

**Istituto Tecnico Industriale
"M. Panetti" – Bari**

Sistemi Elettronici Automatici

Alunno : Scardicchio Sebastiano

Classe : 5[^] ETB

Docente: Prof. Ettore Panella

Generatore di funzioni



**Scardicchio Sebastiano
Docente: Ettore Panella**

a.s.2002/2003

Generatore di Funzioni con Visual Basic 6

Introduzione

Il progetto realizza un generatore di funzioni sviluppato in Visual Basic 6.

La funzione generata può essere visualizzata sul monitor o inviata, ad esempio ad un oscilloscopio, mediante interfacciamento del PC alla porta CENTRONICS.

Nella realizzazione del progetto si è posto quindi anche il problema di convertire la grandezza digitale fornita dal computer in una grandezza analogica. Questo è stato possibile mediante l'utilizzo di un convertitore digitale analogico (D/A) realizzato con DAC 0830.

Valutazione delle espressioni

Visual Basic è uno strumento incredibilmente potente, tuttavia gli manca una *feature* indispensabile per creare delle sofisticate applicazioni matematiche e scientifiche:

la capacità di valutare una espressione inserita dall'utente in runtime.

Un buon valutatore di espressione può risparmiare un bel po' di noioso lavoro di scrittura di codice, in quanto permette di parametrizzare delle parti delle applicazioni con delle soluzioni che non è possibile implementare mediante del semplice codice VB.

Per esempio, un valutatore di espressioni a runtime diventa particolarmente utile quando si utilizzano delle calcolatrici scientifiche con scopi particolari, oppure delle routine che disegnano il grafico di una funzione, risolvono delle derivate simboliche, eseguono del calcolo integrale, ed altro ancora.

Nel programma realizzato è stato utilizzato un **modulo di classe** denominato **CExpression**, che implementa un valutatore di espressioni facilmente utilizzabile in qualsiasi progetto scritto in Visual Basic. Per un corretto funzionamento si deve inserire anche il modulo di classe **Cvariable**. Per inserire un modulo in un progetto VB si deve attivare il menu **Progetto/Inserisci modulo di classe**.

Gli algoritmi su cui si basa un valutatore di espressioni sono abbastanza complessi ma non si è certo costretti a comprendere appieno il meccanismo su cui si basa la classe per poterla usare a sua volta. L'interprete del VB "compila" una espressione in p-code quando trova una espressione nel programma corrente. Il VB esegue questo passo, detto **parsing**. Il runtime del VB6 interpreta poi il p-code e valuta l'espressione molto velocemente quando il programma va in esecuzione. In altre parole, il parsing eseguito al momento della compilazione permette poi di eseguire il codice alla più alta velocità possibile. Inoltre il parsing assicura che nessuna espressione del programma contenga un errore sintattico, né dei riferimenti a variabili o funzioni inesistenti.

Questo approccio rende l'ambiente VB molto più efficiente di un semplice "interprete". Il Visual Basic esegue il processo di parsing solo una volta prima dell'esecuzione. Se l'espressione si trova in un loop critico che viene eseguito diecimila volte, eseguendo il parsing in anticipo e memorizzando il risultato il risultato in forma p-code si risparmiano 10000 passaggi nel parser a runtime.

Prima di poter scrivere un valutatore di espressioni a runtime è necessario studiare accuratamente quello che fa il VB quando esegue il parsing di una espressione.

Per esempio si consideri questa assegnazione:

$$x=5+y*4$$

Non è possibile valutare questa espressione semplicemente applicando gli operatori matematici nell'ordine in cui essi vengono incontrati dal parser. Infatti, il simbolo di moltiplicazione ha una priorità rispetto al simbolo dell'addizione. Quindi occorre valutare $(y*4)$ prima di aggiungere 5. Se poi aggiungete la possibilità di nidificare delle sotto-espressioni tra le parentesi, le cose si complicano notevolmente. Si consideri:

$$x = (5*(y+4))/6$$

In questo caso occorre prima di tutto valutare $(y+4)$, poi moltiplicare per 5 e infine dividere il risultato per 6. Le regole generali per implementare l'ordine delle operazioni sono molto semplici. Per prima cosa vanno valutate tra le coppie più interne in parentesi. Poi vanno applicati i simboli di moltiplicazione e divisione, prima delle addizioni e delle sottrazioni. Infine si valutano i simboli con la medesima priorità (ad esempio i simboli più o meno), da sinistra verso destra. Anche se queste regole funzionano, esse non contemplano tutti i possibili casi (ad esempio ci possono essere degli esponenti che possono avere priorità rispetto al simbolo di moltiplicazione e della divisione).

Né queste regole spiegano come comportarsi con le funzioni definite dall'utente o con gli operatori di stringa. Se si vuole automatizzare il processo di parsing mediante una routine di VB occorre percorrere una strada completamente diversa. Un modo conveniente di valutare una espressione consiste nell'usare uno stack per conservare gli operatori e gli operandi in sospeso. Questo modo di ragionare differisce da quello seguito dalla maggioranza delle persone alle prese con un'espressione aritmetica, ma funziona molto bene con i computer, in quanto la stessa CPU usa uno stack quando esegue dei veri calcoli matematici.

Nella pagina seguente è riportato un algoritmo per la valutazione delle espressioni algebriche.

L'algoritmo per valutare una normale espressione (notazione fissa) è decisamente complesso, anche se questo flow-chart non tiene neanche in considerazione le sotto-espressioni fra parentesi. E' da notare che ci sono due loop innestati il che ovviamente tende a rallentare l'esecuzione.

La complessità dell'algoritmo di parsing è una delle principali ragioni per le quali molti programmatori preferiscono utilizzare delle routine più semplici, meno complete o efficienti.

Per esempio, si potrebbe costruire una funzione che ricerca la sottoespressione contenuta tra la coppia di parentesi più interna, poi sostituirla con il suo risultato e continuare le sostituzioni andando verso le parentesi più esterne.

Questo approccio è facilmente realizzabile per mezzo di una routine ricorsiva. Tuttavia, l'algoritmo di parsing che verrà mostrato in questo articolo presenta un paio di vantaggi rispetto ad una più semplice routine ricorsiva.

Per prima cosa esso può valutare in modo efficiente delle espressioni di qualsiasi grado di complessità. Secondo, può generare una specie di p-code che può essere poi rielaborato per valutare nuovamente e velocemente la medesima espressione.

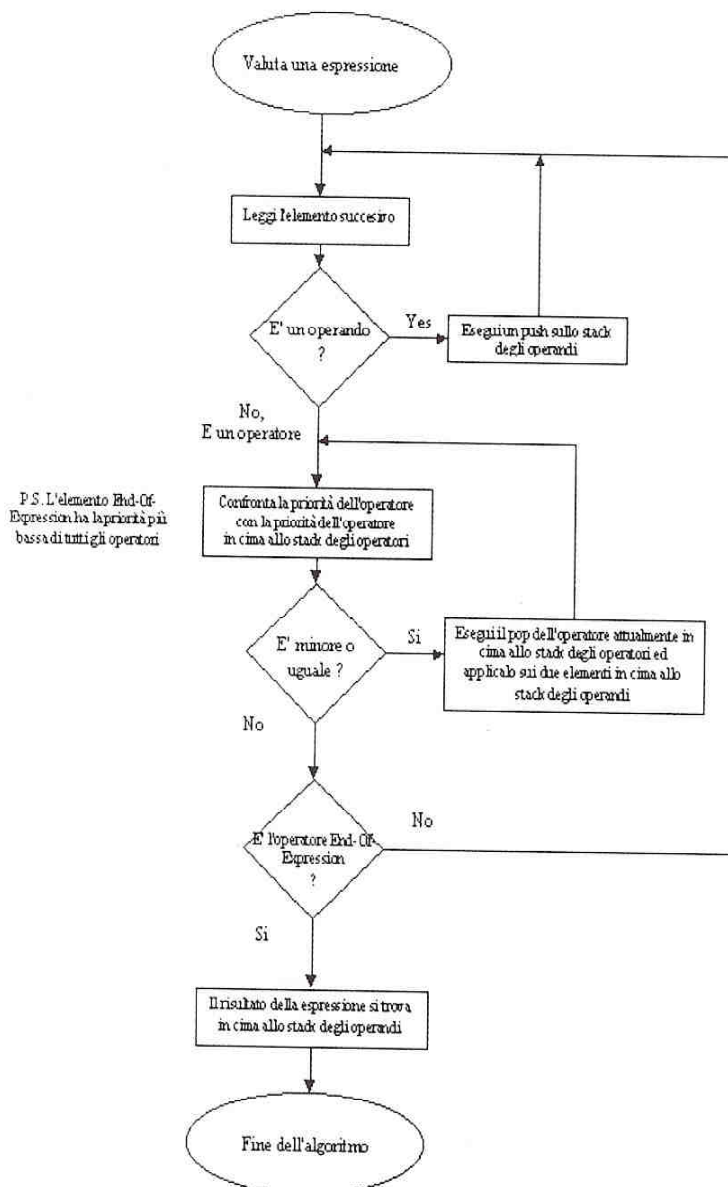
Si è deciso di differenziare il processo di parsing dal processo di valutazione del risultato, e di implementare pertanto due routine distinte. La prima routine esegue l'analisi dell'espressione inserita dall'operatore e restituisce la sua forma p-code equivalente.

La seconda routine accetta in input tale p-code e restituisce il valore numerico. Questo approccio sembra meno efficiente di quello diretto, fino a quando non ci si rende conto che è possibile realizzare la versione p-code della espressione dopo aver assegnato dei valori differenti alle variabili presenti nella espressione stessa.

In questo si è emulato in pratica il comportamento dell'interprete VB. La classe segue il parsing della espressione una volta soltanto e la può in seguito valutare più volte. Inoltre, delegando il parsing ad una routine separata si rende più semplice l'individuazione degli errori di sintassi nell'espressione prima di tentare di valutarla.

Del resto, questo è esattamente quello che fa l'editor del VB quando si inserisce una riga di codice non corretta sintatticamente.

Per ottenere questo comportamento si è modificato l'algoritmo originale per pre-processare una espressione ed ottenere il suo p-code equivalente.



Se l'elemento incontrato dal parser è un operando, la classe lo accoda immediatamente al flusso del p-code. Altrimenti confronta l'operatore incontrato con l'operatore in cima allo stack degli operatori, esattamente come accade nell'algoritmo originale.

La sola differenza è che gli operatori tolti dallo stack vengono accodati al flusso del p-code invece di essere applicati agli operandi sullo stack. Lo scopo di questa fase di parsing è di produrre un flusso di opcode che può essere poi dato in input al motore del valutatore.

La routine di parsing deve rimuovere tutte le parentesi e modificare l'ordine degli operandi e degli operatori in modo che poi arrivino al motore del valutatore nell'ordine che quest'ultimo si aspetta. E' da notare che la routine di parsing non utilizza lo stack degli operandi, in quanto gli operandi sono accodati al flusso degli opcode man mano che vengono incontrati. Al contrario, il motore del valutatore non richiede lo stack degli operatori, in quanto gli operatori sono applicati agli operandi appena vengono estratti dal flusso del p-code.

