'abilitando l'evento su RB4-RB5-RB6-RB7
setbit(INTCON,7) 'Pone a 1 il bit n. 7 di INTCON (GIE)
 'abilitando la rilevazione generale

while true 'Programma a ciclo INFINITO
 portb.0=1 'Accende la luce VERDE
 delay_ms(1000) 'Attesa di 1 secondo
 portb.0=0 'Spegne la luce VERDE
 delay_ms(1000) 'Attesa di 1 secondo
wend

end.
```

può essere effettuato anche molte volte in un intero ciclo, anche se il programma risulta "congelato" da una istruzione di attesa (delay_ms).
È altresì importante ribadire che il controllo dei pulsanti non influenza minimamente la frequenza di lampeggio del diodo verde.

Il programma SENZA Interrupt

È interessante adesso esaminare come si può risolvere lo stesso problema evitando l'uso degli interrupt, per poter paragonare le due soluzioni. Ciò comporta una lieve modifica del listato sorgente, in modo che il controllo sui pulsanti sia eseguito molte volte in un piccolo arco di tempo.
Il listato 3 mostra il programma sorgente. Commentiamo i punti salienti.
Innanzitutto si nota la totale assenza di procedure e istruzioni per la configurazione e la gestione della routine di interrupt.
Il controllo dei pulsanti è infatti relegato ad un ciclo di verifica (polling) che "testa" continuamente lo stato delle porte di input.
È stato necessario naturalmente "duplicare" questa verifica, poiché gli stati del primo led (verde) sono due, rispettivamente di acceso e spento. Inoltre la pausa di un secondo è stata "frazionata" ripetendo per mille volte una attesa di 1 millisecondo, tempo relativamente breve nel quale il micro può tranquillamente dedicarsi ad altri compiti.
In definitiva il programma risulta leggermente più complesso del precedente.

CONSIGLI

Anche se l'utilizzo delle routine di interrupt semplifica enormemente la gestione del programma e, di conseguenza, la lunghezza del codice prodotto, esso risulta il più delle volte critico e di difficile implementazione. Può accadere infatti che l'intera procedura non sia ben progettata e gestita, ed il programma finale possa incorrere facilmente in blocco (crash).
Con la programmazione tradizionale questo non accade MAI! Vediamo quali sono le regole da seguire per la miglior gestione delle chiamate da interrupt:

- In un programma è bene limitare al massimo l'uso degli interrupt; se la lunghezza di codice è la stessa, meglio preferire la normale codifica di controllo (polling).
- La routine di interrupt, e quindi il codice presente nella relativa procedura, deve risultare **la più corta possibile**; non gestite pertanto lunghi ritardi e procedure tortuose, ma semplicemente un codice compatto ed ottimizzato.
- In un microprocessore (CPU tipo Intel o Amd) esistono centinaia di chiamate di interrupt; sono infatti molto più complesse

di quelle di un microcontrollore, che ne gestisce un numero limitato.

- Con il Mikrobasic il programmatore deve solamente effettuare alcune semplici operazioni (come abbiamo visto) per gestire ed organizzare una interruzione.
  Programmando in Assembler egli deve invece preoccuparsi di configurare, salvare e predisporre tutti i registri coinvolti e non, determinando in questo modo una difficoltà enorme nella codifica e implementazione.
- L'interrupt è una prerogativa **hardware**, quindi sicuramente efficiente e affidabile, se ben utilizzata.
- State molto attenti a gestire gli interrupt con il Mikrobasic in quanto, come linguaggio ad alto livello, espande le procedure e non ottimizza il codice come se fosse scritto manualmente in Assembler.
- Un interrupt si può verificare in qualsiasi momento, per questo vi è un'alta criticità nel controllo e nel *debug* del software prodotto.
- Grazie agli interrupt possono essere tenuti costantemente sotto controllo gli eventi esterni in maniera automatica.
- In qualche caso potrebbe accadere che, con il Mikrobasic, un interrupt non risponda immediatamente all'appello, proprio perché abbiamo dinanzi un compilatore di linguaggio ad alto livello.
- Probabilmente la gestione degli interrupt risulta una delle più delicate nella programmazione.
- Gli altri tipi di interrupt non esaminati nell'articolo funzionano esattamente come quelli visti. Occorre solo modificare i controlli e gli azzeramenti dei flag.
- È possibile gestire più interrupt contemporaneamente, eseguendo controlli multipli sui rispettivi flag.
- Durante il processo di una interruzione, si può abilitare l'esecuzione di altri interrupt. È possibile infatti "nidificare" fino a otto interruzioni simultaneamente, ma ciò comporta un riempimento dello *stack* e la creazione di un codice "altamente" critico.

[Listato 3]

```
program Led

dim k as integre 'Conteggia i millisecondi

portb=0 'Azzera PORTB
trisb=%00110000 'Configura pin 4-5 come ingresso
 'e tutti gli altri come uscita

while true 'Programma a ciclo INFINITO
 portb.0=1 'Accende la luce VERDE
 for k=1 to 1000
 delay_ms(1) 'Attesa di 1 millisecondo
 if portb.4=1 then 'Se premuto primo pulsante
 portb.1=1 'Accende Led Rosso
 end if
 if portb.5=1 then 'Se premuto secondo pulsante
 portb.1=0 'Spegne Led Rosso
 end if
 next k

 portb.0=0 'Spegne la luce VERDE
 for k=1 to 1000
 delay_ms(1) 'Attesa di 1 millisecondo
 if portb.4=1 then 'Se premuto primo pulsante
 portb.1=1 'Accende Led Rosso
 end if
 if portb.5=1 then 'Se premuto secondo pulsante
 portb.1=0 'Spegne Led Rosso
 end if
 next k
wend