Per esempio, si applichino le seguenti linee guida ad un'espressione reale:

$$x=4+5*6-7$$

Dopo il parsing, il flusso p-code diventa:

$$4\ 5\ 6\ *\ +\ 7\ -$$

E' da notare che questa stringa utilizza la Notazione Postfissa. "Postfissa" significa che ogni operatore segue gli operandi a cui si riferisce:

$$5\ 6\ *$$

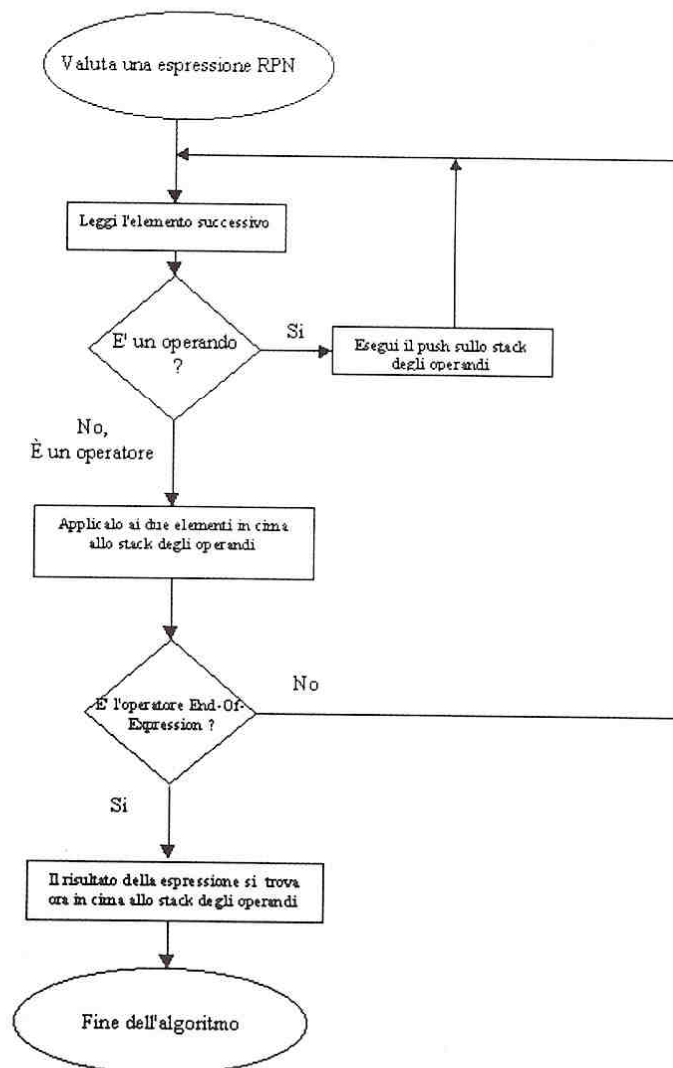
Si confronti la precedente sequenza con la normale notazione algebrica (Notazione Infissa), secondo la quale l'operatore si trova sempre tra gli operandi a cui si riferisce:

$$5*6$$

La Notazione Postfissa è anche nota come **Notazione Polacca Inversa** (o RPN, Reverse Polish Notation). Si può valutare un'espressione RPN più velocemente di una espressione infissa perché non si deve far altro che leggere l'espressione RPN da sinistra verso destra ed applicare gli operatori man mano che si incontrano. La RPN non richiede uno stack per conservare gli operatori in attesa di essere applicati, né richiede di tener conto delle parentesi o della differente priorità di ciascun operatore.

La valutazione di una espressione RPN richiede solo uno stack per gli operandi ed una semplice routine che estrae gli elementi dal flusso di p-code e li applica.

In figura è riportato il flow-chart per la valutazione delle espressioni in notazione polacca inversa.



Valutare le espressioni già convertite in notazione polacca inversa è un processo semplice e veloce. C'è solo un ciclo, e non occorre preoccuparsi delle parentesi né della priorità differente dei vari operatori.

Nelle applicazioni reali la maggior parte delle espressioni sono valutate più di una volta in un loop. Per esempio, si valuti la stessa espressione per differenti valori della variabile X (o qualsiasi variabile si trovi sull'asse delle ascisse) quando si deve disegnare il grafico di una funzione matematica.

Le applicazioni possono quindi risparmiare del tempo in ciascuna iterazione del ciclo eseguendo il parsing della espressione e producendo il corrispondente flusso di p-code, anche se questo richiede un po' di tempo per il parsing stesso.

Tralasciando il discorso della RPN, la parte più semplice dell'intero processo è l'utilizzo della classe stessa. Infatti, per utilizzarla bisogna esclusivamente creare un'istanza alla classe, assegnare l'espressione alla sua proprietà **expression** e leggere il risultato nella proprietà **value**.

Nella pagina seguente è riportata la lista completa della proprietà e dei metodi di classe. Le istruzioni per realizzare quest'operazione sono le seguenti:

```
Dim expr as New Cexpression  
Expr.expression = "4 * ( 5 + 6 )"  
Print expr.Value ` prints 44
```

Se non si ha nessun controllo sulla espressione che si sta valutando si dovrebbe anche controllare che l' espressione sia ben bilanciata e non contenga nessun errore di sintassi.

Si può far ciò testando la proprietà **ErrorCode** dopo aver assegnato una stringa alla proprietà **Expression**:

```
Dim expr As New Cexpression, x As String  
On Error Resume Next  
X= InputBox ("Enter a math expression")  
If expr.ErrorCode  
MsgBox expr.Error.ErrorDescription  
Else  
Print expr.Value  
End if
```

Bisognerebbe anche assicurarsi che la valutazione della espressione non provochi degli errori di runtime, come "divisione per 0". Si possono intercettare gli errori nelle espressioni come se si stesse valutando una espressione VBA:

```
On Error Resume Next  
Print expr.Value  
If Err Then MsgBox Err.Description
```

Si può anche decidere di trattare gli errori senza attivare esplicitamente alcun errore nel programma VB. Ciò si ottiene impostando su False la proprietà **RaiseErrors**, ed usando le proprietà esposte dalla classe Cexpression:

```
`non vi è bisogno di On Error Resume  
Next  
Expr.RaiseErrors = False  
Print expr.Value  
If expr.ErrorCode Then  
MsgBox expr.ErrorDescription  
`cancellare errore interno  
expr.ClearError  
End if
```

Name	Tipo	Descrizione
AutoCreateVariables	Proprietà (Boolean)	Se True (il valore di default) la classe crea automaticamente una o più istanze della classe CVariable, in modo che ciascuna istanza corrisponda ad una variabile trovata durante il parsing della espressione.
AddVariable (cvar As CVariable)	Function (Long)	Aggiunge un oggetto CVariable alla collection delle variabili gestite dal valutatore di espressioni. Questo metodo è particolarmente utile quando la proprietà AutoCreateVariables è False, perché in tal caso dovete creare tutte le variabili prima di assegnare una stringa alla proprietà Expression.
ClearError	Sub	Azzerata il codice di errore corrente.
DefaultValue	Proprietà (Variant)	Questo è il valore restituito come il valore della espressione quando avviene un errore di runtime, ad es. "Divisione per Zero". Questa proprietà è effettivamente usata solo se RaiseErrors è False.
ErrorCode	Proprietà Read-only (Long)	Il codice di errore restituito dalla operazione di parsing o di valutazione eseguita più di recente. Può essere uno dei seguenti valori: expOK, expSyntaxError, expUnknownFunction, expUnknownOperator, expWrongNumberOfArguments.
ErrorDescription	Proprietà Read-only (String)	La descrizione dell'errore avvenuto più di recente.
ErrorPos	Proprietà Read-only (Integer)	L'indice del carattere nella espressione originale che ha causato un errore di sintassi. Questa proprietà è utile se intendete costruire un editor interattivo di formule e espressioni.
Evaluate (expr\$, ParamArray vars())	Funzione (Variant)	Esegue il parsing e la valutazione di una espressione in un unico passo. Se l'espressione contiene delle variabili, potete passare il loro valore come argomenti opzionali. Potete usare questo metodo quando non intendete valutare ripetutamente la medesima espressione, in qual caso non beneficereste di un passo separato di compilazione.
Expression	Proprietà (String)	L'espressione che deve essere valutata. Quando si assegna una stringa a questa proprietà, l'espressione viene immediatamente analizzata, e gli eventuali errori sintattici possono essere scoperti testando le proprietà Error*.
FindRoots (lowLimit#, highLimit#, [interval#], [varName\$] [tolerance#])	Sub	Ricerca tutte le radici di una equazione, ossia tutti i valori della variabile che azzerano l'espressione. I primi due argomenti rappresentano l'intervallo di ricerca, <i>interval</i> è il passo usato per analizzare l'intervallo di ricerca, <i>varName</i> è il nome della variabile come compare nella espressione, e <i>tolerance</i> è un piccolo valore positivo che viene usato per controllare se il valore della espressione è vicino allo zero.
RaiseErrors	Proprietà (Boolean)	Se False (il valore di default), la classe attiva i suoi errori a runtime nella applicazione client usando il metodo Err.Raise, esattamente come farebbe una espressione VB standard. Se impostata al valore True, non viene attivato alcun errore nella applicazione client, e le condizioni di errore possono essere scoperte solo attraverso le proprietà Errors*.
Root(index)	Proprietà Read-only (Double)	La n-esima radice trovata dopo aver invocato il metodo FindRoots.
RootsCount	Proprietà Read-Only (Long)	Il numero delle radici trovate dopo aver eseguito il metodo FindRoots.
RPNEExpression	Proprietà Read-only (String)	La versione RPN della espressione correntemente contenuta nella istanza della classe.
Value(ParamArray vars())	Funzione (Double)	Valuta l'espressione compilata usando il valore corrente delle variabili, oppure dopo aver assegnato alle variabili i valori passati come argomenti alla funzione. Se vi sono più variabili è necessario passare loro il valore seguendo l'ordine alfabetico dei nomi delle variabili. In altre parole, passare prima il valore di X, poi di Y, poi di Z.
Variable (index varName\$, [createIfNeeded])	Funzione (restituisce un oggetto CVariable)	Restituisce la variabile con il nome specificato, oppure che compare nella collection con quel particolare indice. Se la variabile non esiste e se createIfNeeded è True, questo metodo crea automaticamente un nuovo oggetto CVariable, viceversa restituisce Nothing.
VariablesCount	Funzione (Long)	Restituisce il numero delle variabili correntemente definite.

La classe Cexpression permette anche di trattare espressioni che contengono variabili. In questo caso si deve fornire il valore di ciascuna variabile al momento di eseguire il metodo Value:

```
Dim expr As New Cexpression  
Expr.Expression = "x*4+y^2"  
Print expr.Value (5, 10) ` mostra 120
```

E' da notare che il metodo value prevede che i valori siano passati alle variabili in ordine alfabetico quando l' espressione contiene più di una singola variabile, ad esempio che si passi prima il valore da assegnare alla variabile x.