end.
```

CONCLUSIONI

Purtroppo lo spazio è sempre tiranno. Ci sarebbero ancora tantissime cose da spiegare sulla gestione degli interrupt, come lo Sleep, il Watch Dog e altre gestioni interne molto utili, vi rimandiamo quindi ad un prossimo articolo.
In ogni caso è bene che il lettore familiarizzi con le procedure citate nell'articolo, in modo che egli continui a districarsi nella problematica dei microcontrollori, in special modo di questi piccoli grandi componenti.

Codice MIP255076

E il PIC disse

In questo articolo apprenderemo le tecniche per far "parlare" il microcontrollore PIC, senza utilizzare sintetizzatori vocali. Sebbene la sua funzione non sia quella di elaborare audio digitale, i risultati che otterremo saranno sorprendenti.

Questo mese vedremo come riprodurre i campioni digitali per emettere, niente di meno, una parola. Sino ad ora abbiamo fatto svolgere al nostro povero microcontrollore i più disparati compiti, anche faticosi; gli mancava solo la parola...

IL METODO DA SEGUIRE

Certo, far pronunciare delle frasi al nostro microcontrollore non è proprio cosa facile. La fase più critica è quella della memorizzazione di tutti i campioni digitali, tenendo conto che lo spazio disponibile non è poi così generoso. In ogni caso le informazioni accumulate all'interno del chip devono essere alla fine "riprodotte" sulla PORT-B del micro, costituente un DAC nel nostro caso, ad una velocità e una cadenza opportunamente calcolati. Questi impulsi generati attraverso un altoparlante, producono un suono udibile, dalla forma d'onda non molto regolare ed imprevedibile, anzi molto complessa.



Il metodo utilizzato prevede due fasi distinte:

1. La raccolta delle tensioni analogiche del segnale sonoro e loro conversione in un valore corrispondente digitale, ossia una tecnica che prende il nome di campionamento (ADC). Questa operazione sarà svolta "manualmente";
2. La riproduzione, da parte del microcontrollore, di tutti i dati digitali, tramite un DAC, al fine di ricostruire il messaggio iniziale.

IL CAMPIONAMENTO

È l'operazione mediante la quale, dal segnale originario analogico, si prendono "alcuni" valori che esso assume durante alcuni intervalli di tempo costante. La distanza temporale tra un campione ed un altro si dice "periodo di campionamento", mentre il suo reciproco è la "frequenza di campionamento", ossia il numero di campioni valutati durante un secondo.
Come ben si capisce, il metodo è a perdita di informazione, poiché nell'operazione, alcuni elementi sono sacrificati per agevolare, in qualche modo, la velocità, lo spazio in memoria, il costo sostenuto e la semplicità operativa.
Naturalmente esistono delle regole alle quali affidarsi per l'esecuzione di un ottimo campionamento. Regole che, purtroppo nel nostro esempio, non possiamo rispettare per la mancanza di velocità e di memoria del nostro microcontrollore.
Una regola da considerare afferma che la minima frequenza di campionamento, idonea a ricostruire fedelmente il segnale originario, deve essere almeno pari al doppio della frequenza più alta in esso contenuta. Se questa non viene

“mamma”



di Giovanni Di Maria

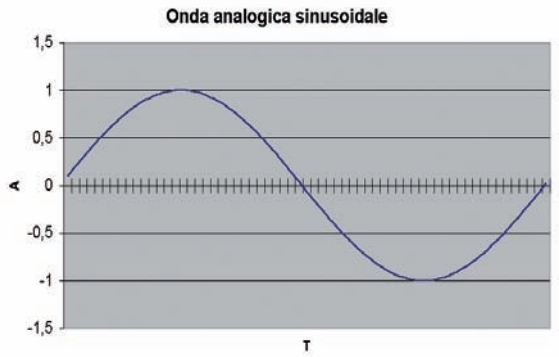


Figura 1 *Segnale analogico sinusoidale puro*

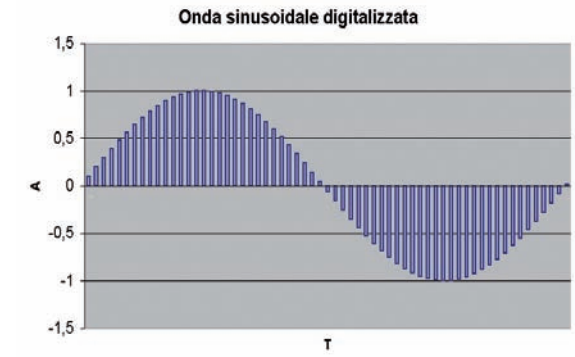


Figura 2 *Segnale digitale sinusoidale con un numero molto elevato di campioni*

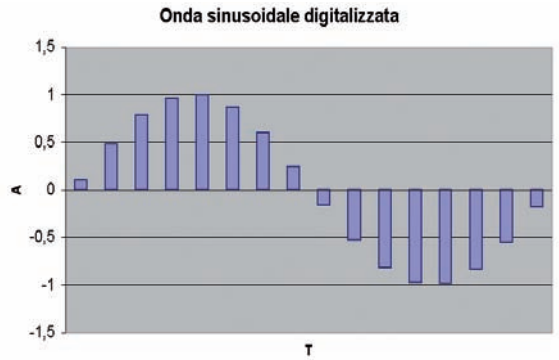


Figura 3 *Segnale digitale sinusoidale con un numero basso di campioni*

rispettata, il segnale digitale subirà non indifferenti perdite dovute ad “ammanchi” di informazioni, e più in generale forti distorsioni. Questa “tosatura” viene definita aliasing. Le figure 1, 2 e 3 mostrano i diversi tipi di segnale con la relativa qualità di campionamento.

Solitamente la frequenza di campionamento da seguire, al fine di ottenere ottimi risultati, è la seguente:

- Frequenza di 44.000 Hz, per riproduzioni con qualità eccellente;
- Frequenza di 22.000 Hz per riproduzioni musicali di buona qualità;
- Frequenza di 8.000 Hz per riproduzione con qualità telefonica.

Purtroppo, data l’esigua capacità del microcontrollore utilizzato, la frequenza da noi impiegata sarà pari a 3.000 Hz, ossia ai limiti della comparsa dell’effetto di *Aliasing*. Ma considerando i mezzi a nostra disposizione, il risultato sarà più che accettabile.

DOVE MEMORIZZIAMO LA VOCE?

Le informazioni sulla voce registrata comprendono centinaia di dati da memorizzare. Ma dove è possibile immagazzinare tante informazioni, se il 16F84 ha una RAM di soli 68 byte e poco di più i modelli superiori? Il compilatore Mikrobasic consente di “allocare” nella memoria Flash di programma, sfruttandola al meglio, le nostre informazioni, attraverso il comando *const*, come vedremo in seguito. In questa maniera, il software ottiene il massimo rendimento, in maniera impropria ma perfettamente funzionale, utilizzando la memoria “programma” del Pic.

LIMITI DELLA PROCEDURA

Per l’esperimento utilizzeremo naturalmente il compilatore Mikrobasic.
Purtroppo l’unico limite è imposto dallo spazio limitato della memoria. Infatti una parola, anche se molto breve, occupa tantissime celle del micro. Pertanto, se un PIC16F84 potremmo forse far pronunciare un “SI”, risultati migliori si potranno ottenere adoperando un 16F628 o, meglio ancora un 16F876. A tale scopo occorre consultare la tabella 1, per un riferimento quantitativo.

Microcontrollore	Numero dei Campioni
PIC16F84	Circa 950
PIC16F628	Circa 1900
PIC16F876	Circa 7500

Tabella 1 Il numero dei campioni ad 8 bit memorizzabili

I FERRI DEL MESTIERE

Per raggiungere lo scopo, abbiamo inoltre bisogno di alcuni strumenti software e hardware, secondo la lista seguente:

- Personal Computer dotato di scheda audio (qualsiasi tipo).
- Microfono di qualunque tipo.
- Software Shareware "GOLDWAVE" per l'elaborazione sonora (o equivalente). Nell'esempio è stata usata la versione 5.08.
- Un programma di foglio elettronico.
- Ambiente di sviluppo Mikrobasic.
- Programmatore per PIC e relativo software.

Sono tutti elementi di facile reperibilità. Per quanto riguarda il foglio elettronico si può optare per il potente programma freeware OpenOffice o altri software commerciali...

SCHEMA ELETTRICO

Siamo pronti per la realizzazione dello schema. Le caratteristiche consentono di ottenere risultati lusinghieri. Esse si possono così riassumere:

- Tipologia: DAC a resistori pesati.
- Numero di ingressi digitali: 8.
- Numero di uscite analogiche: 1.
- Numero massimo di tensioni ottenibili: 256.
- Range di copertura: da 0 Volt a 5 Volt.
- Risoluzione massima: 19.6 mV.

Lo schema elettrico, visibile in figura 4, è una tipica applicazione per microcontrollore PIC, le cui porte sono configurate in uscita. I resistori pesati consentono di ricavare un qualsiasi valore analogico, nel dominio delle massime prestazioni consentite. Il microcontrollore utilizzato è il PIC16F628, utilizzante un quarzo da 4 Mhz assieme ai relativi condensatori di oscillazione da 22 pF. Esso consente di riprodurre una sola parola, con una quantità media di memoria. L'unico aspetto negativo è l'utilizzo completo della PORT-B.

Come ottenere i valori delle resistenze

Come si vede dallo schema elettrico, i valori scelti delle resistenze non sono commerciali e pertanto non si trovano sul mercato. È possibile però raggiungere perfettamente la grandezza desiderata, collegando opportunamente alcune resistenze tra loro. La tabella 2 mostra come operare in tal senso. Se proprio volete utilizzare una sola resistenza per pin, indirizzate la scelta verso i seguenti valori, in ohm: 1.000, 2.200, 4.700, 10.000, 22.000, 47.000, 100.000, 220.000. Tenete conto però che con tale soluzione i valori in uscita non risultano esatti ed affidabili.

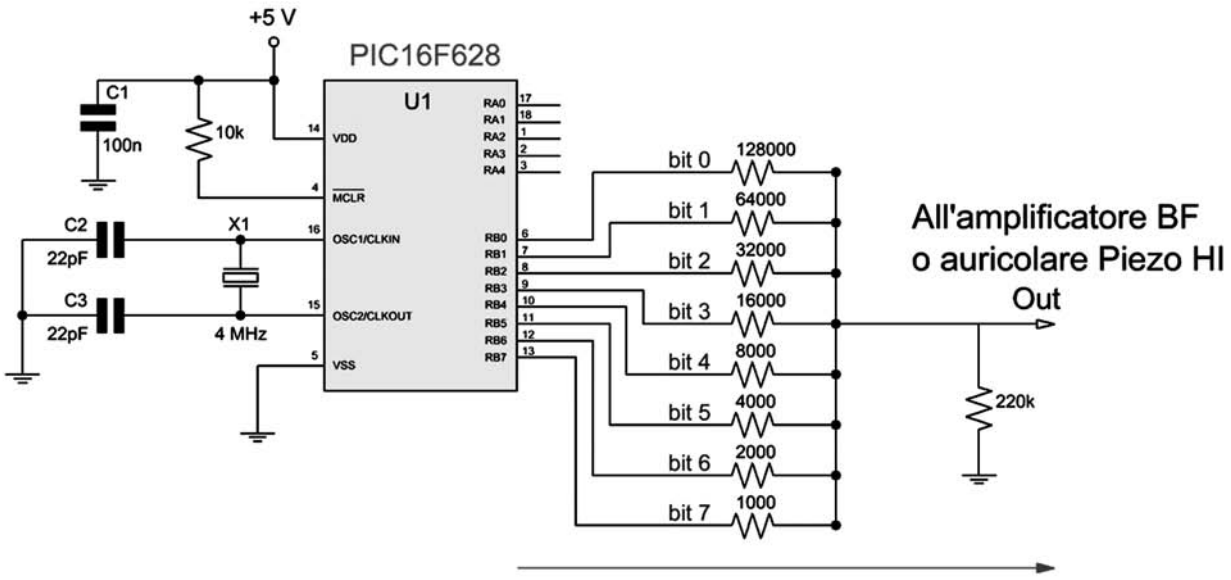


Figura 4 Schema elettrico del DAC a 8 bit

Valore da ottenere	Come ottenere il valore resistivo
1.000	Questo valore è reperibile sul mercato
2.000	Collegare due resistenze in serie da 1.000 ohm
4.000	Collegare due resistenze in serie da 1.800 ohm e 2.200 ohm
8.000	Collegare due resistenze in serie da 3.300 ohm e 4.700 ohm
16.000	Collegare due resistenze in serie da 1.000 ohm e 15.000 ohm
32.000	Collegare due resistenze in serie da 10.000 ohm e 22.000 ohm
64.000	Collegare tre resistenze in serie da 15.000 ohm, 22.000 ohm e 27.000 ohm
128.000	Collegare tre resistenze in serie da 33.000 ohm, 39.000 ohm e 56.000 ohm

Tabella 2 *Come ottenere i valori resistivi della rete pesata*

PRIMA FASE: REGISTRAZIONE DELLA VOCE

Iniziamo dunque la procedura occupandoci della registrazione del segnale sonoro. Questa operazione non serve tanto per "fissare" indelebilmente la voce sul supporto di massa, quanto per raccogliere i dati numerici corrispondenti alle varie ampiezze del segnale periodico.
Per lo scopo abbiamo utilizzato il programma shareware "Goldwave", ottimo per l'analisi e l'elaborazione del suono. Una volta approntato il microfono ed impostato correttamente il mixer del sistema operativo, possiamo avviare il software sonoro e creare un nuovo file, naturalmente selezionando il menù File e la voce New. Ovviamente il nostro campionamento deve avvenire a "bassa risoluzione" per motivi di memoria del PIC, per cui scegliamo una traccia Mono (non Stereo), avente un Sampling Rate di 3000 (purtroppo è un valore molto basso, ma non possiamo fare di più) ed una durata di 10 secondi. Con un po' di destrezza si può arrivare a registrare anche a 6 Khz, con un'ottima resa sonora. La figura 5 mostra le opzioni da impostare.

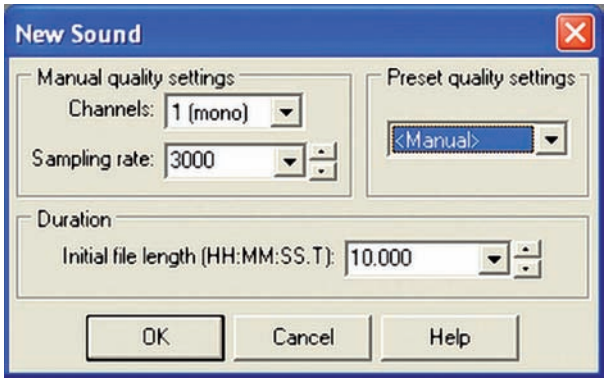


Figura 5 *Le caratteristiche del file sonoro*

Confermando la scelta, appare una traccia completamente vuota, che attende la registrazione della nostra voce. A tale scopo occorre premere il tasto circolare rosso (REC) per avviare la registrazione. Occorre recitare velocemente la parola "MAMMA". Alla fine bisogna interrompere la registrazione con il pulsante quadrato rosso (STOP). Finalmente la traccia audio contiene la voce registrata, come visibile in figura 6.
Riascoltando la traccia con il pulsante Play, si può notare come la voce, in effetti, somigli un po' meno a quella reale. Questo perché i parametri di campionamento sono stati scelti deliberatamente molto bassi. Non fatevi tentare ad alzare la qualità, il microcontrollore non potrebbe contenere i dati. Come si vede, è presente una quantità esagerata di silenzio, prima e dopo la voce, che aumenta lo spazio occupato in memoria. Occorre, a tal scopo, "tagliare" le informazioni indesiderate lasciando solamente la voce. Questa operazione viene definita "Trimming" e si effettua con la selezione della sola onda sonora utile con la successiva pressione dei tasti <CTRL> <T>. In tal modo si prende solamente la voce e si elimina tutto il resto. Se si desidera si può salvare il file per una successiva elaborazione.