Poi il valore della variabile y e così via. Se si è sicuri che l' espressione sia sintatticamente corretta si può anche utilizzare il metodo **Evaluate** ed assegnare tutte le variabili in un solo colpo:

```
Dim expr as New Cexpression  
Print Expr.Evaluate ("x*4 + y ^2" , 5 , 10)
```

Questo approccio tende a ridurre la leggibilità del codice, in quanto occorre ricordare a quale variabile corrisponde ciascun valore. Una soluzione è di conservare le variabili di una stringa passata Cexpression in una relazione uno ad uno con le variabili nel programma, senza costringere il programmatore a passare i valori come argomenti a ciascuna chiamata ai metodi **value o Evaluate**.

Tuttavia, questo obiettivo appare a prima vista irraggiungibile, poiché sembra impensabile che la classe Cexpression legga il valore di una variabile dichiarata ed assegnata altrove nel programma. Inoltre risulta impensabile che la variabile x dichiarata nella stringa usata dalla variabile Cexpression corrisponda alla variabile x utilizzata nel programma principale.

Questi problemi sarebbero banali se il VB supportasse i puntatori alla memoria, ma purtroppo le cose non stanno così – a meno di non ricorrere a delle tecniche molto avanzate. Questo problema è stato risolto, utilizzando le caratteristiche **object – oriented** standard di VB.

Per prima cosa si è definita una nuova classe di nome **Cvariable**. Questa classe espone tre sole proprietà: **Value** (che è anche la proprietà di default per la classe), **Name e VarType**.

Gli oggetti Cvariable sono generati automaticamente e memorizzati in una collection privata della classe CExpression ogni volta che alla proprietà Expression della classe è assegnata una espressione che contiene una o più variabili.

Il programma principale può quindi ottenere un riferimento a tali oggetti CVariable usando il metodo Variable della classe CExpression :

```
Dim expr as New CExpression  
Dim x as Cvariable, y as Cvariable  
Expr. Expression = " x*4 + y ^2"  
Set x = expr.Variable ( "x")  
Set y= expr.Variable ("y")
```

Le variabili oggetto X ed Y fanno ora riferimento allo stesso oggetto Cvariable memorizzato internamente nella classe CExpression. Se il programma modifica la

proprietà value dell' oggetto X, indirettamente sarà modificato il valore della variabile X che appare nella espressione in corso di valutazione, e viceversa.

Questo significa che le 2 variabili X – quella interna alla espressione e la variabile utilizzata all' interno dell' applicazione – sono in realtà il medesimo oggetto , che è in realtà l' obiettivo che si cercava di raggiungere. Inoltre, si può omettere la proprietà Value poiché essa è la proprietà di default della classe Cvariable.

Se si omette la proprietà Value, gli oggetti Cvariable assomigliano così tanto alle variabili normali che non si potrebbe dire che si ha a che fare con una variabile oggetto a meno di non andare a vedere la sua dichiarazione. Per esempio, di seguito sono riportati i comandi per poter costruire una tabella bidimensionale di valori:

Dim Table (100, 100) As Double

X = 1

Do while x <=100

Table (x, y) = expr.Value

Y = y + 1

Loop

X = x +1

Loop

Uno dei pochi svantaggi di questa tecnica è che non si possono usare queste variabili oggetto come variabili di controllo in un ciclo For...Next, e si devono invece utilizzare i cicli Do...Loop.

Generatore di funzioni in Visual Basic

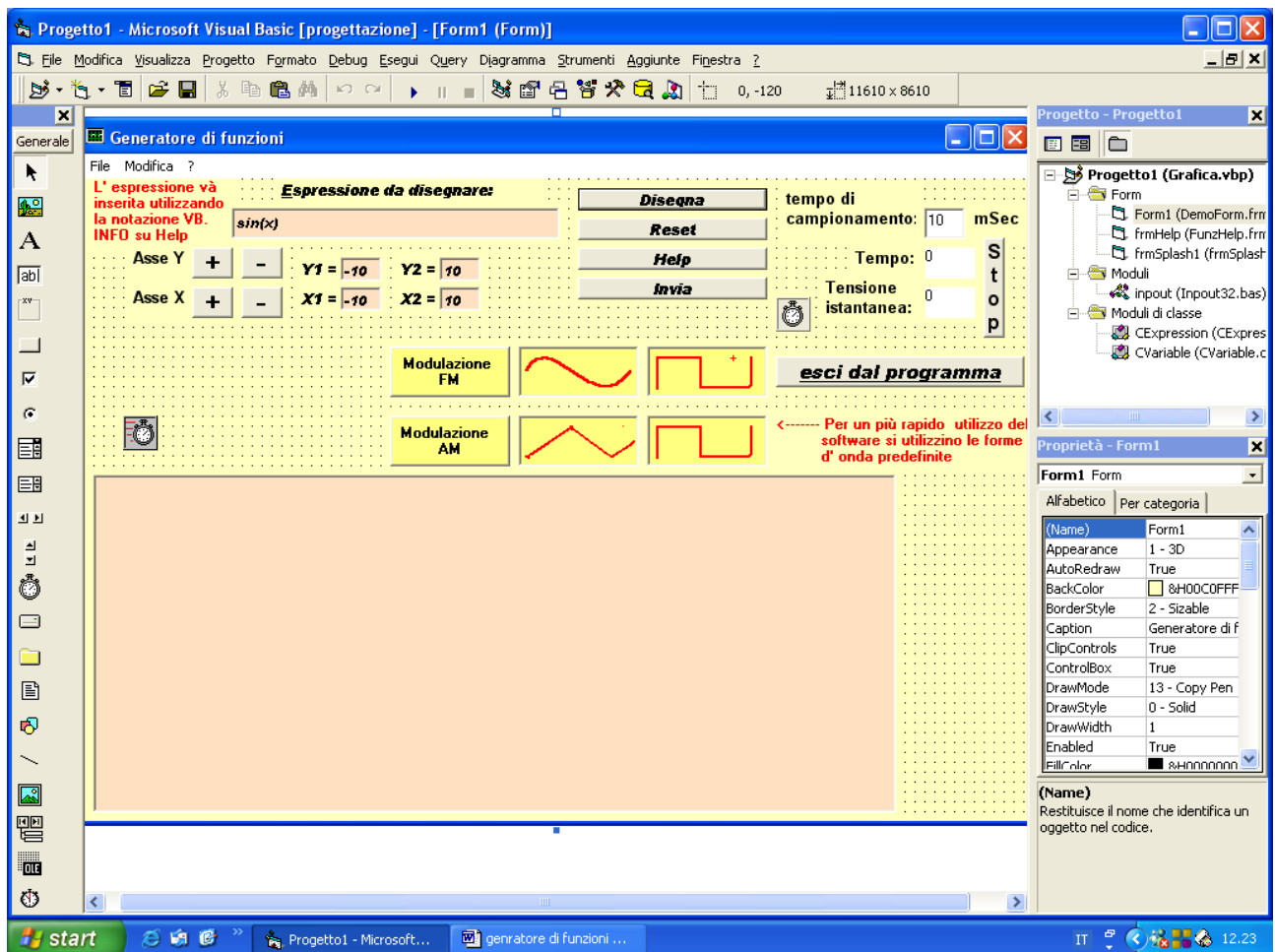
Nel programma realizzato, la classe Cexpression è stata impiegata per disegnare il grafico della funzione inserita nella txtExpression.

Prima di poter effettuare ciò, è stato effettuato un appropriato scaling degli assi X ed Y.

Si riporta la struttura del programma relativa al Form1.

Sono presenti alcuni componenti non standard presenti nella cartella del programma.

- **moduli di classe** per la valutazione delle espressioni in runtime **Cexpression e Cvariable;**
- modulo INPOUT32.bas per consentire a VB6 di gestire l'interfaccia Centronics utilizzando le istruzioni di Input e Output. Per poter utilizzare tali istruzioni in Windows98 si deve inserire in C:\\Windows\\System il file inpout32.dll. Nel caso in cui si opera in ambiente Windows 200, 200XP o NT è necessario scaricare il file *Windows 95NT Port I/O driver* che consente di caricare il file *port95nt.exe* che installa i necessari componenti. Il file è disponibile gratuitamente al sito www.driverlinx.com.
- MMTimer.ocx. Consente di utilizzare un Clock più veloce di quello standard di VB6. Il Timer standard non permette di operare con tempi inferiori a circa 10 msec. MMTimer è almeno 10 volte più veloce.



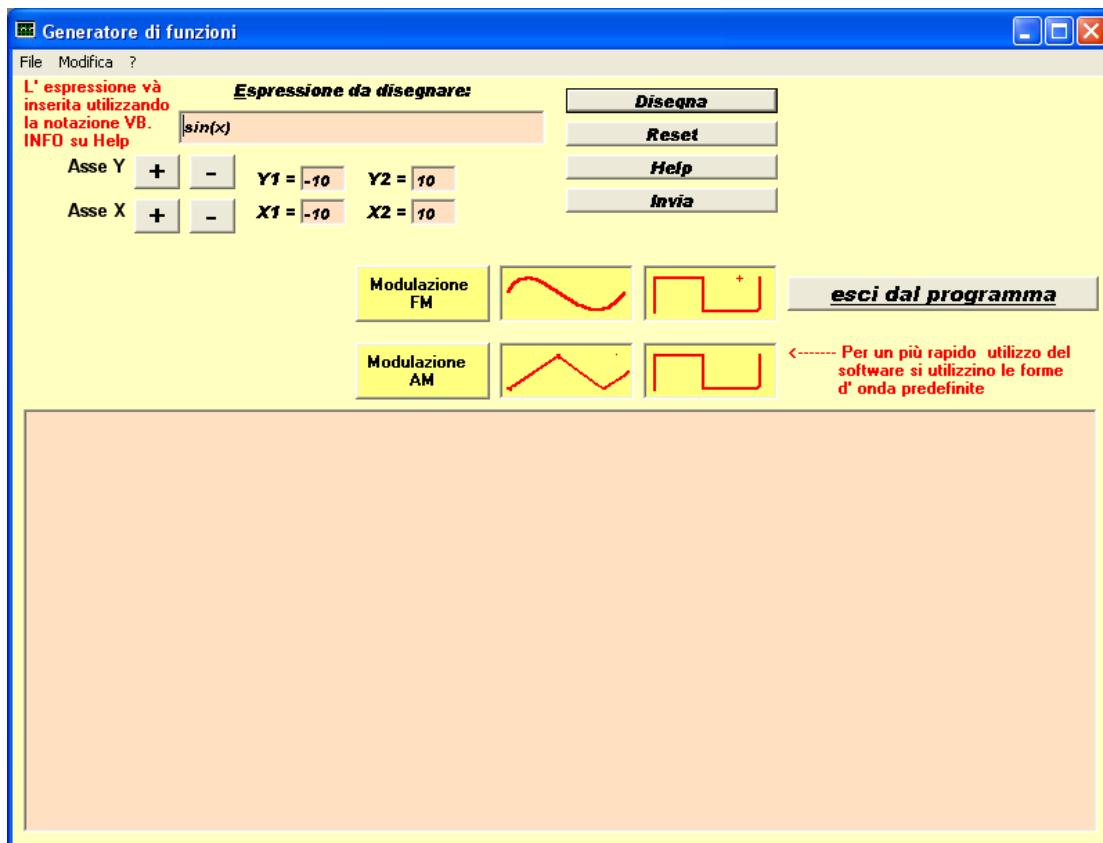
Si descrive brevemente il software.