SECONDA FASE: MEMORIZZAZIONE DEI CAMPIONI IN FORMATO NUMERICO

La seconda fase permette di ottenere i "singoli" campioni come dato numerico. In pratica il programma Goldwave consente di esportare la forma d'onda ottenuta in un formato numerico Ascii. Per eseguire l'operazione occorre accedere al menù File e selezionare la voce *Save As* (Salva con nome).

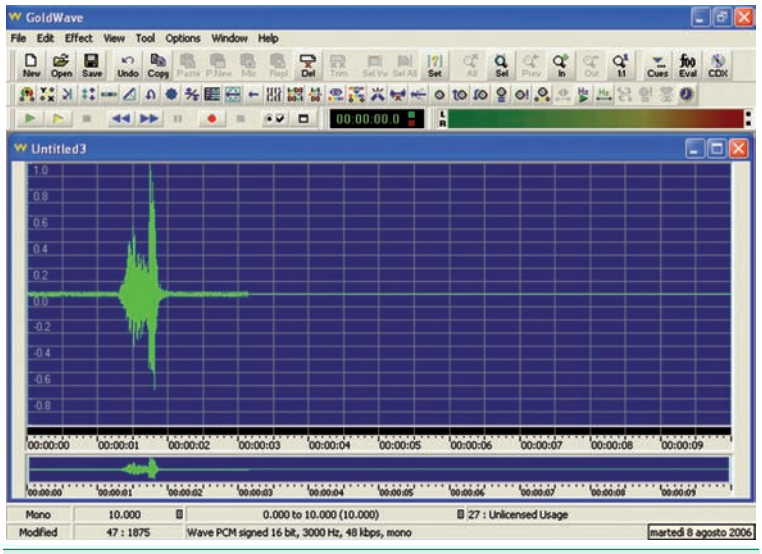


Figura 6 La traccia sonora registrata

Prima di confermare la memorizzazione occorre scegliere, come formato di output, il tipo di file *Numerical Text* e come attributo *ASCII integer mono*. La figura 7 mostra esattamente la procedura da seguire. Dopo aver premuto il pulsante Salva, verrà creato un file di testo elencante tutti i campioni utilizzati nella traccia audio.

Esaminiamo la parte iniziale del file di testo prodotto. La prima riga costituisce un'intestazione che riporta, tra l'altro, la frequenza di campionamento (3000 Hz) e, soprattutto, il numero di campioni memorizzati, nel nostro caso ben 1374 (solo per dire la parola "Mamma").

Consultando la tabella 1 possiamo notare che l'intera traccia sonora non può essere contenuta nel PIC16F84. Dalla seconda riga in poi sono presenti, uno dopo l'altro, tutti i campioni, in formato numerico *integer* (doppio byte) e

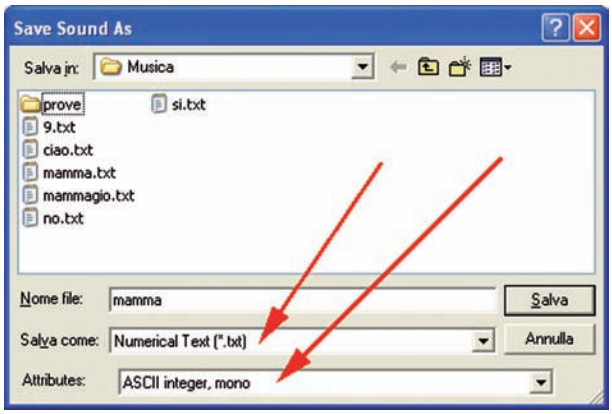


Figura 7 Come salvare i campioni numerici della traccia

signed. Vedremo nel prossimo paragrafo come effettuare la conversione.

TERZA FASE: TRASPORTO DEI CAMPIONI SU FOGLIO ELETTRONICO

La successiva fase serve per riportare, su un programma di foglio elettronico, tutti i campioni per la loro elaborazione. È un'operazione molto semplice: occorre selezionare tutti i valori (dalla seconda riga in poi) del file di testo, ed effettuare un *Copia* e *Incolla* sul foglio elettronico. Fate attenzione al fatto che i campioni non sono solamente quelli che si vedono a video, ma si trovano anche in fondo al documento.

QUARTA FASE: TRASFORMAZIONE POSITIVA DEI CAMPIONI

Dal momento che i campioni trattati ed esportati sono del tipo signed integer (intero con segno), il dominio relativo è compreso tra i valori 32768 e –32767. Occorre pertanto "traslare" i campioni in un range positivo, semplicemente aggiungendo il valore 32767. In questa maniera, il nuovo intervallo dei valori sarà compreso tra 0 e 65535. A tale scopo si deve creare, nel foglio elettronico, una seconda colonna che preveda una semplice addizione ai valori della prima colonna. In ogni caso riportiamo la formula, da inserire nella cella B1, e da ricopiare in tutte le altre celle sottostanti:

=A1+32767

QUINTA FASE: ADEGUAMENTO AL RANGE 0-255

Quest'altra fase serve per adattare i valori della seconda colonna a quelli compatibili all'uso con il microcontrollore, di tipo byte. Occorre, allo scopo, creare sul foglio elettronico una terza colonna che preveda una proporzione, proprio per restringere il campo dei valori ai limiti 0-255. La formula da inserire nella cella C1, e poi su tutte le altre sottostanti, è la seguente:

=B1*255/65535

Tale formula, naturalmente, fornirà dei risultati decimali.

SESTA FASE: TRONCAMENTO PARTE DECIMALE

Dal momento che i campioni devono essere memorizzati in formato byte all'interno del microcontrollore, occorre eliminare la parte decimale con un ulteriore formula del foglio elettronico. La quale provvede anche ad arrotondare eventualmente per eccesso i valori. La formula per troncare la parte decimale, da inserire nella cella D1 e successive, è la seguente:

=INT(C1+0,5)

SETTIMA FASE: GRAFICO DI PROVA DEL FOGLIO ELETTRONICO

Per effettuare la verifica dei dati elaborati, si selezionino tutti i valori contenuti nella colonna D del foglio elettronico e si crei un grafico "a linee". Dovrebbe apparire la rappresentazione di cui in figura 8. Con quest'ultima operazione abbiamo effettuato la raccolta dei campioni e loro adeguamento al corretto formato digitale. Senza l'ausilio del foglio elettronico, avremmo dovuto operare manualmente (parliamo di 1374 campioni)!

OTTAVA FASE: FINALMENTE IL PROGRAMMA

Commentiamo velocemente il programma, che potete scaricare dal sito di Fare elettronica e che per problemi di spazio non abbiamo potuto riportare interamente.
Il vettore "campione" contiene ben 1374 valori numerici, numerati progressivamente da 0 a 1373. Essi vengono "pescati" grazie all'indice k, di tipo word poiché si supera il valore di 255. Il ciclo for-next si occupa proprio di questo lavoro. La distanza temporale tra un campione e il successivo è di 333 microsecondi, poiché la frequenza di campionamento è di 3 KHz (1/3000). La durata dell'intera traccia è pertanto 333uS x 1374 = 0,457 secondi.
Alla fine della riproduzione, dopo una pausa di "riposo" di un secondo, l'intero processo inizia nuovamente e si ripete all'infinito.

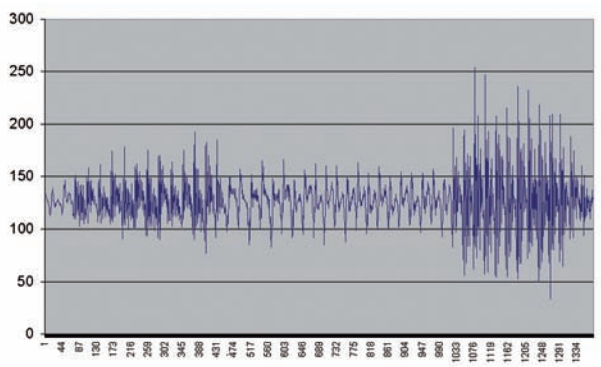


Figura 8 Il grafico di verifica dei dati del foglio elettronico

```
rem Il Pic dice MAMMA
program mamma

const campione as byte[1374]= (
133,134,132,131, ...
(n.d.r. il programma completo lo potete scaricare
dal sito di Fare Elettronica)
... 125,137,125,131,129 )

dim k as word 'Indice per i campioni
portb=0 'Azzera portb
trisb=0 'definisce PORTB in output

while true 'Ciclo infinito
for k=0 to 1373
portb=campione[k] 'Legge/riproduce i campioni
delay_us(333) 'Attesa
next k
delay_ms(1000) 'Aspetta un secondo
wend
end.
```

ESERCIZIO PER I LETTORI

Provate a realizzare un prototipo, dotato di due pulsanti normalmente aperti. Se si preme il primo pulsante, si sentirà la parola SI, se si preme il secondo pulsante, naturalmente si sentirà un secco NO. Qualunque tipo di microcontrollore è ammesso. Potete mandare le risposte direttamente all'autore.

CONCLUSIONI

Il segnale prodotto nell'altoparlante dell'amplificatore audio, o nell'auricolare ad alta impedenza, è affetto da una leggera percentuale di Aliasing. Del resto con una digitalizzazione a 3 KHz non possiamo pretendere di più. In ogni caso il messaggio riprodotto è perfettamente chiaro ed intelligibile. Usando questa tecnica è possibile far riprodurre al fedele PIC qualsiasi parola. Risultati migliori, in termini di lunghezza e qualità si potranno ottenere utilizzando il modello 16F876. Nel prossimo articolo analizzeremo la tecnica che ci ha permesso la realizzazione di questo progetto, vedremo come "generare forme d'onda con il PIC".

Codice MIP256044

Generare forme d'onda

Nello scorso numero abbiamo visto come riprodurre un file audio con il PIC. In questo numero analizzeremo le tecniche utilizzate per questa particolarissima applicazione, vedremo come generare una qualsiasi forma d'onda dalla eccellente qualità armonica.

La generazione di un segnale variabile nel tempo, avente una qualsiasi forma d'onda, è un'operazione puramente analogica. Come può un microcontrollore eseguire questa operazione, dal momento che esso può solo trattare segnali di 0V e 5V? Scopriamolo assieme leggendo questo articolo.

L'ONDA PERIODICA

È un'onda che si ripete nel tempo con una certa regolarità. Nel nostro studio verranno esaminate le onde i cui punti operativi sono compresi in un dominio digitale positivo (0 Volt e 5 Volt) e non quelli passanti per lo zero (corrente alternata). Ai fini dell'analisi il risultato prodotto è comunque equivalente.

IL DAC

Per poter riprodurre i vari punti appartenenti ad un'onda periodica, occorre "generare" varie tensioni analogiche, corrispondenti all'andamento dell'ampiezza della stessa. Il microcontrollore utilizzato deve pertanto, in qualche modo, produrre delle differenze di potenziale, in maniera sequenziale e a grande velocità. Esistono tante modalità per raggiungere lo scopo. Una di queste è rappresentata dalla tecnica del PWM, ampiamente documentata nella rivista Firmware del mese di Ottobre. Il metodo che utilizzeremo in queste pagine consente di ottenere risultati migliori ed è conosciuto con il nome di "Convertitore D/A a resistori pesati".

CONVERTITORE D/A CON RESISTORI PESATI

È ovviamente un convertitore che consente di ottenere una grandezza analogica, partendo da una digitale. È composto da una rete di resistori collegati ad altrettante porte digitali. La risoluzione della conversione dipende dal numero di resistenze utilizzate. Il valore di ogni resistenza deve risultare il doppio della precedente, in modo da realizzare dei "pesi" ed ottenere dei valori sicuramente non riproponibili.
Applicando all'ingresso della rete una serie di valori digitali (0V – 5V) ai vari nodi è possibile ottenere, in uscita, una tensione analogica compresa tra i due estremi previsti. Questo avviene grazie alla sovrapposizione degli effetti. Esaminiamo i vantaggi e gli svantaggi del metodo, mettendo in evidenza i punti principali.

Vantaggi

In generale, i vantaggi che si possono ottenere utilizzando il metodo proposto sono i seguenti:

- La risoluzione del DAC dipende dal numero di resistori utilizzati, calcolati come potenza in

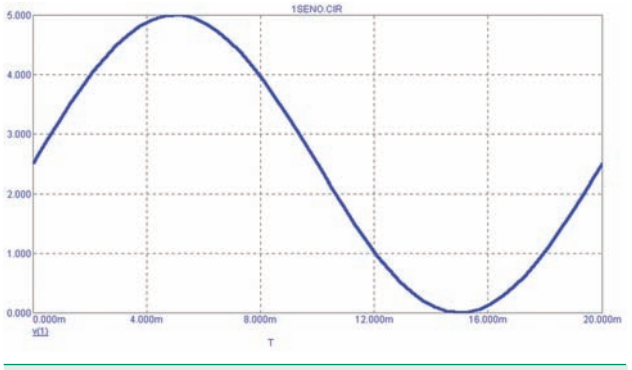


Figura 1 Tipica onda sinusoidale positiva



di Giovanni Di Maria

base 2. Il numero massimo pertanto di tensioni ottenibili è dato dalla formula 2^n dove n rappresenta il numero dei resistori impiegati. Con otto resistenze possono essere prodotte ben 256 differenze di potenziale diverse (DAC a 8 bit).

- Dal momento che nel circuito non esistono elementi capacitivi, la velocità di elaborazione e di produzione è molto alta. Sono assenti infatti effetti transitori e costanti di tempo che rallentino l'intero processo. Una tensione pertanto viene prodotta immediatamente.
- Pesando opportunamente i valori digitali sugli ingressi si ottiene "in tempo reale" la tensione analogica corrispondente, praticamente impossibile utilizzando la tecnica del PWM.
- Teoricamente non occorre collegare all'uscita alcun filtro, in quanto non esiste una componente ad alta frequenza ed il valore di tensione è generato istantaneamente.
- Semplicità circuitale: sono sufficienti solamente alcuni resistori per la creazione della rete. L'unica condizione da rispettare è quella di adottare valori sempre "raddoppiati" per ciascuna resistenza.

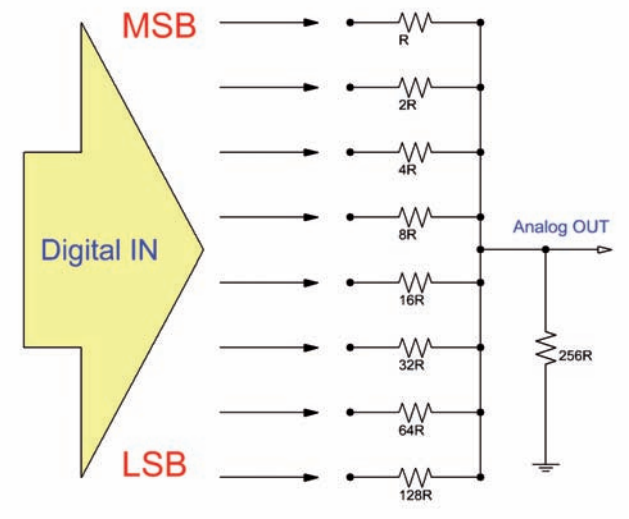


Figura 2 Schema di principio di Convertitore D/A a resistori pesati

Svantaggi

Veramente limitati sono gli svantaggi del metodo, che comunque restano ben ripagati dagli aspetti positivi che quest'ultimo può fornire:

- Occorre "sacrificare" molte uscite del microcontrollore, tante quante sono i resistori utilizzati. Più porte vengono usate, più alta è la risoluzione ottenuta.
- L'accuratezza dei risultati dipende dalla precisione delle resistenze, che devono essere scelte con molta cura e bassa tolleranza.
- Utilizzando molte porte, il valore delle resistenze si innalza velocemente, obbligando a dimensionare il carico ad impedenze molto elevate.
- La corrente erogata dalla rete è dipendente dallo stato logico dei bit di ingresso.

Comprendiamo il metodo utilizzando 3 resistenze

Per capire bene il metodo, proponiamo un esempio di DAC a 3 bit, utilizzando appunto tre resistenze. È possibile ottenere in questo caso solamente 8 valori analogici diversi (2^3) con una "risoluzione" di circa 0,713 Volt. La tensione ottenuta dipende molto dalla resistenza di "carico" presente all'uscita, che deve essere alquanto elevata. Osserviamo lo schema proposto in figura 3 nonché la tabella 1. Essa riporta i valori ottenuti sviluppando tutte le combinazioni possibili delle tensioni digitali ai relativi ingressi. Occorre notare che modificando i segnali ai vari ingressi pesati, il valore della tensione analogica in uscita cambia e, soprattutto, genera differenze di potenziale univoche.

Formule matematiche

Da quanto detto si evincono alcuni fatti che determinano la bontà del segnale generato, sotto forma di equazioni matematiche:

1. Il numero massimo di tensioni generate

dipende dal numero di resistori utilizzati, secondo la formula:

$$N_{max} = 2^{NumRs}$$

2. La risoluzione ottenuta, ossia il più piccolo valore di tensione tra un campione ed il successivo è il rapporto tra l'intervallo di tensione ed il numero di combinazioni totali, secondo la formula:

$$V_{ris} = \frac{V_{max} - V_{min}}{2^{NumRs}}$$

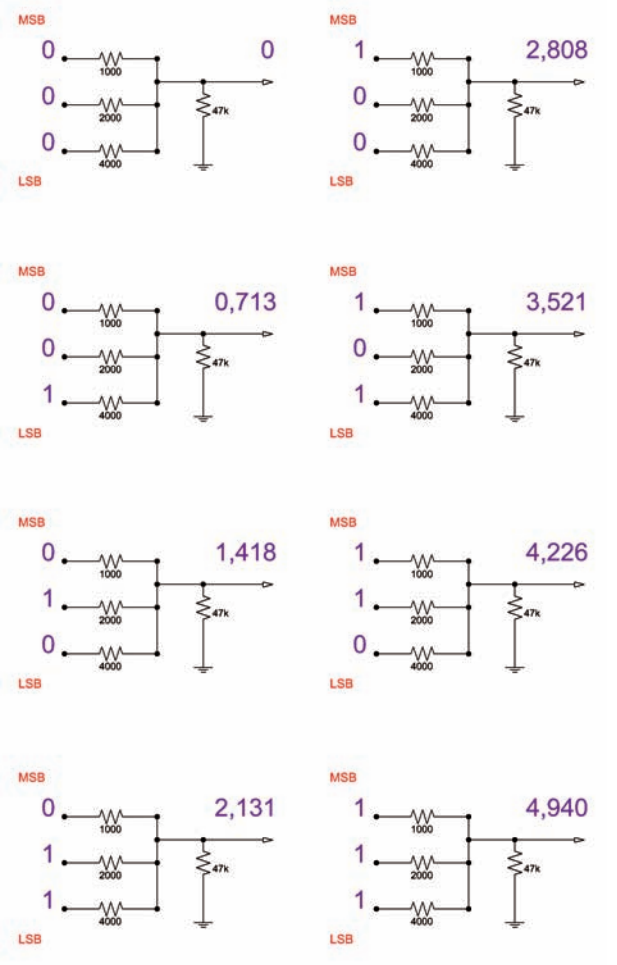


Figura 3 Schema di esempio di un DAC a 3 bit

REALIZZIAMO IL CONVERTITORE

Siamo pronti adesso per la realizzazione del convertitore. Le caratteristiche che consentono di ottenere risultati lusinghieri si possono così riassumere:

- Tipologia: DAC a resistori pesati
- Numero di ingressi digitali: 8
- Numero di uscite analogiche: 1
- Numero massimo di tensioni ottenibili: 256
- Range di copertura: da 0 Volt a 5 Volt
- Risoluzione: 19.6 mV

Schema elettrico

Lo schema elettrico, visibile in figura 4, è una tipica applicazione per microcontrollore PIC, le cui porte sono configurate in uscita. I resistori pesati consentono di ricavare un qualsiasi valore analogico, nel dominio delle massime prestazioni consentite. Il microcontrollore utilizzato è ormai l'intramontabile PIC16F84, utilizzante un quarzo da 4 Mhz assieme ai relativi condensatori di oscillazione da 22 pF.
L'unico aspetto negativo è l'occupazione dell'intera PORT-B, ma sono disponibili ancora cinque pin tramite la PORT-A. Il carico è rappresentato da una resistenza da 220k, un valore sufficientemente alto per non "caricare" troppo il segnale. Più elevato è il suo valore e minore sarà la sua l'influenza sulla tensione in uscita.

Ingresso 2 = bit 2 (su 100 ohm)	Ingresso 1 = bit 1 (su 200 ohm)	Ingresso 0 = bit 0 (su 400 ohm)	Tensione in uscita (Volt)
0	0	0	0
0	0	5	0,713
0	5	0	1,418
0	5	5	2,131
5	0	0	2,808
5	0	5	3,521
5	5	0	4,226
5	5	5	4,940

Tabella 1: Esempio di DAC a resistori pesati a 3 bit

Come ottenere i valori delle resistenze

Come si vede dallo schema elettrico, i valori scelti delle resistenze non sono commerciali e pertanto non si trovano sul mercato. È possibile però raggiungere perfettamente la grandezza desiderata, collegando opportunamente alcune resistenze tra loro. La tabella 2 mostra come operare in tal senso. Se proprio

Valore da ottenere	Come ottenere il valore resistivo
1.000	Questo valore è reperibile sul mercato
2.000	Collegare due resistenze in serie da 1.000 ohm
4.000	Collegare due resistenze in serie da 1.800 ohm e 2.200 ohm
8.000	Collegare due resistenze in serie da 3.300 ohm e 4.700 ohm
16.000	Collegare due resistenze in serie da 1.000 ohm e 15.000 ohm
32.000	Collegare due resistenze in serie da 10.000 ohm e 22.000 ohm
64.000	Collegare tre resistenze in serie da 15.000 ohm, 22.000 ohm e 27.000 ohm
128.000	Collegare tre resistenze in serie da 33.000 ohm, 39.000 ohm e 56.000 ohm

Tabella 2: *Come ottenere i valori resistivi della rete pesata*

volete utilizzare una sola resistenza per pin, indirizzate la scelta verso i seguenti valori, in ohm: 1.000, 2.200, 4.700, 10.000, 22.000, 47.000, 100.000, 220.000. Tenete conto però che con tale soluzione i valori in uscita non risultano esatti ed affidabili.

Analisi del DAC

La tabella 3 mostra tutti i valori analogici ottenibili combinando gli otto ingressi digitali. La simulazione è stata effettuata creando un firmware ad-hoc che realizzasse un conteggio sequenziale binario sulle otto porte del PIC. La figura 5 mostra invece il grafico, estremamente lineare, dell'andamento dei valori digitali ed analogici in questione. Per chi non volesse riferirsi alla tabella è possibile utilizzare la formula teorica per ricavare il valore della tensione digitale (per otto bit):

$$V_U = \frac{V_R}{2^n}\,(S_7 \cdot 2^7 + S_6 \cdot 2^6 + S_5 \cdot 2^5 + S_4 \cdot 2^4 + S_3 \cdot 2^3 + S_2 \cdot 2^2 + S_1 \cdot 2^1 + S_0 \cdot 2^0)$$

dove naturalmente:

- V_U è la tensione analogica d'uscita che si vuole ottenere;
- V_R è la tensione di alimentazione di riferimento massima;
- n è il numero di ingressi digitali utilizzati;
- S_n è lo stato digitale del bit considerato (0 oppure 1).

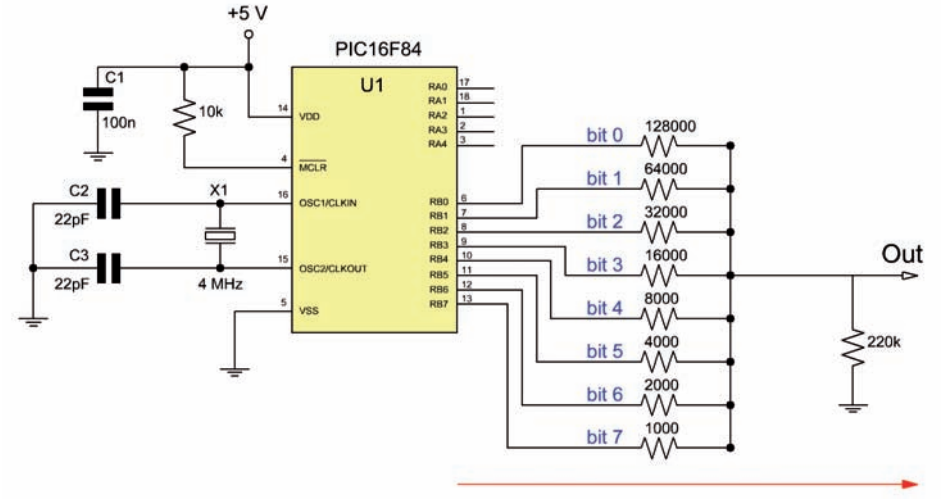


Figura 4 *Schema elettrico del DAC a 8 bit*

OTTENIAMO UNA TENSIONE ANALOGICA FISSA

Bene, siamo pronti per creare il primo software da implementare sul microcontrollore. Ricordiamo che esso è scritto in linguaggio Basic, utilizzando l'ambiente di sviluppo Mikrobasic.

Scopo del programma

Utilizzando lo schema elettrico di cui in figura 4, si deve scrivere un firmware che consenta di ottenere in uscita una tensione di circa 3,3 Volt per alimentare, ad esempio, un ulteriore MCU. È ovvio che per generare tale differenza di potenziale occorre "attivare" alcune uscite, lasciando al potenziale di massa le altre. Per stabilire quali sono le porte coinvolte, consultate la tabella 3, sino a trovare il valore analogico di circa 3,3 Volt. Ad esso corrisponde un valore digitale di 171_{10} che in base binaria corrisponde a 10101011_2. In pratica occorre considerare, come sempre, la PORT-B come un

byte, composto da 8 bit; si devono porre a livello logico altro rispettivamente i bit 0, 1, 3, 5, 7. Il software Basic eseguirà tale compito.
La figura 6 mostra in dettaglio la procedura, focalizzando le resistenze coinvolte nella "pesatura" della tensione considerata. Le linee di colore rosso indicano la presenza di tensione digitale. Il programma è riportato nel listato 1.

E ADESSO GENERIAMO LE FORME D'ONDA

Per generare una qualsiasi forma d'onda occorre riprodurre molto velocemente, nel tempo, tante tensioni di ampiezza che la caratterizzano. È ovvio che maggiore è il numero di tensioni di riferimento, maggiore sarà la qualità e la fedeltà del segnale prodotto.

ONDA SINUSOIDALE

Analizziamo la forma d'onda riprodotta in figura 7 e la tabella 4 che elenca i punti operativi di un'onda sinusoidale; queste informazioni devono essere dati "in pasto" al microcontrollore. Le caratteristiche dell'onda da riprodurre possono essere così riassunte:

- Frequenza: 50 Hz

Dig.	Out	Dig.	Out	Dig.	Out	Dig.	Out	Dig.	Out	Dig.	Out	Dig.	Out	Dig.	Out
0	0,000	32	0,626	64	1,246	96	1,871	128	2,467	160	3,093	192	3,712	224	4,338
1	0,020	33	0,646	65	1,265	97	1,891	129	2,486	161	3,112	193	3,732	225	4,358
2	0,039	34	0,665	66	1,285	98	1,911	130	2,506	162	3,132	194	3,752	226	4,377
3	0,059	35	0,685	67	1,305	99	1,930	131	2,526	163	3,152	195	3,771	227	4,397
4	0,079	36	0,704	68	1,324	100	1,950	132	2,545	164	3,171	196	3,791	228	4,417
5	0,098	37	0,724	69	1,344	101	1,970	133	2,565	165	3,191	197	3,811	229	4,436
6	0,118	38	0,744	70	1,363	102	1,989	134	2,585	166	3,210	198	3,830	230	4,456
7	0,138	39	0,763	71	1,383	103	2,009	135	2,604	167	3,230	199	3,850	231	4,476
8	0,157	40	0,783	72	1,403	104	2,029	136	2,624	168	3,250	200	3,869	232	4,495
9	0,177	41	0,803	73	1,422	105	2,048	137	2,643	169	3,269	201	3,889	233	4,515
10	0,196	42	0,822	74	1,442	106	2,068	138	2,663	170	3,289	202	3,909	234	4,535
11	0,216	43	0,842	75	1,462	107	2,087	139	2,683	171	3,309	203	3,928	235	4,554
12	0,236	44	0,862	76	1,481	108	2,107	140	2,702	172	3,328	204	3,948	236	4,574
13	0,255	45	0,881	77	1,501	109	2,127	141	2,722	173	3,348	205	3,968	237	4,593
14	0,275	46	0,900	78	1,521	110	2,146	142	2,742	174	3,368	206	3,987	238	4,613
15	0,295	47	0,920	79	1,540	111	2,166	143	2,761	175	3,387	207	4,007	239	4,633
16	0,314	48	0,940	80	1,559	112	2,185	144	2,780	176	3,406	208	4,026	240	4,652
17	0,333	49	0,959	81	1,579	113	2,205	145	2,800	177	3,426	209	4,046	241	4,672
18	0,353	50	0,979	82	1,599	114	2,224	146	2,820	178	3,446	210	4,066	242	4,691
19	0,373	51	0,999	83	1,618	115	2,244	147	2,839	179	3,465	211	4,085	243	4,711
20	0,392	52	1,018	84	1,638	116	2,264	148	2,859	180	3,485	212	4,105	244	4,730
21	0,412	53	1,038	85	1,658	117	2,283	149	2,879	181	3,505	213	4,124	245	4,750
22	0,432	54	1,057	86	1,677	118	2,303	150	2,898	182	3,524	214	4,144	246	4,770
23	0,451	55	1,077	87	1,697	119	2,323	151	2,918	183	3,544	215	4,164	247	4,789
24	0,471	56	1,097	88	1,716	120	2,342	152	2,937	184	3,563	216	4,183	248	4,809
25	0,490	57	1,116	89	1,736	121	2,362	153	2,957	185	3,583	217	4,203	249	4,829
26	0,510	58	1,136	90	1,756	122	2,382	154	2,977	186	3,603	218	4,222	250	4,848
27	0,530	59	1,156	91	1,775	123	2,401	155	2,996	187	3,622	219	4,242	251	4,868
28	0,550	60	1,175	92	1,795	124	2,421	156	3,020	188	3,642	220	4,262	252	4,888
29	0,569	61	1,195	93	1,815	125	2,440	157	3,036	189	3,662	221	4,281	253	4,907
30	0,589	62	1,215	94	1,834	126	2,460	158	3,055	190	3,681	222	4,301	254	4,927
31	0,608	63	1,234	95	1,854	127	2,480	159	3,075	191	3, 701	223	4,321	255	4,946

Tabella 3: *Tutti i valori analogici in uscita, corrispondenti agli otto ingressi digitali*

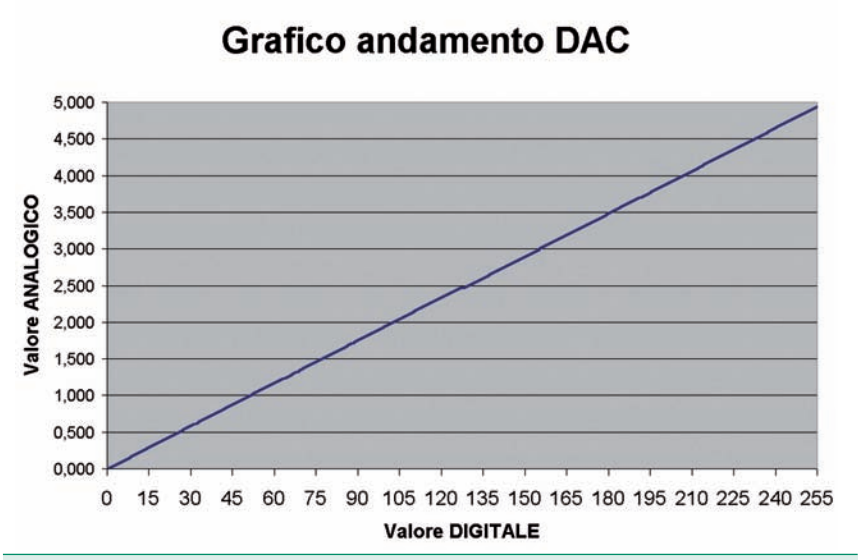


Figura 5 Grafico dell'andamento DAC

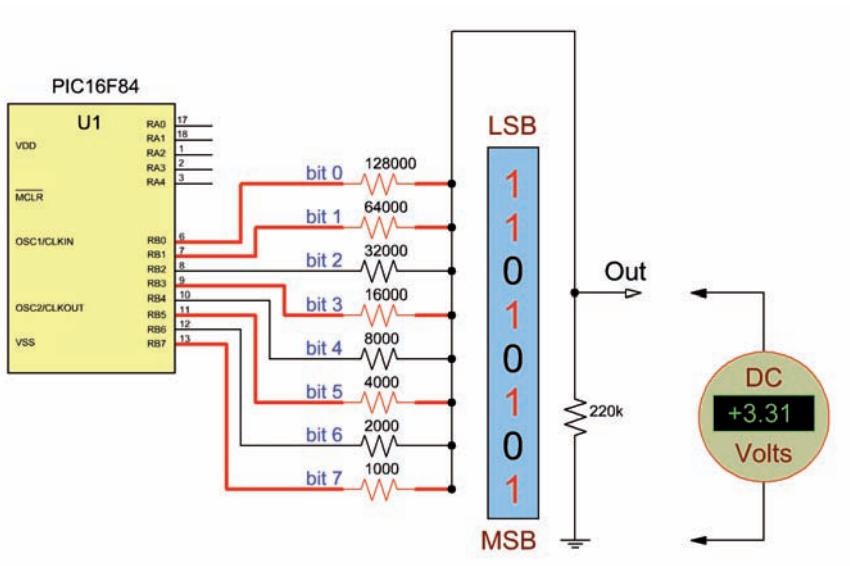


Figura 6 I resistori coinvolti per l'ottenimento dei 3,3 Volt

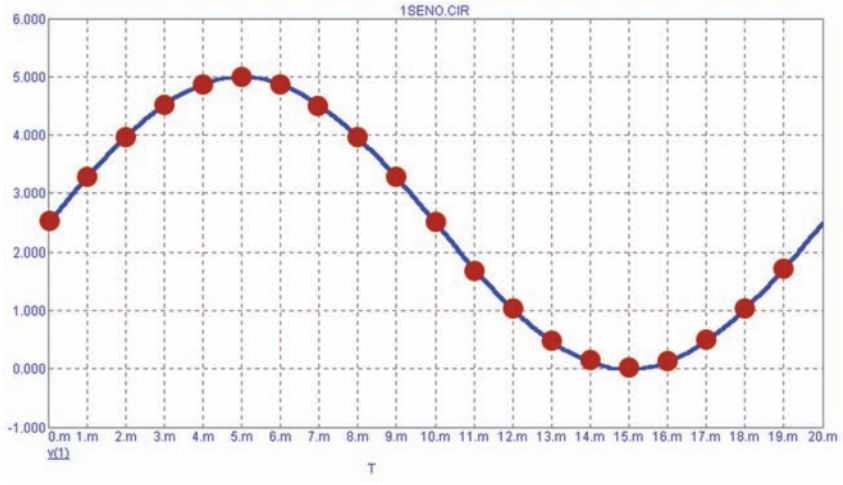


Figura 7 L'onda sinusoidale da riprodurre

- Durata periodo: 20 mS
- Ampiezza: 5 V_{0p}
- Numero di campioni: 20
- Durata temporale di un campione: 1 ms
- Percentuale ripple: assente

Metodo da seguire

Occorre, in pratica, memorizzare in un vettore i venti valori dei campioni (tabella di look-up). Quindi, all'interno di un ciclo, essi dovranno essere riprodotti, con un intervallo di tempo pari ad un millisecondo (nel nostro caso). In questa maniera la PORT-B assumerà, di volta in volta, il valore delle cifre binarie memorizzate e, grazie alla rete dei resistori pesati, il valore analogico sarà disponibile all'uscita. Basta infine ripetere il ciclo all'infinito per ottenere un'onda costante nel tempo. Utilizziamo lo schema proposto in figura 4.
Per ottenere un'onda ancora più "pulita" basta aumentare il numero dei campioni, ma già un numero di 20 rappresentano un ottimo compromesso tra qualità e spazio occupato in memoria.

Il programma

Il programma, riportato nel listato 2, è veramente molto semplice. Si noti la preparazione del vettore "campione", contenente i 20 valori relativi ai punti operativi della forma d'onda. Un ciclo infinito si occupa della lettura dei valori e relativa riproduzione sulla PORT-B del micro. Ogni campione resta "fissato" sulla porta per un tempo pari ad un millisecondo.

[Listato 1]

```
rem Utilizzo dei pesi
rem per ottenere 3,3 Volt

program dac

portb=0    'Azzera portb
trisb=0    'definisce PORTB in output
portb=171  'Imposta la PORTB con i
           'bit utili per ottenere 3,3V
           'oppure 10101011 in binario
end.
```

[Listato 2]

```
rem Generazione di una
rem onda sinusoidale

program dac
const campione as byte[20]=(130,169,205,
    233,251,255,251,233,205,169,130,
    89,53,24,6,0,6,24,53,89)
main:
dim k as byte
portb=0    'Azzera portb
trisb=0    'definisce PORTB in output
while true       'Ciclo infinito
   for k=0 to 19
       'Legge e riproduci campioni
       portb=campione[k]
       'Attesa 1 millisecondo
       delay_ms(1)
   next k
wend
end.
```

La forma d'onda ottenuta

Il risultato ottenuto è veramente degno di nota. Fortunatamente non esiste il fastidioso ripple, presente invece utilizzando la tecnica del PWM. Si nota invece un ridotto effetto a "scalino" dovuto proprio al riprodursi ciclico dei campioni, ma è un fatto normale e previsto, non un difetto. La figura 8 mostra l'onda sinusoidale ottenuta, con l'effetto a gradino. L'ampiezza

Posizione temporale del punto operativo	Relativa tensione analogica	Corrispondente valore digitale
0 ms.	2,500 Volt	130
1 ms.	3,271 Volt	169
2 ms.	3,967 Volt	205
3 ms.	4,520 Volt	233
4 ms.	4,874 Volt	251
5 ms.	4,997 Volt	255
6 ms.	4,874 Volt	251
7 ms.	4,520 Volt	233
8 ms.	3,967 Volt	205
9 ms.	3,272 Volt	169
10 ms.	2,500 Volt	130
11 ms.	1,729 Volt	89
12 ms.	1,033 Volt	53
13 ms.	0,480 Volt	24
14 ms.	0,126 Volt	6
15 ms.	0,003 Volt	0
16 ms.	0,126 Volt	6
17 ms.	0,480 Volt	24
18 ms.	1,033 Volt	53
19 ms.	1,728 Volt	89

Tabella 4: *I 20 punti operativi di un'onda sinusoidale positiva*

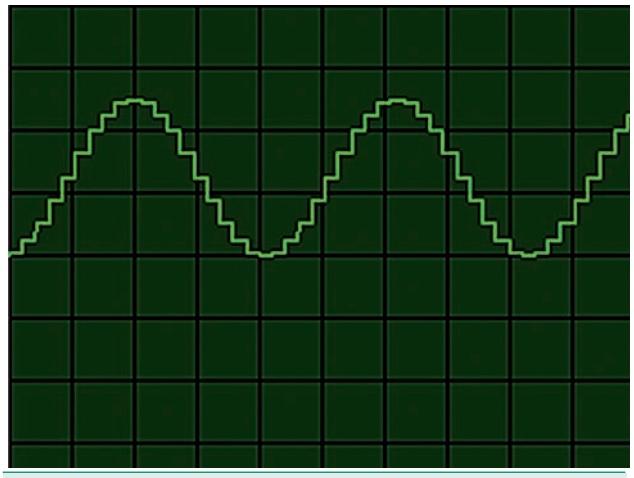


Figura 8 L'onda sinusoidale riprodotta con l'effetto a gradino

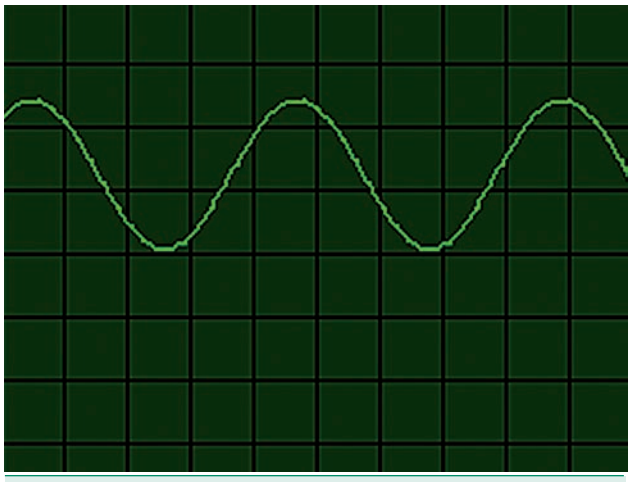


Figura 9 L'onda sinusoidale riprodotta senza l'effetto gradino grazie al condensatore

massima del segnale è di 5 Volt.
Se si vuole eliminare la caratteristica seghettata, basta inserire un condensatore in parallelo alla resistenza di uscita, per smussare perfettamente i fronti spigolosi del segnale, creando addirittura una sorta di "interpolazione" dei punti mancanti. Non è possibile definire una formula generale, per la determinazione del valore della capacità, poiché infinite possono essere le forme d'onda prodotte, pertanto la reattanza capacitiva è differente da caso a caso. Possiamo solamente dire che occorre scegliere un condensatore che non produca (assieme al resistore) una costante di tempo troppo elevata, tale da annullare il segnale prodotto. Nel nostro caso si è scelto un valore pari a 2,2 microFarad.
Si noti, in figura 9, come adesso l'onda sinusoidale sia perfetta e dall'andamento molto dolce, grazie alla presenza del condensatore elettrolitico.

ESERCIZIO PER I LETTORI

Provate a realizzare da soli un programma che consenta di ottenere le seguenti forme d'onda:

- Onda Triangolare, di frequenza pari a 80 Hz, utilizzando 35 campioni.
- Onda a dente di sega, di frequenza pari a 100 Hz, utilizzando 30 campioni.

A livello temporale, tutti i campioni devono risultare equidistanti tra loro. Potete inviare le risposte direttamente all'autore.

CONCLUSIONI

Con l'adozione del metodo proposto in queste pagine è possibile riprodurre praticamente qualsiasi forma d'onda. Basta semplicemente gestire i vari campioni e memorizzarli all'interno del micro, ricordandosi che più elevato è il numero dei campioni e più la forma d'onda è pura e pulita. Anche la frequenza relativa può essere eccezionalmente elevata, specialmente utilizzando un quarzo da 20 Mhz, in quanto non esiste il transiente per una produzione del segnale PWM. Vi consiglio di rivedere l'articolo proposto lo scorso mese dove applichiamo le tecniche qui descritte e consentiamo al PIC di pronunciare la sua prima parola…

Codice MIP257048

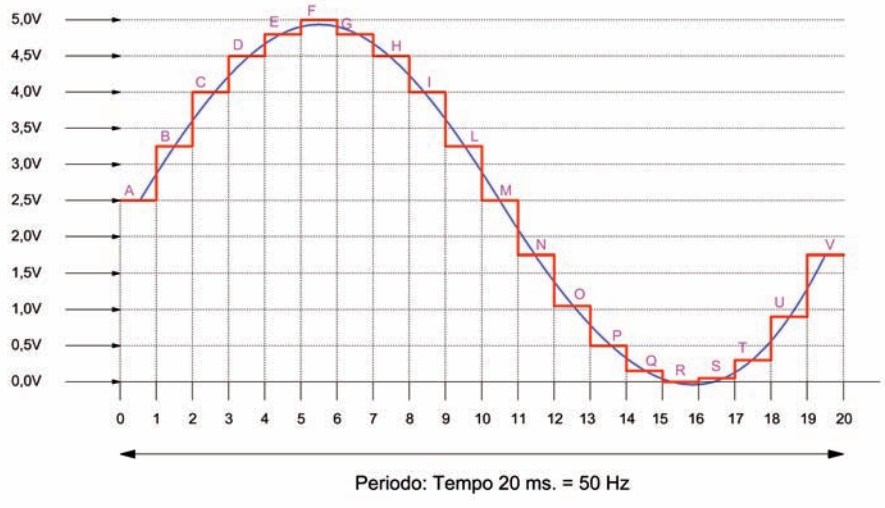


Figura 10 L'onda sinusoidale a gradini elaborata dal PIC