Il programma è articolato in tre form:

- FormSplash
- DemoForm
- FormHelp

All' apertura di programma, il form che viene aperto è il FormSplash. Questo form realizza la presentazione del programma (mostrando la schermata riportata all'inizio del documento).

Il FormSplash è visibile per un certo lasso di tempo, determinato dall'intervallo del Timer1, dopodiché la schermata che si presenta è la seguente:



Sono presenti i seguenti elementi:

- Text **Espressione da disegnare** : all' interno di questa text v' inserita l'espressione della funzione da disegnare.
 - Comandi **Asse Y, Asse X + / -** : Questi comandi permettono di allargare o di restringere i margini entro i quali v' disegnata la funzione (effettuano lo zoom sul grafico)
 - Tex **X1, X2, Y1, Y2** : Consentono di inserire direttamente i margini entro i quali disegnare la funzione.
 - Comando **Disegna**: Consente di effettuare il disegno del grafico della funzione
 - Comando **Reset**: Cancella il grafico e l' espressione inserita nella text espressione
 - Comando **Help**: Apre il formHelp nel quale è inserito l' Help del programma.
 - Comandi **funzioni predefinite** : Consentono di scrivere direttamente la funzione matematica di funzioni predefinite (questa proprietà risulta utile nel momento in cui si utilizza questo software come generatore di funzioni)
 - Comando **Invia** : Consente l' invio sulla porta Centronics, della forma d' onda selezionata
 - Comando **Esci dal programma**: esce dal programma.

Quando il programma viene aperto, il software esegue le istruzioni contenute nella subroutine **Sub Form_Load**:

```
Private Sub Form_Load()  
Form1.Visible = False  
frmSplash1.Show  
MTimer1.Enabled = False  
lblFM.Caption = ""  
Label2.Visible = False  
Label3.Visible = False  
txtCamp.Visible = False  
lblTempo.Visible = False  
lblTensione.Visible = False  
lblTempo1.Visible = False  
lblTensione1.Visible = False  
cmdStop.Visible = False
```

In questa subroutine vengono definite le condizioni dalle quali parte il programma:

Il form1, non è visibile, il formSplash (la presentazione) viene mostrato (Show). Inoltre l'Mmtimer1 è disabilitato (questo componente è un particolare OCX, il cui funzionamento e proprietà verranno descritte in seguito). Inoltre viene esplicitato che la labelFM non possiede alcuna Caption (il suo funzionamento verrà descritto in seguito).

Infine, tutte le label, le text ed i comandi relativi alla subroutine Invia, sono resi invisibili. Questi componenti, alla pressione del tasto invia, verranno resi visibili.

Dall'esecuzione della subroutine **Form_Load ()** viene aperto il form Splash, ed eseguito quindi il codice relativo ad esso. In particolare le istruzioni eseguite sono le seguenti:

```
Private Sub Timer1_Timer()  
Unload frmSplash1  
Form1.Visible = True  
Timer1.Enabled = False  
End Sub
```

Il Form Splash esegue la subroutine del Timer1, che prevede le seguenti istruzioni:

- Termina il caricamento del form Splash
 - Rende visibile il Form1(Demoform), impostando su True la proprietà **visibile**
- Disabilita il Timer1 ;

Queste istruzioni sono eseguite una sola volta poiché al termine dell'esecuzione di queste istruzioni il Timer1 viene disabilitato

Al termine dell'esecuzione di queste istruzioni, il software apre la schermata principale.

Il cuore del programma si trova nella subroutine dei comandi **Disegna ed Invia**. Quando viene premuto il comando Disegna, il programma esegue la seguente subroutine:

Private Sub cmdPlot_Click()

```
Dim x1 As Double, x2 As Double  
Dim y1 As Double, y2 As Double  
Dim dx As Double, dy As Double  
Dim x As Double, y As Double  
Dim numOfPoints As Long  
Dim interval As Double  
Dim i As Long
```

```
expr.Expression = txtExpression  
If expr.ErrorCode Then Exit Sub
```

` Legge il range di X e Y

```
x1 = txtX1  
x2 = txtX2  
y1 = txtY1  
y2 = txtY2
```

` Inverte per mantenere l'ordine (swap)

```
If x1 > x2 Then
```

```
  dx = x1  
  x1 = x2  
  x2 = dx
```

```
End If
```

```
If y1 > y2 Then
```

```
  dy = y1  
  y1 = y2  
  y2 = dy
```

```
End If
```

```
dx = x2 - x1
```

```
dy = y2 - y1
```

` Il numero di punti è uguale a width

` sulla picture box in pixels

```
numOfPoints = Picture1.ScaleX(Picture1.Width, vbTwips, vbPixels) * 100
```

```
interval = dx / numOfPoints
```

` Conversioni

```
With Picture1
```

```
  .ScaleLeft = x1
```

```
  .ScaleTop = y2
```

```
  .ScaleWidth = dx
```

```
  .ScaleHeight = -dy
```

```
End With
```

` Traccia l'asse x e y

```
Picture1.Cls
```

```
Picture1.Line (x1, 0)-(x2, 0), vbBlack
```

```
Picture1.Line (0, y1)-(0, y2), vbBlack
```

In questa parte della subroutine vengono dimensionate le variabili contenute all'interno del programma come ad esempio i margini della funzione, l'intervallo, il numero dei punti con i quali la funzione verrà disegnata etc.

Questo comando è fondamentale poiché viene dichiarato che la proprietà Expression è uguale alla txtExpression (Il comando seguente svolge la funzione di uscire dalla subroutine, se l'espressione inserita contiene degli errori. In questa maniera non si creano errori nel programma e non è necessario riavviare lo stesso, nel momento in cui vengono inserite espressioni non corrette. Per la spiegazione delle altre istruzioni, si può fare riferimento ai commenti inseriti nella stesura del programma.

```

`Traccia graduazione assi
For i = Int(y1) To Int(y2)
  Picture1.Line (-dx / 100, i)-(dx / 100, i), vbBlack
Next
For i = Int(x1) To Int(x2)
  Picture1.Line (i, -dy / 50)-(i, dy / 50), vbBlack
Next

```

' Traccia il grafico della funzione

```

x = x1
For i = 1 To numOfPoints
  y = expr.Value(x)
  Picture1.PSet (x, y), vbRed
  x = x + interval
Next

```

Per disegnare l'espressione, il software va a calcolare tutti i punti della funzione negli intervalli inseriti e con il relativo passo. Associa quindi il calcolo dei punti a due variabili X ed Y (utilizzate come ascisse e come ordinate)

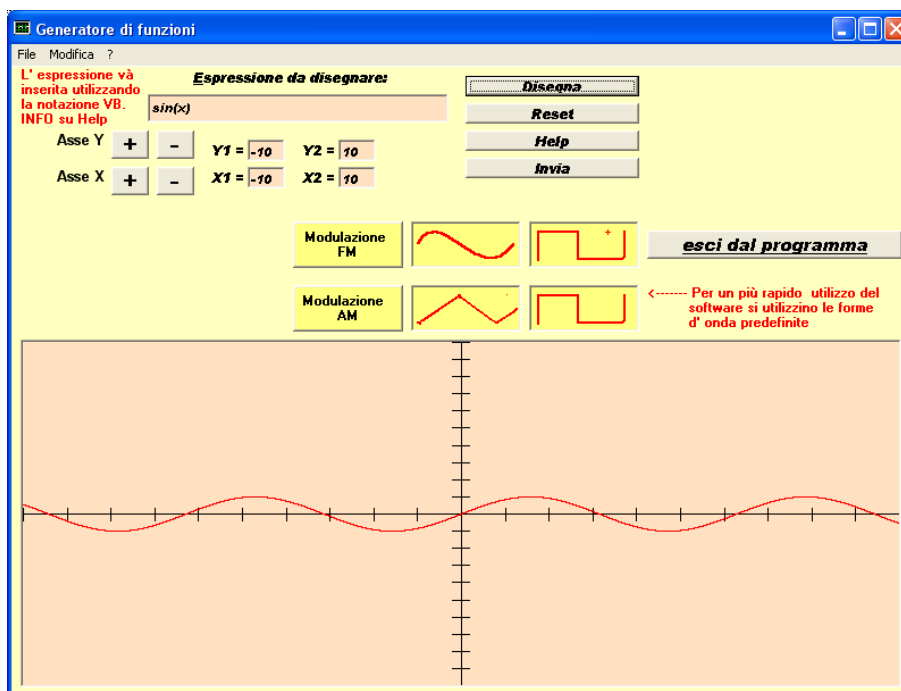
' Reset i fattori di scala

```

With Picture1
  .ScaleLeft = 0
  .ScaleTop = 0
  .CurrentX = 0
  .CurrentY = 0
End With
End Sub

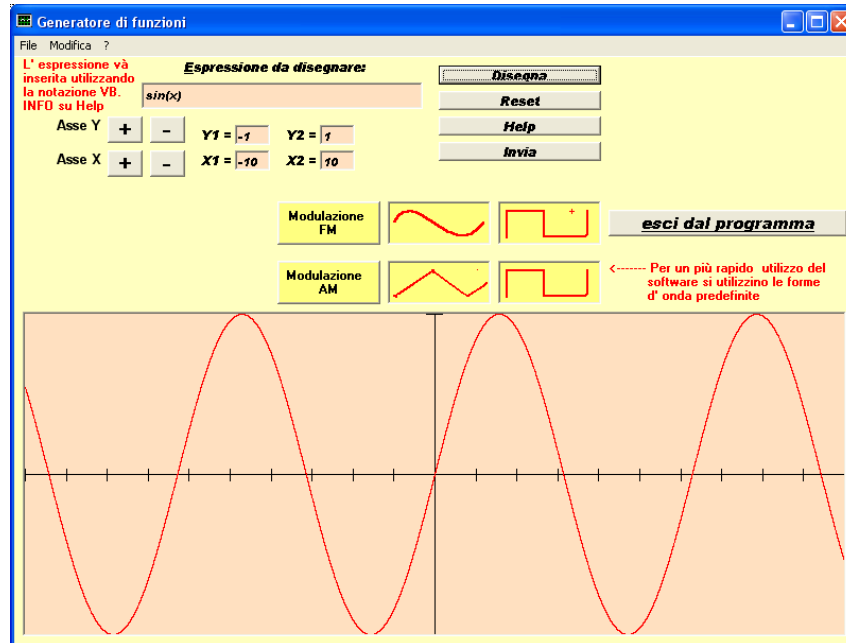
```

A questo punto, il software disegna la funzione contenuta nella txtExpression. Nel caso di default, l'espressione inserita nella txtExpression è **sin(x)**. Il risultato prodotto dal software è il seguente:

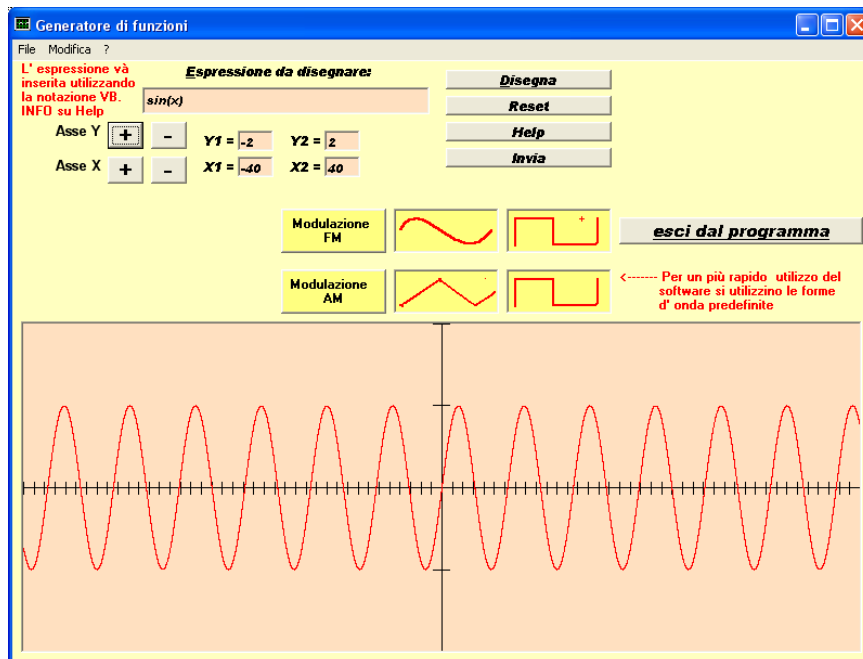


Attraverso i comandi **Asse Y**, **Asse X + / -** è possibile effettuare lo zoom in o lo zoom

Zoom in



zoom out



Come si può vedere dalle schermate, il valore contenuto nelle txt X1, X2, Y1,Y2 cambia contemporaneamente con la pressione dei comandi per il ridimensionamento degli assi. Alla pressione dei tasti infatti, il valore delle text per la definizione dei margini della funzione, viene diviso o moltiplicato per due a seconda del tasto che viene premuto.

Il codice inserito in linguaggio VB è il seguente:

```

Private Sub Command1_Click()
If Val(txtX1) = -1 Then txtX1.Text = "-1" Else: txtX1 = Val(txtX1) / 2
If Val(txtX2) = 1 Then txtX1.Text = "1" Else: txtX2 = Val(txtX2) / 2
Call cmdPlot_Click
End Sub

```

```

Private Sub Command2_Click()
txtX1 = Val(txtX1) * 2
txtX2 = Val(txtX2) * 2
Call cmdPlot_Click
End Sub

```

```

Private Sub Command3_Click()
If Val(txtY1) = -1 Then txtY1.Text = "-1" Else: txtY1 = Val(txtY1) / 2
If Val(txtY2) = 1 Then txtY2.Text = "1" Else: txtY2 = Val(txtY2) / 2
Call cmdPlot_Click
End Sub

```

```

Private Sub Command4_Click()
txtY1 = Val(txtY1) * 2
txtY2 = Val(txtY2) * 2
Call cmdPlot_Click
End Sub

```

Dall'analisi del codice, si può osservare che per l'allargamento degli assi, il codice inserito è semplice, poiché basta moltiplicare per 2 il valore contenuto nelle text.

Per la riduzione dei margini invece, bisogna tener conto che i valori contenuti nelle text, non devono essere minori dell'unità (positiva o negativa). Per effettuare questo controllo vengono utilizzate le istruzioni **ifthen...Else**.

Al termine di ogni ridimensionamento viene richiamata la subroutine **'Disegna'**, mediante l'istruzione **'Call cmdPlot_Click'**.

Mediante le forme d'onda predefinite, è possibile modificare il contenuto della txtExpression con un semplice click sulla forma d'onda desiderata (in questo caso vengono inserite le equazioni matematiche delle forme d'onda utilizzate nelle applicazioni elettroniche).

Le forme d'onda disponibili sono:

- | | |
|----------------------|-------------------------------|
| • Sinusoidale | sin(x) |
| • Triangolare | Tan(sin(x)) |
| • Quadra (unipolare) | Sgn(Sin(x))+1)/2 |
| • Quadra (bipolare) | Sgn(Sin(x)) |
| • Modulazione AM | sin(x+sin(x)) |
| • Modulazione FM | (1 + 1 *cos(x))*cos(x) |

Le istruzioni per modificare il contenuto della txtExpression in funzione del comando premuto sono le seguenti:

(supponendo di aver premuto il comando 'Seno')

```
Private cmdSeno_Click  
txtExpression.Text = "Sin(x)"  
End Sub
```

Come si può vedere la procedura effettuata è estremamente semplice poiché basta modificare la proprietà Text della txtExpression.

Le cose sono cambiate un po' per scrivere il codice relativo ai comandi per la modulazione AM e per la modulazione FM.

Nel caso della modulazione AM:

```
Private Sub cmdModulazioneAM_Click()
```

```
txtExpression.Text = "(1 + 1 *cos(x))*cos(x)"  
lblFM.Caption = "Si ricorda che l' equazione generica di un segnale modulato  
in ampiezza è  $y=(A_p + A_m *cos( F_m *x)) *cos( F_p*x)$  dove  $A_p$  è la ampiezza  
del segnale portante,  $A_m$  è l' ampiezza del segnale modulante,  $F_m$  è la  
frequenza del segnale modulante ed  $F_p$  è la frequenza del segnale portante"
```

```
End Sub
```

Nel caso della modulazione FM:

```
Private Sub cmdModulazioneFM_Click()
```

```
txtExpression.Text = "sin(x+sin(x))"  
lblFM.Caption = "Si ricorda che l' equazione generica di un segnale modulato  
in frequenza è  $y= A_p *sin(F_p*x+A_m *sin(F_m*x))$ . Dove: $A_p$  è l' ampiezza del  
segnale portante,  $A_m$  è l' ampiezza del segnale modulante,  $F_p$  è la frequenza  
del segnale portante ed  $F_m$  è la frequenza del segnale modulante"
```

```
End Sub
```

Risulta di estrema facilità comprendere che l'unica variazione effettuata rispetto agli altri comandi (per le forme d' onda predefinite), è quella di variare la proprietà Caption della label FM (lblFM).

In questa label vengono inseriti degli aiuti riguardanti la modulazione AM ed FM. Viene ricordata l' equazione generica della specifica modulazione. Alla pressione del tasto RESET, il testo della label torna ad essere nullo (**lbl FM.Caption = ""**)

Comando dell'interfaccia Centronics

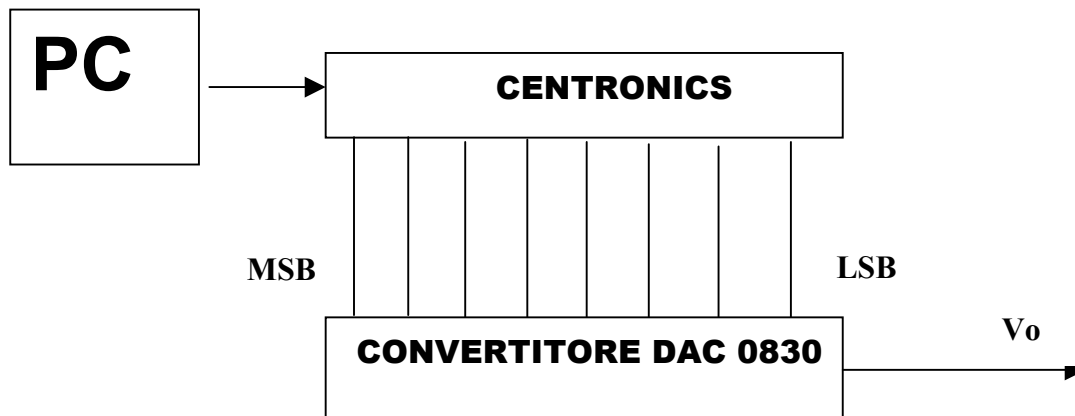
È possibile inviare sulla l'interfaccia CENTRONICS, la forma d'onda selezionata.

E' possibile cioè **utilizzare il computer come generatore di funzioni.**

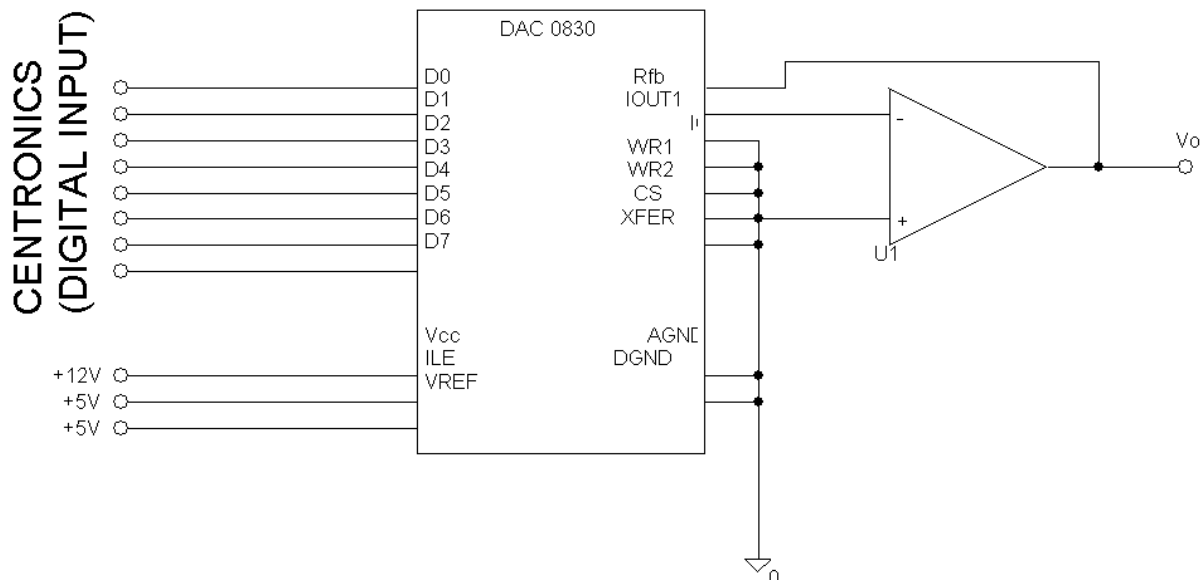
Per compiere questa operazione è stato necessario costruire un circuito, in grado di convertire la grandezza digitale fornita dalla CENTRONICS, in un segnale elettrico analogico. In Visual Basic si deve attivare il **pulsante Invia**

Il circuito utilizzato è un convertitore digitale/ analogico realizzato con DAC 0830.

Si riporta lo schema a blocchi della connessione effettuata, e lo schema circuitale del convertitore digitale analogico realizzato.



Si mostra lo schema base di funzionamento del DAC 0830.

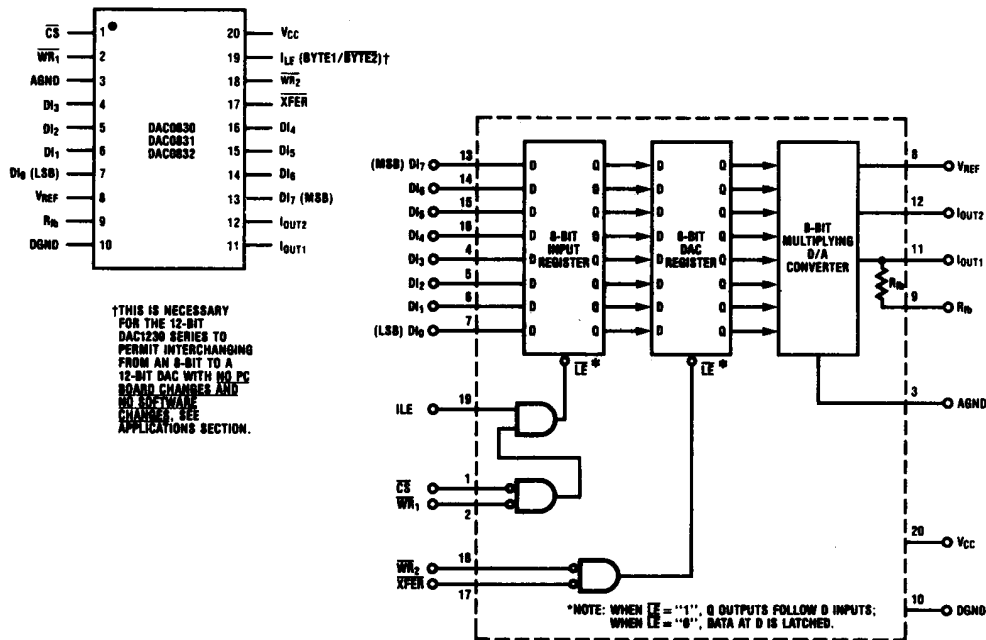


Il circuito realizzato, utilizza il DAC 0830, un convertitore digitale analogico integrato, utilizzato in molte applicazioni elettroniche. Il circuito presenta 8 linee (digital input), le quali vanno collegate con l' uscita dati della CENTRONICS.

Il funzionamento del DAC 0830 è basato su una rete di resistenze R-2R e quindi presenta due uscite in corrente. In figura si mostra il suo schema a blocchi.

Il circuito è costituito da due registri ad 8 bit ciascuno dei quali realizzato da 8 flip-flop D , da una rete di resistenza R-2R e da una logica combinatoria di controllo.

La memorizzazione della parola binaria applicata agli ingressi di ciascun registro avviene portando al livello basso la linea LE (latch-enable). Se $LE = 1$ le uscite Q inseguono gli ingressi D .



Schema a blocchi del DAC0830.

Altre particolari linee del DAC 0830 sono:

CS : Chip select (attiva bassa). In combinazione con ILE abilita WR_1 .

ILE : Input latch enable (attiva alta).

WR_1 : Write 1 (attiva bassa). E' usata per caricare i bit del dato di ingresso digitale (DI) nel primo registro. Il caricamento avviene ponendo $WR_1 = 0$ se $ILE = 1$ e $CS = 0$. Il dato rimane memorizzato nel registro (latch) se $WR_1 = 1$.

WR_2 : Write 2 (attiva bassa). Se $XFER = 0$, ponendo $WR_2 = 0$ si trasferisce il dato digitale dal registro di ingresso DAC.

$XFER$: Transfer control signal (attiva bassa).

Oltre a queste il circuito presenta altre linee:

DI_0-DI_7 : Ingressi digitali. Il bit DI_0 è quello meno significativo (LSB) mentre DI_7 è quello più significativo (MSB).

I_{OUT1} : Corrente di uscita 1. E' massima se nel registro Dac vi sono tutti 1 e vale 0 se in tale registro vi sono tutti 0.

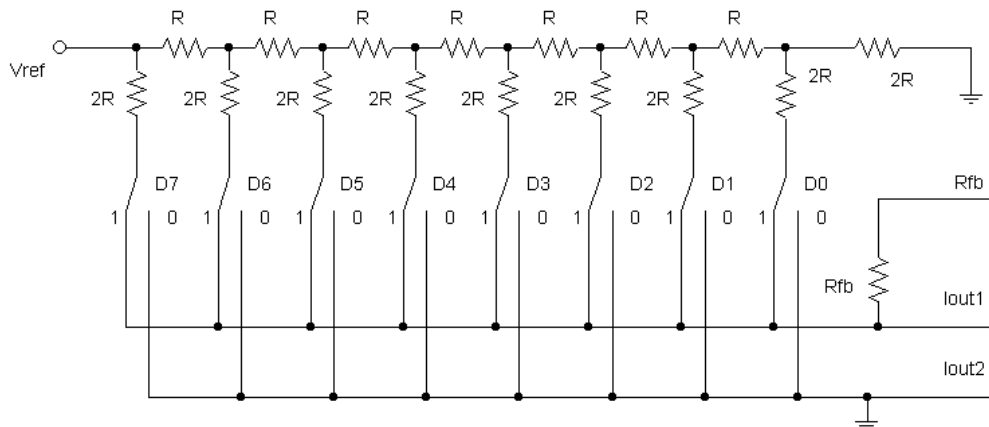
I_{OUT2} : Corrente di uscita 2. E' esattamente il contrario di I_{OUT1} per cui si ha per ogni ingresso digitale: $I_{OUT1} + I_{OUT2} = \text{costante}$.

R_{fb} : Resistenza di feedback. E' cablata all'interno del chip e dovrebbe essere sempre usata quando si impiega un operazionale per ottenere l'uscita in tensione.

(Nel caso del circuito realizzato in questo progetto, l' amplificatore operazionale è utilizzato in configurazione convertitore corrente – tensione, ed è reazionato tramite la resistenza di feedback interna al DAC).

V_{REF} : Tensione di riferimento. A questo ingresso si deve applicare una sorgente di tensione costante di precisione di valore compreso tra -10 V e + 10V che serve a polarizzare le resistenze della rete R-2R.

Di seguito è riportato lo schema interno del DAC :



Il codice digitale di ingresso controlla la posizione dei deviatori elettronici e quindi il valore delle correnti I_{O1} ed I_{O2} .

La corrente di ingresso erogata da V_{REF} si ottiene cortocircuitando a massa tutti i deviatori elettronici. Ci si accorge che si possono effettuare delle esemplificazioni poiché si trovano resistenze in parallelo ed in serie (del valore R o 2R). Si giunge, dopo aver effettuato queste esemplificazioni, ad un circuito costituito da V_{REF} e da una resistenza in serie del valore R. Per la legge di Ohm, la corrente erogata da V_{REF} è pari a:

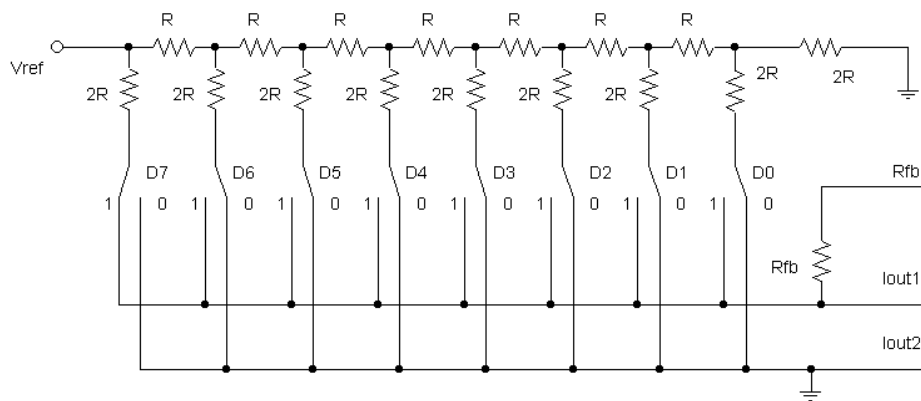
$$I = \frac{V_{REF}}{R}$$

Per determinare l' espressione della corrente di uscita I_{O1} , bisogna andare ad analizzare volta per volta, il valore assunto dalla corrente in questione, in corrispondenza dell' 1 logico applicato ad un ingresso alla volta.

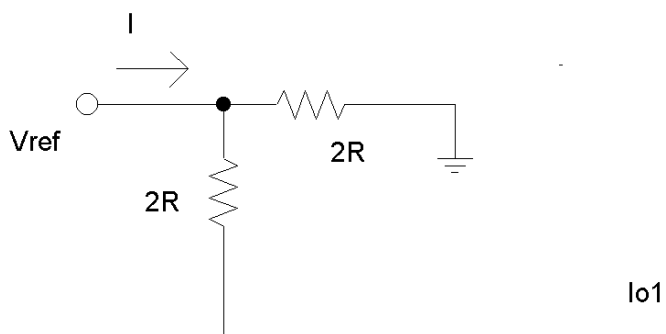
Di seguito viene riportato tale procedimento:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

Il circuito equivalente è il seguente:



Partendo, da destra, ci si accorge che è possibile effettuare delle esemplificazioni sulle resistenze (poiché due resistenze del valore $2R$, sono poste in parallelo; si ha quindi una resistenza equivalente di valore R posta in serie ad una ltra di valore R , quindi $R+R= 2R...$ e così via). Si giunge quindi al seguente circuito:

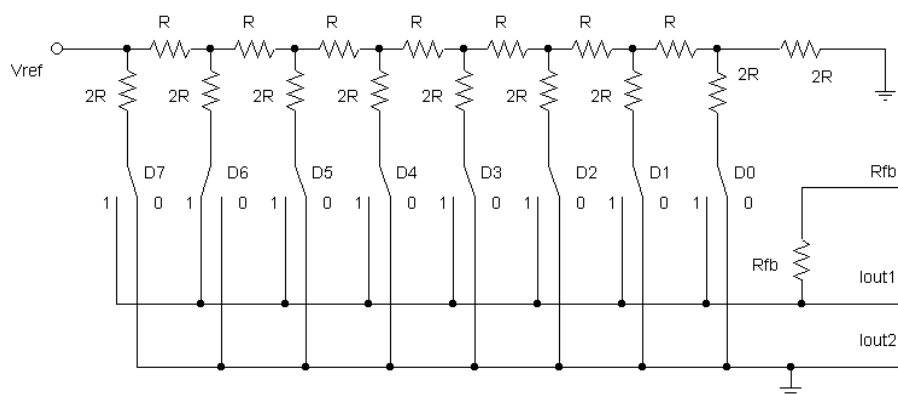


La corrente I si divide chiaramente in due rami che hanno la stessa resistenza. Quindi la corrente I_{01} , assume il seguente valore:

$$I_{01} = \frac{V_{REF}}{2} = \frac{I}{2}$$

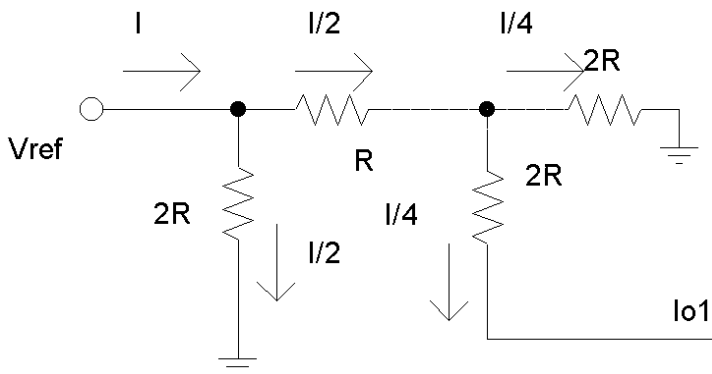
Combinazione n.2
 D7 D6 D5 D4 D3 D2 D1 D0
 0 1 0 0 0 0 0 0

Il circuito equivalente è il seguente:



Sempre partendo da destra, si procede ad analoghe semplificazioni nella rete $R-2R$.

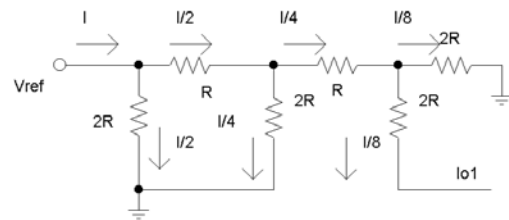
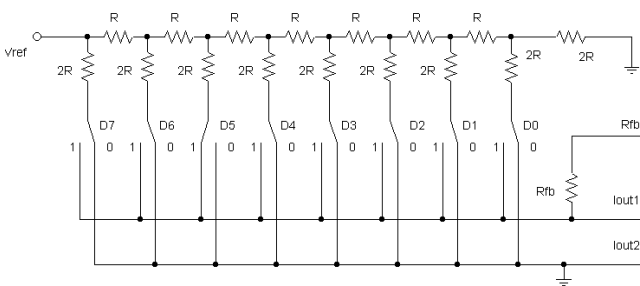
Il circuito risultante è costituito dal 'generatore' VREF, dalla resistenza del valore 2R, in parallelo al ramo costituito dalla serie della resistenza di valore R e del parallelo delle resistenze di valore 2R:



Risulta chiaro che la corrente I/2, giunta al parallelo delle due resistenze del valore 2R, si divide in due parti uguali per cui:

$$I_{O1} = \frac{\frac{V_{REF}}{R}}{2} = \frac{I}{4}$$

combinazione n.3
D7 D6 D5 D4 D3 D2 D1 D0
0 0 1 0 0 0 0 0



$$I_{O1} = \frac{\frac{I}{4}}{2} = \frac{I}{8}$$

Ci si accorge quindi che il procedimento della semplificazione è sequenziale. A seconda del bit posto a 1, man mano che si va avanti, la corrente si divide sempre in due parti uguali, quindi ad ogni nodo si dimezza. Di seguito, verranno quindi riportate solo le espressioni della corrente di uscita Io1, in funzione della combinazione di ingresso applicata:

combinazione n.4
D7 D6 D5 D4 D3 D2 D1 D0
0 0 1 0 0 0 0 0

$$I_{O1} = I/16$$

combinazione n.5
D7 D6 D5 D4 D3 D2 D1 D0
0 0 1 0 0 0 0 0

$$I_{O1} = I/32$$

combinazione n.6 Io1=I/64
 D7 D6 D5 D4 D3 D2 D1 D0
 0 0 1 0 0 0 0 0

combinazione n.7 Io1=I/128
 D7 D6 D5 D4 D3 D2 D1 D0
 0 0 1 0 0 0 0 0

combinazione n.8 Io1=I/256
 D7 D6 D5 D4 D3 D2 D1 D0
 0 0 1 0 0 0 0 0

Risulta quindi evidente che il 'peso' dei bit nella conversione da digitale ad analogica, è diverso. A seconda della posizione degli interruttori elettronici, e quindi dello stato delle linee digitali di ingresso, si avrà in uscita una tensione di ampiezza diversa.

Se il bit è posto a 1, la corrispondente corrente che fluisce in quel ramo, contribuisce al valore di Io1. Se il bit è 0, la corrente va verso massa (Io2).

Ricordando che la corrente Io1 attraversa Rfb e l' A.O è montato in configurazione convertitore corrente tensione invertente, la tensione di uscita dell' A.O assume la seguente espressione:

$$V_o = -I_{o1} \cdot R_{fb}$$

$$V_o = -R_{fb} \cdot \left(\frac{I}{2} \cdot b_7 + \frac{I}{4} \cdot b_6 + \frac{I}{8} \cdot b_5 + \frac{I}{16} \cdot b_4 + \frac{I}{32} \cdot b_3 + \frac{I}{64} \cdot b_2 + \frac{I}{128} \cdot b_1 + \frac{I}{256} \cdot b_0 \right)$$

$$V_o = -R_{fb} \cdot \left(\frac{I}{2} \cdot b_7 + \frac{I}{2^2} \cdot b_6 + \frac{I}{2^3} \cdot b_5 + \frac{I}{2^4} \cdot b_4 + \frac{I}{2^5} \cdot b_3 + \frac{I}{2^6} \cdot b_2 + \frac{I}{2^7} \cdot b_1 + \frac{I}{2^8} \cdot b_0 \right)$$

Mettendo in evidenza I e moltiplicando e dividendo per 28 si ottiene:

$$V_o = -\frac{R_{fb}}{2^8} \cdot I \cdot (b_7 \cdot 2^7 + b_6 \cdot 2^6 + b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2)$$

Ponendo il risultato ottenuto nella parentesi uguale a D, cioè il numero digitale applicato all' ingresso del convertitore, e sostituendo ad I il valore calcolato all' inizio di questa relazione (la corrente erogata da VREF, ponendo tutti gli ingressi a 0 cioè posizionando i deviatori sulla massa), si ottiene la seguente espressione:

$$V_o = -\frac{R_{fb}}{2^8} \cdot \frac{V_{REF}}{R} \cdot D$$

Ponendo ancora R= Rfb, si ottiene l' espressione definitiva della tensione d' uscita del circuito:

$$V_o = -\frac{V_{REF}}{M} \cdot D$$

Dove:

- D è il numero digitale applicato sugli ingressi del convertitore
- VREF, è la tensione di riferimento decisa dall'utente
- M è il modulo del convertitore, cioè il numero di configurazioni che gli interruttori digitali possono assumere, e quindi il numero di livelli di tensione che il convertitore può associare alle combinazioni binarie in ingresso.

Un problema incontrato durante la realizzazione del progetto è stato quello di minimizzare i tempi di lavoro del VB6. Per l'invio dei dati sull'uscita infatti, è stato impiegato una particolare OCX: l' **MMTimer**. Utilizzando questo particolare timer è stato possibile ridurre l'intervallo di campionamento ad un millisecondo. In questo modo, la frequenza di lavoro è dell'ordine di 1 kHz.

Si è ricorso all'utilizzo di questa OCX poiché il normale Timer fornito dal VB6, non riesce a lavorare con intervalli di tempo molto bassi. Il VB6 non dà errori durante il suo utilizzo a frequenze elevate, ma nella pratica risulta evidente che al di sotto di una certa frequenza non lavora correttamente.

Quando si clicca sul pulsante **INVIA**, viene eseguita la seguente subroutine:

```
Private Sub cmdInvia_Click()  
Dim p As Double  
Timer2.Enabled = False  
MMTimer1.Enabled = True  
MMTimer1.Interval = txtCamp.Text  
p = Val(txtCamp.Text) / 100  
Out (890), 0  
Label2.Visible = True  
Label3.Visible = True  
txtCamp.Visible = True  
lblTempo.Visible = True  
lblTensione.Visible = True  
lblTempo1.Visible = True  
lblTensione1.Visible = True  
cmdStop.Visible = True  
lblConsiglio.Caption = "Per fermare l' invio dei dati sulla CENTRONICS  
premere il pulsante STOP"  
End Sub
```

Durante l'esecuzione di questa subroutine, viene abilitato l'MMTimer (mediante il comando **MMTimer1.Enabled = True**). Inoltre, viene definito l'intervallo del timer, come il valore contenuto nella **txtCamp** ...: **MMTimer1.Interval = txtCamp.Text**

La subroutine **cmdInvia_Click ()** contiene inoltre il comando **Out (890), 0**, il cui funzionamento verrà spiegato in seguito.

Quando un Timer viene attivato, ad intervalli regolari (definiti dalla proprietà *Interval*), viene richiamata la sua subroutine.

Il software svolge quindi ciclicamente i seguenti comandi:

```
Private Sub MMTimer1_Timer()  
Static x As Double  
Dim p As Double 'p è l' incremento della variabile x  
p = 0.01  
x = x + p  
lblTempo.Caption = Format(x, "#0.00")
```

```
Dim y As Double  
expr.Expression = txtExpression  
If expr.ErrorCode Then Exit Sub  
y = 127 * expr.Value(x) + 128  
Out (888), y
```

```
lblTensione.Caption = Format(y, "#0.00")
```

Durante l' esecuzione di questa subroutine viene definito un passo p, ed ogni qual volta la subroutine viene richiamata dall' MMtimer, il valore della X si incrementa del valore di questo passo. Subito dopo viene ripetuta la stessa procedura usata per il comando disegna. La proprietà Expression della classe, viene identificata con il contenuto della txtExpression.

Il software calcola quindi il valore dell' espressione, per il valore istantaneo di X, e lo pone sull' uscita CENTRONICS del PC. Questa operazione viene svolta attraverso i seguenti comandi:

```
expr.Expression = txtExpression  
If expr.ErrorCode Then Exit Sub  
y = 127 * expr.Value(x) + 128  
Out (888), y
```

Le ultime due righe di questa subroutine sono quelle più importanti, poiché una, calcola il valore da porre sull'uscita della CENTRONICS, mentre l' ultima, trasferisce il valore calcolato sull' uscita. La Y da inviare in uscita subisce particolari variazioni poiché la funzione potrebbe assumere valori negativi (per questo viene sommato alla y il valore 128. Quest' ultimo non è preso a caso, ma deriva dalle seguenti considerazioni: poiché il numero dei bit utilizzati dalla CENTRONICS, per l' invio dei dati è pari ad 8, il numero delle combinazioni possibile è pari a $2^8=256$. Il numero massimo è quindi 255.

L' onda da inviare sull' uscita deve però assumere solo valori positivi. Per questo motivo il valore della funzione da inviare è moltiplicato prima per 127 e poi sommato a 128 (= $256/2$).

L' ultima istruzione è quella che consente di inviare il valore della y sull' uscita:

```
Out (888), y
```

Quest'ultima istruzione per poter essere interpretata da VB6 necessita l'installazione di una DLL (INPOUT32.dll) all'interno della cartella *'System' di Windows*. Senza l' installazione

di questo componente, il programma non è in grado di gestire l'invio dei dati sulla CENTRONICS.

La DLL utilizzata, non garantisce però il corretto funzionamento con tutti i sistemi operativi, poiché può essere non compatibile con il sistema operativo in uso. Questo problema viene incontrato ad esempio se si utilizza Windows 200XP. In questo caso, come già detto in precedenza, si deve installare il **file port95nt.exe** scaricabile gratuitamente dal sito www.driverlinx.com.

Il software descritto in questo progetto opera in Windows98.

Interfaccia Centronics

La CENTRONICS è un'interfaccia parallela ad 8 bit di tipo asincrona usata soprattutto per collegare un computer ad una stampante parallela. Il flusso dei dati è tipicamente monodirezionale e va, naturalmente, dal computer alla stampante. Recentemente grazie all'evoluzione della interfaccia è possibile anche sulle stesse linee, dati che vanno dal dispositivo periferico al computer e ciò consente l'utilizzo di tale interfaccia anche per il collegamento di dispositivi di input come ad esempio, lo scanner o la webcam.

Il connettore sul retro del computer è di tipo D a 25 poli femmina. Su un PC possono prendere posto fino a tre interfacce parallele denominate LPT1, LPT2 e LPT3 (Line Printer Terminal). Ciascuna delle tre LPT presenta 3 indirizzi contigui destinati alle periferiche di I/O.

L'indirizzo base della LPT1 è 888 (in esadecimale : 378), gli altri due indirizzi sono 889 e 890. L'indirizzo base della LPT2 è 632 (in esadecimale :278), gli altri due indirizzi sono 633 e 634. L'indirizzo base della LPT3 è 956 (In esadecimale : 3BC), gli altri due indirizzi sono 957 e 958.

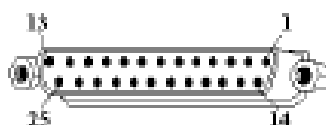
Si riassume la situazione nella seguente tabella:

	Indirizzo base	Indirizzo base+1	Indirizzo base+2
LPT1	888	889	890
LPT2	632	633	634
LPT3	956	957	958

Alla LPT1 viene riservato l'interrupt IRQ7. Il registro base di indirizzo 888, denominato **registro dati**, contiene 8 bit di uscita del PC.

Il registro di indirizzo successivo 889, noto come **registro di stato**, è accessibile solo dall'esterno e solamente per 5 dei suoi 8 bit. E' detto registro di stato poiché ciascuna delle 5 linee individua un particolare stato in cui si trova la stampante (occupata, carta esaurita, errore etc.). Il registro di indirizzo 890, noto come **registro di controllo**, rende disponibili solo 4 bit di uscita.

In figura si mostra il connettore a 25 poli posto sul retro del computer.



Nella tabella seguente si riporta la piedinatura, la denominazione, la direzione, l'indirizzo e l'uso delle linee che l'interfaccia Centronics mette a disposizione sul connettore D a 25 poli agli indirizzi 888, 889 e 890 della LPT1.

PIN	NOME	DIREZIONE	INDIRIZZO ED USO
1	$\overline{\text{STROBE}}$	USCITA	890 OUT 890,1 0V 890 OUT 890,0 5V
2	DATA1	USCITA	OUT 888,N Per $0 \leq N \leq 255$
3	DATA2	USCITA	
4	DATA3	USCITA	
5	DATA4	USCITA	
6	DATA5	USCITA	
7	DATA6	USCITA	
8	DATA7	USCITA	
9	DATA8	USCITA	
10	ACK	INGRESSO	B3 (64) SE ALTO
11	$\overline{\text{BUSY}}$	INGRESSO	B4 (128) SE BASSO
12	PAPER OUT	INGRESSO	B2 (32) SE ALTO
13	SELECTED	INGRESSO	B1 (16) SE ALTO
14	$\overline{\text{AUTOFEED}}$	USCITA	890 OUT 890,2 0V
15	ERROR	INGRESSO	B0 (8) SE ALTO
16	INIZIALIZE PRINTER	USCITA	890 OUT 890,4 5V
17	$\overline{\text{SELECT INPUT}}$	USCITA	890 OUT 890,8 0V
18...25	GND		

Registro dati (indirizzo 888) :

Le linee di uscita DATA8...DATA1, di indirizzo 888 (378 esadecimale), situate tra i pin 9....2, sono memorizzate, cioè rappresentano i bit di uscita di altrettanti flip-flop.

Linee	DATA 8	DATA 7	DATA 6	DATA 5	DATA 4	DATA 3	DATA 2	DATA 1
PIN	9	8	7	6	5	4	3	2

Registro di stato (indirizzo 889):

L'interfaccia Centronics possiede 5 linee di ingresso all'indirizzo 889 (379 esadecimale) con i seguenti valori e logiche di funzionamento:

- $\overline{\text{BUSY}}$ (pin 11) vale 128 se è al livello basso (logica negativa).
- ACK (pin 10) vale 64 se è al livello alto (logica positiva).
- PAPER OUT (pin 12) vale 32 se è al livello alto (logica positiva).
- SELECTED (pin 13) vale 16 se è al livello alto (logica positiva).
- ERROR (pin 15) vale 8 se è al livello alto (logica positiva).

Linee	BUSY	ACK	PAPER OUT	SELECTED	ERROR	1	1	1
PIN	11	10	12	13	15			

I rimanenti 3 bit meno significativi del byte di ingresso non sono accessibili sul connettore e sono poste, internamente all'interfaccia, al livello alto (1112 = 710).

Avendo a disposizione 5 bit (32 combinazioni) è possibile acquisire, via software in una variabile A, un valore compreso tra 0 e 31.

Ciò si realizza con i seguenti passaggi:

$A = A \text{ XOR } 128$ (per complementare il bit applicato su BUSY);

$A = A - 7$ (per azzerare i tre bit meno significativi : operazione, comunque, non necessaria);

$A = A / 8$ (per lo scorrimento a destra di 3 posizioni).

L'algorithmo di acquisizione in linguaggio QBASIC risulta, pertanto:

$A = ((\text{INP}(889) \text{ XOR } 128) - 7) / 8$

In questo modo la parola di ingresso A assume un valore compreso tra 0 e 31.

Registro di controllo (indirizzo 890):

L'indirizzo Centronics 890 (37A esadecimale) rende disponibile in uscita altri 4 bit di cui tre attivi in logica negativa ed una in logica positiva :

$\overline{\text{STB}}$ = STROBE (pin 1) vale 1 se è al livello basso (logica negativa).

$\overline{\text{AF}}$ = AUTOFEED (pin 14) vale 2 se è al livello basso (logica negativa).

$\overline{\text{IP}}$ = INIZIALIZE PRINTER (pin 16) vale 4 se è al livello alto (logica positiva).

$\overline{\text{SI}}$ = SELECT INPUT (pin 17) vale 8 se è al livello basso (logica negativa).

$\overline{\text{IRQE}}$ = ABILITA INTERRUPT vale 16 se è al livello alto (logica positiva)

ma non è disponibile fisicamente.

Linee	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
				IRQE	SI	IP	AF	STB
PIN			17	16	14	1		

Il linguaggio di programmazione Visual Basic consente di mettere a punto interfacce grafiche estremamente elaborate ed accattivanti con poca fatica, ma purtroppo in forma nativa, non supporta le istruzioni di input e di output direttamente da unità periferiche.

Per ovviare a ciò è possibile utilizzare, come già detto, un particolare file di libreria DLL, come ad esempio **INPOUT32.DLL**, funzionante su computer con Windows 95 e 98 ma

non con Windows 2000. Tale DLL v'è inserita nella cartella **C:\Windows\System**, del proprio Hard Disk.

Affinché si possa invocare nel nostro applicativo, il file di libreria INPOUT32.DLL, si deve includere nel progetto VB generato il modulo **INPOUT32.BAS**, presente nel file riportato:

```
Public Declare Function Inp Lib "Inpout32.dll"  
Alias "Inp32" (ByVal PortAddress as Integer) as Integer  
Public declare Sub Out Lib "inpout32.dll"  
Alias "out32" (ByVal PortAdress as Integer, ByVal Value as Integer)
```

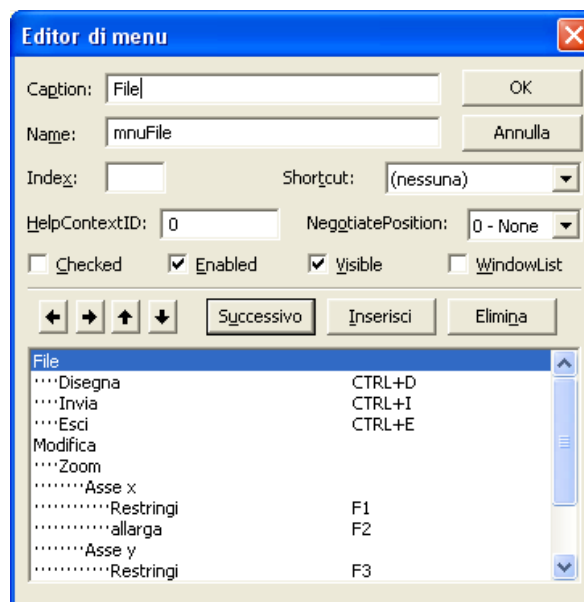
Nel software realizzato vengono utilizzate due istruzioni già incontrate in precedenza riportate qui di seguito con il loro funzionamento:

- **Out (890), 0** Pone a 0 il bit 5 del registro di controllo (890), cioè dichiara il byte DATA8....DATA1 del registro dati all'indirizzo 888 come uscita (OUTPUT).
- **Out (888), y** Pone il valore y in uscita (sul byte DATA8...DATA1). Manda in uscita i valori dell'onda impostata mediante il software.

Nella parte finale della realizzazione del progetto ci si è posti il problema di rendere il software piacevole dal punto di vista grafico, e sono state perfezionate alcune parti del programma.

E' stato ad esempio, aggiunto **editor di menù**.

Per aggiungere l'editor di Menù è stata aperta la finestra per la creazione dell'editor di Menù mediante la pressione combinata dei tasti CTRL+E. Al programmatore appare la seguente finestra:



Al programmatore viene quindi richiesto di assegnare ad ogni comando una **caption** cioè un nome da visualizzare durante l'esecuzione del software, ed un **name**, cioè il nome con cui verrà chiamato il comando durante la stesura del codice.

Ad ogni comando è stato poi assegnata una combinazione di tasti, mediante l'utilizzo della proprietà **shortcut**.

In conclusione, il progetto realizzato non sarà un ottimo generatore di funzioni, poiché la frequenza alla quale lavora il Visual Basic è troppo bassa, ma è comunque un ottimo strumento di lavoro matematico, poiché permette di visualizzare l'andamento di una funzione matematica, semplicemente scrivendo la sua equazione.

Quest'operazione viene spesso svolta da software molto complessi quale ad esempio il 'Derive'.

Per realizzare un buon generatore di funzioni, bisognerebbe scrivere il listato del programma in linguaggio macchina (utilizzando ad esempio il linguaggio C).

Di seguito è riportata la schermata del programma durante l'invio di un segnale modulato in ampiezza sulla CENTRONICS:

