VB 2005

PETROS KRONIS

VISUAL BASIC

programming for electronics enthusiasts

Free booklet with Elektor Electronics January 2006

elektor electronics
leading the way

9 770268 451128 01

Contents by subject

Subject	Page
Introduction	3
Starting Visual Basic 2005	4
The Integrated Development Environment of VB 2005	5
Programming through a simple example: Ohm's Law	5
Placing controls on the form	6
Attaching code to controls	9
Saving programs	10
Running the program	10
Program Resistors in Series and in Parallel	11
Variables	11
Program Resistor Values	15
Conditional statements	17
Program Vibration/Oscillation systems	19
Loops	19
Program OP-Amp Database	22
Creating a file on a mass storage device	22
Loading information from a file	25
Adding items in the program Resources	25
Executable files	26

Visual Basic for the Electronics Enthusiasts

This short course on Visual Basic 2005 programming is designed especially for the electronics enthusiasts. These are people most likely to be interested in learning computer programming. Modern electronics technology pushes more and more towards programmable devices and many of these, need to communicate with personal computers.

Many enthusiasts get stuck in the early stages of their efforts to learn programming in spite of the trend and their willingness. This is due to various reasons. Firstly, even though programming languages, like Basic, were designed initially for simplicity of use, are now complex tools that present the beginner with formidable starting hurtles. Secondly, manuals and text books on the subject lose their readers in the early chapters by delving too deeply into the workings of the language.

This series of lessons will get you started with hands on practical experience immediately without long-winded jargon filled talk. We will try to explain things in the simplest possible way, focusing on the essential and the important, through plenty of examples relevant to electronics.

Even though Visual Basic is now a very powerful and complex language, it is also easy to use, provided one is led through the early stages. The built-in power of Visual Basic will enable you to write simple programs that produce professionally-looking

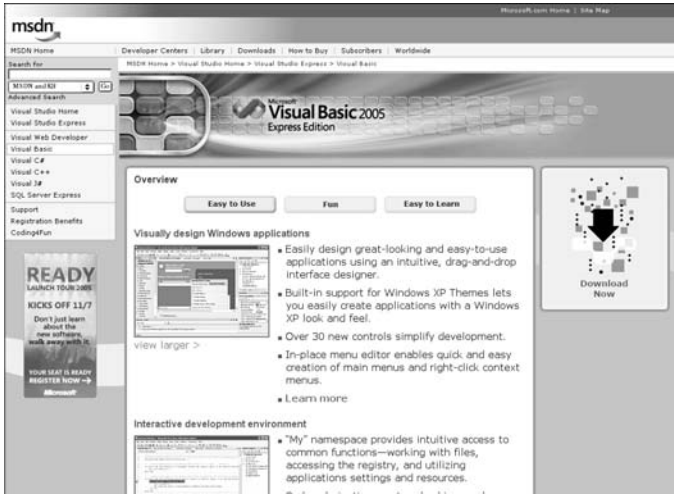
results that would seem unimaginable to those whose experience goes only as far as the early versions of Basic of some years ago.
Visual Basic Net is the latest version of Visual Basic developed by Microsoft and enables the user to produce Windows and Web applications.

The example programs presented in this course were developed using Microsoft Visual Basic Express (beta 2 version). A later version of the program may exist at the time of publishing this booklet. Visual Basic Express may be downloaded from
http://msdn.microsoft.com/vstudio/express/vb/default.aspx
Having downloaded and installed the software on your hard drive you have 30 days to register the product. To do this go to **Help → Activate** Product and follow the instructions.

The program files for the examples presented in the course are available for free downloading from the Publishers' website at www.elektor-electronics.co.uk. The files are accessible from **Magazine → January 2006 → Visual Basic 2005**.

Free booklet with Elektor Electronics January 2006
ISSN 0268/4518

Printed in the Netherlands



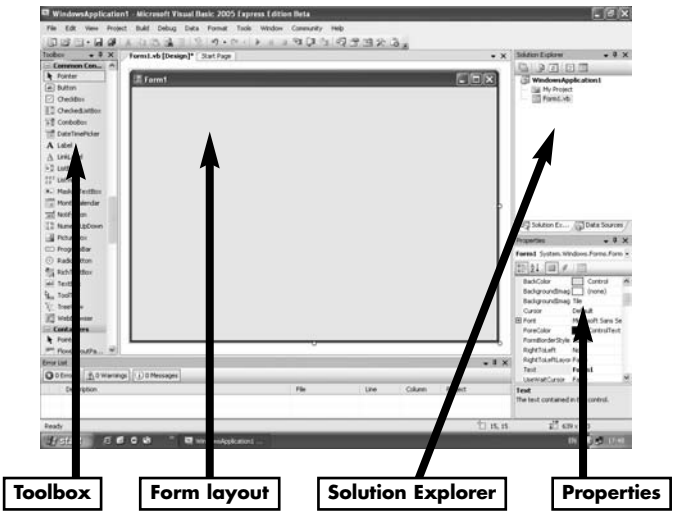
Starting Visual Basic

After installation start the language by double clicking the shortcut icon or click:

- **Start → Visual Basic 2005 Express**
  (The start up screen is presented)
- **File → New Project → Windows Application**
- Type in the name of the project. The path where the project will be stored will be asked by the system when you first save the project.

The Important parts of the Visual Basic development screen

The following screen (called the Integrated Development Environment, or IDE in short) should now appear on your screen:



- The **Toolbox** contains controls that we can place on the form e.g. text boxes containing text or numbers.
- The **Form** is what is presented to the user when the program is run and contains the controls placed on it by the programmer.
- The **Solution Explorer window** shows the files created by the program.
- The **Properties window** contains information about each item on the form and the form itself.

Notes.

- If for some reason any of the above windows are missing from your screen then from the menu select:
  View → Toolbox
  View → Solution Explorer, or
  View → Properties Window
- To bring into view the Form double click the Form file in the Solution Explorer window.

Our first program.

We will introduce Visual Basic programming through a simple example.

Program title: Ohm's law.

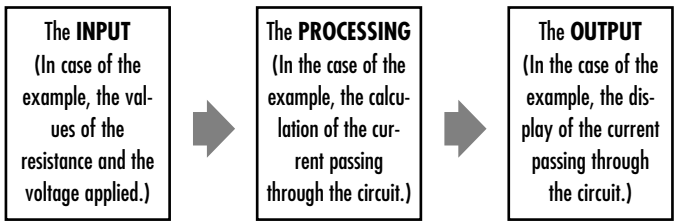
Program specification:

1. Program to display a simple circuit with a resistance R and a voltage V applied across it.
2. The user to be allowed to enter the values of the resistance and the voltage applied.
3. The program to calculate and display the current flowing through the circuit.

Program operation:

You will notice from the program specification that for the pro-

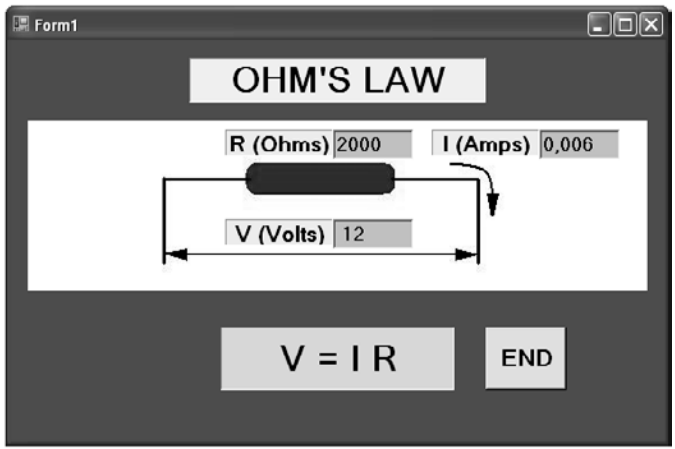
gram to work, three important things must be defined as shown below. Note that this is a general requirement for any type of program.



Writing Visual Basic programs involves the following:

- Picking objects or controls from the toolbox and placing them on the form.
- Sizing and arranging the objects in the required position.
- Changing the properties of objects as required.
- Attaching code to objects or controls.

The screen below shows one possible arrangement that satisfies the program specification.

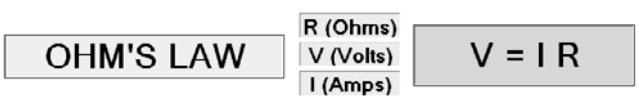


The objects or controls on the form

Picture box:
Picture boxes are controls that enable the programmer to place pictures on the form.

- Use a program like Microsoft Paint to draw a picture like the one above and save it.
- Pick a Picture box control from the toolbox and place it on the form and size it to suit.
- Select the picture box by clicking on it once. In the properties window click on the button with the three dots next to the BackgroundImage property.
- Select the file which contains your picture and click Open.
- Adjust, position and size the picture box so that the image is completely visible.

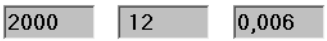
Labels:
Labels contain text and are usually placed on a form to help the user to understand the contents of the form. (e.g. the title of the program). Labels can also be used by the program to display data.



Pick labels from the toolbox and place five of them on the form. Select each one in turn and change their properties as shown in the table below.
Some controls like labels have their AutoSize property set to True by default. This means that they will adjust in size automatically as text is typed into their text property. To adjust their size manually set the AutoSize property to False.

Control	Property	Setting
Label1	Text	OHM'S LAW
	Font	Size 24, Bold
	BackColor	Choose from palette
	TextAlign	Middle centre
Label2	Text	R (Ohms)
	Font	Size 14, Bold
	BackColor	Choose from palette
	TextAlign	Middle centre
Label3	Text	V (Volts)
	Font	Size 14, Bold
	BackColor	Choose from palette
	TextAlign	Middle centre
Label4	Label	I (Amps)
	Font	Size 14, Bold
	BackColor	Choose from palette
	TextAlign	Middle centre
Label5	Label	V=IR
	Font	Size 24, Bold
	BackColor	Choose from palette
	TextAlign	Middle centre

Text boxes:
Text boxes are controls that enable the program to display results but also allow the user to input data.



Pick three text boxes from the toolbox and place them on the form next to the appropriate labels. Select each one in turn and change their properties as shown in the table below.

Control	Property	Setting
Text Box	Name	txtR
	Font	Size 12, Bold
	BackColor	Choose from palette
	Text	R
Text Box	Name	txtV
	Font	Size 12, Bold
	BackColor	Choose from palette
	Text	V
Text Box	Name	txtI
	Font	Size 12, Bold
	BackColor	Choose from palette
	Text	I
	Read Only	True

The property **ReadOnly** for the last text box is set to **True** so that the user will not be allowed to enter values in this text box. It will be used only for outputting the result.

Command buttons:
Command buttons are controls whose function is to execute a command or a series of commands when clicked. In our example the program will end when the button END is clicked.

Click once on the control and change the following properties:

Control	Property	Setting
Button	Name	btnEND
	Font	Size 16, Bold
	BackColor	Choose from palette
	Text	END
	TextAlign	Middle centre

Attaching code to the controls

Now that all the controls are in place on the form, we are ready to attach code to them.

1. Double click the END command button. (The code window will open ready for you to insert command lines within the subroutine start and end statements which are automatically written for you).

2. Simply type in the command **End** as shown highlighted below.

```
Private Sub btnEnd_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnEnd.Click

        End

   End Sub
```

3. Return to the form by clicking on the View designer button above the Solution Explorer window or double clicking the Form1 in the Solution Explorer window.

4. Double click the **txtR** text box (next to the R(Ohms) label). Type in the following (highlighted) code inside the subroutine.

```
Private Sub txtR_TextChanged(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
    txtR.TextChanged

        txtI.Text = Val(txtV.Text) / Val(txtR.Text)

End Sub
```

5. Double click the **txtV** text box (next to the V(Volts) label). Type in the same (highlighted) code inside the subroutine.

```
Private Sub txtV_TextChanged(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
    txtV.TextChanged

        txtI.Text = Val(txtV.Text) / Val(txtR.Text)

    End Sub
```

Notes on the code:

- Notice the first line of the subroutine in paragraph 2 above.

  (Private Sub btnEnd_Click)

  "**Click**" is the event which will trigger the execution of this subroutine when the command button END is clicked at RUN time.

  Visual Basic is said to be an **"event driven language"** because it is designed to respond to **events**.

  Visual Basic is also said to be an **"Object Oriented**

Programming language", or **OOP** in short, because it treats everything as **objects** which have properties that can be changed using predefined methods.

The **End** command is obvious. It simply means, end the program execution.

- Notice now the first line of the subroutine in paragraph 4 above.

```
(Private Sub txtR_TextChanged)
```

TextChanged is the event that will trigger the execution of this subroutine. This means that every time the user types something inside the text box "txtR", then the command line **txtI.Text = Val(txtV.Text) / Val(txtR.Text)** will be executed. This statement means, change the text property of the text box "txtI " (which is the current flowing through the circuit) with the result of the division to the right of the equal sign. Val is a function which takes the numeric value of the text inside the text boxes "txtV" and "txt R"

- The same instruction is executed if the user changes the text in the Voltage text box "txtV" as you can see from paragraph 5 above.

- The events that are displayed when you double click controls on the form are the default events. You can change the type of event to suit your program.

Saving the program

It is good practice to save your work regularly, as Visual Basic does not save it automatically for you. You can save by selecting **File → Save All** from the menu. Visual Basic creates many files for your project. Always save just before running your program in case the program gets stuck.

Running the program

With all the controls in place and the code attached to them, we are ready to run the program. You can do this by clicking the Start button next to the Debug box in the toolbar or by selecting **Debug → Start** from the menu, or by pressing the function key F5 on the keyboard. When you run the program you should see the form you designed open up on the screen. Go into the resistance text box and type in a value for the resistance. Immediately you will see the value 0 displayed in the current text box. This is because you have not yet placed any value for the voltage, so the computer considers the voltage to be 0. Now if you go into the voltage box and type a value, the computer will calculate the current and display it in the "current" text box. Also note that if you give a zero value for the resistance then the current goes to infinity. Even if you give a value of zero to both the resistance and the voltage the program will not stop with an error message (division 0 by 0) but it will place a NaN message in the Current text box and continue running.

If there are syntax errors the system will warn you. Once you run the program successfully Visual Basic Net creates an executable (exe) file which you can use to run the program directly from Windows without having to load the language. You can find this file with the extension .exe if you dig into the path where the project is saved. If you don't specify a different path, this will be found in:

C:/My documents/Visual Studio 2005/Projects/ project name/Bin/Debug/project name.exe

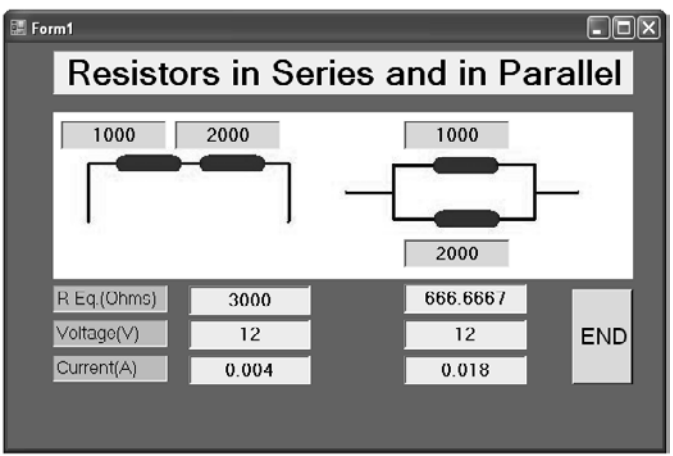
Now on to our second program

In this program we shall introduce Variables.

Program title: Resistors in series and in parallel.

Program specification:

1. Program to show simple circuits of resistors in series and in parallel.
2. The user to be allowed to enter the values of the resistors and the voltage applied.
3. The program to calculate the equivalent resistance of the circuit and the current passing through it.



The **INPUT**

(In the case of the example, the values of the resistance and the voltage applied.)

Place the following controls on the form and size them to suit.

- A label for the title of the program.
- A picture box showing the resistors in series and in parallel which you can draw using Microsoft Paint.
- Four text boxes placed next to the resistors which will be used to enter the values of the resistances.
- Three labels R Eq.(Oms), Voltage (V), and Current (A).
- Next to the above labels, three text boxes for the R equivalent, Voltage and Current for the series circuit and another set of three for the parallel circuit.
- Finally a command button END to end the program.

Select each one of the above controls in turn and change their properties as shown in the table below. We are omitting the font and background colour properties to save space. Change these to your liking.

Control	Property	Setting
Label 1	Text	Resistors in Series and in Parallel
Label 2	Text	R Eq. (Ohms)
Label 3	Text	Voltage (V)
Label 4	Text	Current (I)
Text Box 1	Name	txtSeriesR1
	Text	R1
	TextAlign	Centre
Text Box 2	Name	txtSeriesR2
	Text	R2
	TextAlign	Centre
Text Box 3	Name	txtParallR1
	Text	R1
	TextAlign	Centre
Text Box 4	Name	txtParallR2
	Text	R2
	TextAlign	Centre
Text Box 5	Name	txtRSeriesEquiv
	Text	R Equivalent
	TextAlign	Centre
	ReadOnly	True
Text Box 6	Name	txtRSeriesVolts
	Text	Voltage
	TextAlign	Centre
Text Box 7	Name	txtRSeriesCurrent
	Text	Current
	TextAlign	Centre
	ReadOnly	True

Control	Property	Setting
Text Box 8	Name	txtRParallEquiv
	Text	R Equivalent
	TextAlign	Centre
	ReadOnly	True
Text Box 9	Name	txtRParallVolts
	Text	Voltage
	TextAlign	Centre
Text Box 10	Name	txtRParallCurrent
	Text	Current
	TextAlign	Centre
	ReadOnly	True
Button1	Name	btnEnd
	Text	END
	TextAlign	MiddleCentre

Attach the following code to the appropriate controls by double clicking on each control in turn and typing the code inside the subroutines.

Code attached to button END

```
1     Private Sub Button1_Click(ByVal sender As
              System.Object, ByVal e As System.EventArgs)
                                    Handles Button1.Click
2         End
3     End Sub
```

Code attached to text box txtSeriesR1

```
4     Private Sub txtSeriesR1_TextChanged(ByVal sender As
                System.Object, ByVal e As System.EventArgs)
                             Handles txtSeriesR1.TextChanged
5         Dim REquivalent, Current As Single
6         REquivalent = Val(txtSeriesR1.Text) +
                                        Val(txtSeriesR2.Text)
7         txtRSeriesEquiv.Text = REquivalent
8         Current = Val(txtSeriesVolts.Text) /
                                    Val(txtRSeriesEquiv.Text)
9         txtSeriesCurrent.Text = Current
10    End Sub
```

Code attached to text box txtSeriesVolts

```
11    Private Sub txtSeriesVolts_TextChanged(ByVal sender As
              System.Object, ByVal e As  System.EventArgs)
                          Handles txtSeriesVolts.TextChanged
12        Dim Current As Single
13        Current = Val(txtSeriesVolts.Text) /
                                    Val(txtRSeriesEquiv.Text)
14        txtSeriesCurrent.Text = Current
15    End Sub
```

Code attached to text box txtSeriesR2

```
16    Private Sub txtSeriesR2_TextChanged(ByVal sender As
                System.Object, ByVal e As System.EventArgs)
                             Handles txtSeriesR2.TextChanged
17        Dim REquivalent, Current As Single
18        REquivalent = Val(txtSeriesR1.Text) +
                                        Val(txtSeriesR2.Text)
19        txtRSeriesEquiv.Text = REquivalent
20        Current = Val(txtSeriesVolts.Text) /
                                    Val(txtRSeriesEquiv.Text)
21        txtSeriesCurrent.Text = Current
22    End Sub
```

Code attached to text box txtParallR1

```
23 Private Sub txtParallR1_TextChanged(ByVal sender As
       System.Object, ByVal e As System.EventArgs)
                           Handles txtParallR1.TextChanged
24     Dim REquivalent, Current As Single
25     REquivalent = (Val(txtParallR1.Text) *
         Val(txtParallR2.Text)) / (Val(txtParallR1.Text) +
                                   Val(txtParallR2.Text))
26     txtRParallEquiv.Text = REquivalent
27     Current = Val(txtParallVolts.Text) /
                                  Val(txtRParallEquiv.Text)
28     txtParallCurrent.Text = Current
29 End Sub
```

Code attached to text box txtParallR2

```
30 Private Sub txtParallR2_TextChanged(ByVal sender As
              System.Object, ByVal e As System.EventArgs)
                           Handles txtParallR2.TextChanged
31     Dim REquivalent, Current As Single
32     REquivalent = (Val(txtParallR1.Text) *
           Val(txtParallR2.Text)) / (Val(txtParallR1.Text)
                                    + Val(txtParallR2.Text))
33     txtRParallEquiv.Text = REquivalent
34     Current = Val(txtParallVolts.Text) /
                                  Val(txtRParallEquiv.Text)
35     txtParallCurrent.Text = Current
36 End Sub
```

Code attached to text box txtParallVolts

```
37 Private Sub txtParallVolts_TextChanged(ByVal sender As
              System.Object, ByVal e As System.EventArgs)
                        Handles txtParallVolts.TextChanged
38     Dim Current As Single
39     Current = Val(txtParallVolts.Text) /
                                  Val(txtRParallEquiv.Text)
40     txtParallCurrent.Text = Current
41 End Sub
```

Notes on the code:

• Dimensioning of variables. Refer to line 5 of the code above. Variables are objects which hold data in the memory of the computer. You the programmer can choose the names of variables. **REquivalent** and **Current** are variables which are declared by the Dim (Dimension) statement to be of type Single. These are decimal numbers of single precision. Other types of variables are Integers, String, Date etc.

• Line 6 calculates the value of the equivalent resistance of the series circuit and line 7 places the result into the text box txtRSeriesEquiv. In the same way lines 8 and 9 calculate the Current and the result is placed in the txtSeriesCurrent text box. In effect what is happening here is that the program is changing the text property of the text boxes on the form. When you type the dot after the text box name, Visual Basic opens a window with all the properties available. You choose the one which is applicable.

• Notice that variables are declared in each subroutine as their values are valid only within the subroutine in which they have been declared. See lines 12,17,24,31 and 38.

Now you can run the code by clicking the Start Debugging button to test the program.

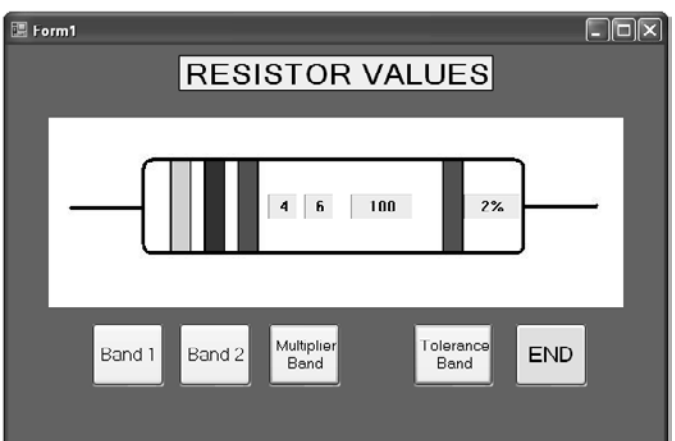
Our third program

This program will be useful for displaying the values of resistors according to their colour code. In the process we are also going to introduce some neat tricks with colours.

Program title: Resistor Values

Program specification:

1. Program to present the user with a picture of a resistor,
2. The user will be prompted to select the colour of the bands on the resistor,
3. Once this is done the program will calculate the resistor value and present it to the user.



Place the following controls on the form and size them to suit.

- A label for the title of the program,
- A picture box for the picture of the resistor,
- Four labels for the colour bands of the resistor,
- Five command buttons for the control of the colour bands including the END.

Select each one of the above controls in turn and change their properties as shown in the table below.

Control	Property	Setting
Label 1	Text	Resistor Values
	Font	Font size 20
	Back Colour	Choose from palette
	Text align	Middle Centre
Label 2	Name	lblBand1
	Text	Empty (Space)
	Back Colour	Red
	AutoSize	False
	Location and Size	Adjust to suit
	Size	Adjust to suit
Label 3	Name	lblBand2
	Text	Empty (Space)
	Back Colour	Red
	AutoSize	False
	Location and Size	Adjust to suit

Control	Property	Setting
Label 4	Name	lblBand3
	Text	Empty (Space)
	Back Colour	Red
	AutoSize	False
	Location and Size	Adjust to suit
Label 5	Name	lblBand4
	Text	Empty (Space)
	Back Colour	Red
	AutoSize	False
	Location and Size	Adjust to suit
Label 6	Name	lblValue1
	Text	Empty (Space)
	Back Colour	Select from palette
	AutoSize	False
	Location and Size	Adjust to suit
Label 7	Name	lblValue2
	Text	Empty (Space)
	Back Colour	Select from palette
	AutoSize	False
	Location and Size	Adjust to suit
Label 8	Name	lblMultiplier
	Text	Empty (Space)
	Back Colour	Select from palette
	AutoSize	False
	Location and Size	Adjust to suit
Label 9	Name	lblTolerance
	Text	Empty (Space)
	Back Colour	Select from palette
	AutoSize	False
	Location and Size	Adjust to suit
Button 1	Name	btnBand1
	Text	Band 1
	TextAlign	Middle Centre
Button 2	Name	btnBand2
	Text	Band 2
	TextAlign	Middle Centre
Button 3	Name	btnBand3
	Text	Multiplier Band
	TextAlign	Middle Centre
Button 4	Name	btnBand4
	Text	Tolerance Band
	TextAlign	Middle Centre
Button 5	Name	btnEnd
	Text	END
	TextAlign	Middle Centre

Attach the following code to the Band 1 button by double clicking the control and typing the code inside the subroutine.

```
1   Private Sub btnBand1_Click(ByVal sender As
       System.Object, ByVal e As System.EventArgs)
                                    Handles btnBand1.Click
```

```
2  Static number As Integer
3     number = number + 1
4     If number > 10 Then number = 1
5     Select Case number
6        Case 1
7           lblBand1.BackColor = Color.Black
8           lblValue1.Text = "0"
9        Case 2
10          lblBand1.BackColor = Color.Brown
11          lblValue1.Text = "1"
12       Case 3
13          lblBand1.BackColor = Color.Red
14          lblValue1.Text = "2"
15       Case 4
16          lblBand1.BackColor = Color.Orange
17          lblValue1.Text = "3"
18       Case 5
19          lblBand1.BackColor = Color.Yellow
20          lblValue1.Text = "4"
21       Case 6
22          lblBand1.BackColor = Color.Green
23          lblValue1.Text = "5"
24       Case 7
25          lblBand1.BackColor = Color.Blue
26          lblValue1.Text = "6"
27       Case 8
28          lblBand1.BackColor = Color.Violet
29          lblValue1.Text = "7"
30       Case 9
31          lblBand1.BackColor = Color.Gray
32          lblValue1.Text = "8"
33       Case 10
34          lblBand1.BackColor = Color.White
35          lblValue1.Text = "9"
36    End Selec
37 End Sub
```

- In line 2 the variable ***number*** is declared as Integer with the Static statement and not with the Dim statement. Static means that the value of the ***number*** will be retained even after exiting the subroutine. In this way the colour change of each band will be in sequence even if the user clicks the band buttons at random.

- The statement of line 3 increases the value of the variable ***number*** by one every time the button is clicked.

- Line 4 is a conditional statement. As we only have 10 different band colours we don't want the value of the variable ***number*** to increase above the number 10. The If statement simply resets its value to 1 every time that happens.

- Lines 5 to 36 contain another kind of conditional statement. It is the Select Case statement. The condition is the value of the variable ***number*** and you can see for example that if the value is 6 then the program will branch to execute the statements of lines 22 and 23. Line 22 sets the BackColor property of label lblBand1 to Green and line 23 places the figure 5 in the label lblValue1.

- Attach the same code to command button btnBand2 but replace lblBand1 with lblBand2 and lblValue1 with lblValue2. You can use Copy/Paste.

Attach the following code to command button btnMultiplier. The code is similar except for the difference in the colours and the reference to the labels lblBand3 and lblMultiplier.

```
Private Sub btnMultiplier_Click(ByVal sender As
              System.Object, ByVal e As System.EventArgs)
                          Handles btnMultiplier.Click
    Static number As Integer
    number = number + 1
    If number > 11 Then number = 1
    Select Case number
        Case 1
            lblBand3.BackColor = Color.Silver
            lblMultiplier.Text = "0.01"
        Case 2
            lblBand3.BackColor = Color.Gold
            lblMultiplier.Text = "0.1"
        Case 3
            lblBand3.BackColor = Color.Black
            lblMultiplier.Text = "1"
        Case 4
            lblBand3.BackColor = Color.Brown
            lblMultiplier.Text = "10"
        Case 5
            lblBand3.BackColor = Color.Red
            lblMultiplier.Text = "100"
        Case 6
            lblBand3.BackColor = Color.Orange
            lblMultiplier.Text = "1k"
        Case 7
            lblBand3.BackColor = Color.Yellow
            lblMultiplier.Text = "10k"
        Case 8
            lblBand3.BackColor = Color.Green
            lblMultiplier.Text = "100k"
        Case 9
            lblBand3.BackColor = Color.Blue
            lblMultiplier.Text = "1M"
        Case 10
            lblBand3.BackColor = Color.Violet
            lblMultiplier.Text = "10M"
        Case 11
            lblBand3.BackColor = Color.Gray
            lblMultiplier.Text = "100M"
    End Select
End Sub
```

Finally attach the following code to the Command button btnTolerance:

```
Private Sub btnTolerance_Click(ByVal sender As
              System.Object, ByVal e As System.EventArgs)
                              Handles btnTolerance.Click
Static number As Integer
number = number + 1
If number > 7 Then number = 1
    Select Case number
        Case 1
            lblBand4.BackColor = Color.Silver
            lblTolerance.Text = "10%"
        Case 2
            lblBand4.BackColor = Color.Gold
            lblTolerance.Text = "5%"
        Case 3
            lblBand4.BackColor = Color.Brown
            lblTolerance.Text = "1%"
        Case 4
            lblBand4.BackColor = Color.Red
            lblTolerance.Text = "2%"
        Case 5
            lblBand4.BackColor = Color.Green
            lblTolerance.Text = "0.5%"
```

```
      Case 6
         lblBand4.BackColor = Color.Blue
         lblTolerance.Text = "0.25%"
      Case 7
         lblBand4.BackColor = Color.Violet
         lblTolerance.Text = "0.1%"
      End Select
End Sub
```

You are now ready to test the program. When clicking the Band buttons you should see the colours of the bands changing and the values of the resistance displayed in the labels.

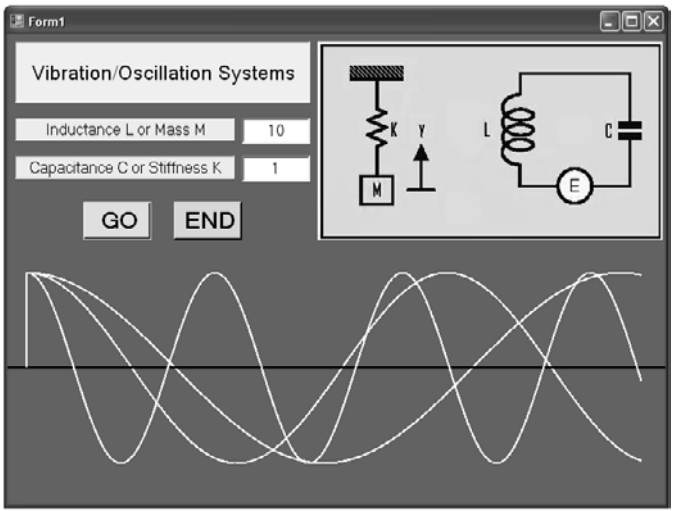
A program to display graphics

In this program we are going to see how an oscillating circuit (or its mechanical equivalent vibrating system) behaves by plotting the results on the form using graphics.

Program title: Vibration/Oscillation Systems

Program specification:

1. Program to show an LC circuit and its mechanical equivalent spring mass system.
2. The user will be prompted to input values for L and C (or K and M),



3. The program will calculate the behaviour of the system and to display the result in graphical form.

Place the following controls on the form and size them to suit.

- A label for the title of the program,
- A picture box for the picture of the LC circuit and the Spring Mass system,
- Two labels for the values of the Inductance and the Capacitance,
- Two text boxes for the input of the above values,
- Two command buttons for GO and END.

Select each one of the above controls in turn and change their properties as shown in the table below.

Control	Property	Setting
Label 1	Text	Vibration/ Oscillation systems
	Font	Font size 15
	Back Colour	Choose from palette
	Text align	Middle Centre
Label 2	Text	Inductance L or Mass M
	Font	Font size 12
	Back Colour	Choose from palette
	Text align	Middle Centre
Label 3	Text	Capacitance C or Stiffness K
	Font	Font size 12
	Back Colour	Choose from palette
	Text align	Middle Centre
Text Box 1	Name	txtL
	Text	1
	TextAlign	Centre
Text Box 2	Name	txtC
	Text	1
	TextAlign	Centre
Button 1	Name	btnGo
	Text	GO
	TextAlign	Middle Centre
Button 2	Name	btnEnd
	Text	END
	TextAlign	Middle Centre

Attach the following code to the command button GO:

```
1  Private Sub cmdGo_Click(ByVal sender As System.Object,
                           ByVal e1 As System.EventArgs)
                                    Handles cmdGo.Click
2      'Declare variables
3      Dim i, j, x, y, x1, y1, shiftX, shiftY,
                                        omega As Single
4      Dim myGraphics As Graphics
5      Dim myPen As New Pen(Color.White, 2)
```

```
6     myGraphics = Graphics.FromHwnd
                              (hwnd:=ActiveForm().Handle)
7     'shift the position of the graph on the form
8     shiftX = 20
9     shiftY = 350
10    x1 = 0 + shiftX
11    y1 = 0 + shiftY
12    'calculate the frequency of the vibration
13    omega = Sqrt(Val(txtC.text) / Val(txtL.text))
14    'draw axis
15    myPen.Color = Color.Black
16    myGraphics.DrawLine(myPen, 0, shiftY, 700, shiftY)
17    'loop to increment the value of x
                             and calculate the value of y
18    For x = 0 To 80 Step 0.2
19       y = -(Cos(x * omega) * 100)
20       myPen.Color = Color.White
21       myGraphics.DrawLine(myPen, x1, y1,
                        (x * 10 + shiftX), (y + shiftY))
22       'save the previous plot position in x1 and x2
23       x1 = x * 10 + shiftX
24       y1 = y + shiftY
25       If x * 10 > 650 Then Exit For
26       'Delay loop
27       For i = 1 To 200000
28          j = i * i
29       Next
30    Next
31 End Sub
```

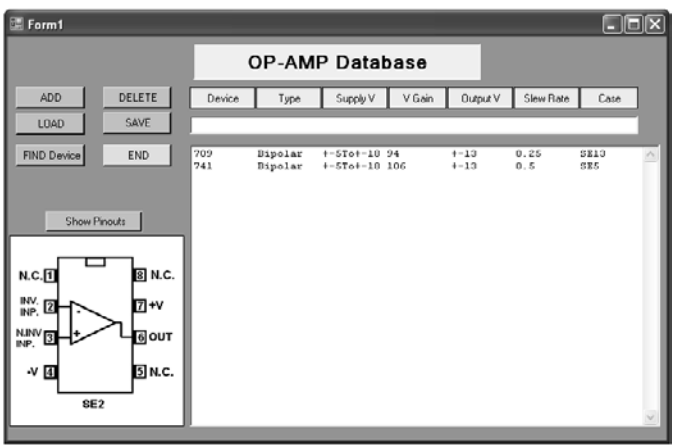
- In lines 2 to 6 we declare the variables and prepare the graphics environment to draw lines on the form. Do not worry if you do not understand some of the statements. (Line 2 is a comment placed to help one to understand the logic behind the code. It is not an executable instruction).
- In lines 8 and 9 we define shift values so that the graph is drawn in the lower half of the form. The origin of the form is at the top left corner.
- In line 13 the frequency of the oscillation is calculated. Sqrt means the square root.
- In lines 15 and 16 we draw a horizontal line in black from left to right.
- Between lines 18 and 30 we program what is called a LOOP. For x = 0 To 80 Step 0.2 means start with a zero value for x and increment by 0.2 until the value of x reaches 80. Each time all the statements which are between lines 18 and 30 will be executed.
- In line 19 the value of y is calculated. This may represent the voltage.
- Lines 20 and 21 draw a line from the previous plot position (x1,y1) to the new calculated plot position (x, y) in colour white. Notice that the value of x is scaled by a factor of 10 to make the wave visible.
- In lines 23 and 24 the values of x and y are placed in x1 and y1 to become the previous plot position for the next loop.
- In line 25 we have a conditional statement which will stop the loop if the plot position reaches the right hand end of the form.
- In lines 27 to 29 we program a delay loop with a dummy calculation inside it to delay the computer so the user can see the wave moving. You can change the value of 200000 according to how fast your computer is. Notice that this is a loop within another loop and is called a nested loop. Every time the outer loop is executed, the inside loop is executed 200000 times.

- Finally we have not programmed for the clearing of the form to start a new plot. However you will find that if you minimise the form and maximise it back the form is cleared. Something which in other cases would be undesirable in this case it is useful.

A program to create files

In this program we are going to build a database containing information about OP-Amps.

Program title: OP-Amp Database



Program specification:

Program to allow the user:

1 To input information about OP-Amps,
2 To add, edit and delete records,
3 To search the database for a particular devise,
4 To display the pin-out diagrams of the devices,
5 To save and load the database records.

Place the following controls on the form and size them to suit.

- Labels for the title of the program and the headings above the op-amp records,
- Command buttons for the ADD, DELETE, LOAD, SAVE, FIND Device, END and Show Pinouts.
- A text box for the input of record data,
- A Label for the display of the pinout diagram,
- A list box into which the database records will be held and displayed.

Select each one of the above controls in turn and change their properties as shown in the table below. (Only the important objects and properties are shown).

Control	Property	Setting
Label	Name	lblPinouts
	Image	Select from Resources
	Image align	Middle Centre
Button 1	Name	btnAdd
	Text	ADD

Control	Property	Setting
Button 2	Name	btnDelete
	Text	DELETE
Button 3	Name	btnLoad
	Text	LOAD
Button 4	Name	btnSave
	Text	SAVE
Button 5	Name	btnFind
	Text	FIND Device
Button 6	Name	btnPinouts
	Text	Show Pinouts
Text Box	Name	txtDevice
	Text	Empty
List Box	Name	lstDevice
	Items	Enter some initial data

Attach the following code to the appropriate controls by double clicking on each control in turn and typing the code inside the subroutines.

Code attached to command button END

```
Private Sub btnEnd_Click(ByVal sender As Object,
            e As System.EventArgs) Handles btnEnd.Click
        End
  End Sub
```

Code attached to command button ADD

```
Private Sub btnAdd_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles btnAdd.Click
   'Add text box content to list box content
   lstDevice.Items.Add(txtDevice.Text)
   'Clear text box and place cursor in it ready for
                                          next item input
   txtDevice.Text = ""
   txtDevice.Select()
End Sub
```

Code attached to command button DELETE

```
Private Sub btnDelete_Click(ByVal sender As Object,
     ByVal e As System.EventArgs) Handles btnDelete.Click
   'Delete the selected item from the list box
   lstDevice.Items.Remove(lstDevice.SelectedItem)
End Sub
```

Code attached to command button LOAD

```
Private Sub btnLoad_Click(ByVal sender As System.Object,
       ByVal e As System.EventArgs) Handles btnLoad.Click
   'Declare variables and open file dialogue
                                            to select file
   Dim i, count As Integer
   OpenFileDialog1.ShowDialog()
   Dim sr As New System.IO.StreamReader
                                (OpenFileDialog1.FileName)
   'Read first item which should be the number
                                      of items in the file
   count = Val(sr.ReadLine)
   'Loop to read the records in the file and
                                       add to the list box
   For i = 0 To count - 1
      lstDevice.Items.Add(sr.ReadLine)
```

```
    Next i
    'Close the file
    sr.Close()
End Sub
```

Code attached to command button SAVE

```
Private Sub btnSave_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles btnSave.Click
    'Declare variables and open file dialogue
                                              to select file
    Dim i, count As Integer
    SaveFileDialog1.ShowDialog()
    Dim sw As New System.IO.StreamWriter
                                (SaveFileDialog1.FileName)
    'Write the number of items in the file
    count = lstDevice.Items.Count
    sw.WriteLine(count)
    'Loop to select records from the list box
                         and write the records in the file
    For i = 0 To count - 1
        lstDevice.SelectedIndex = i
        sw.WriteLine(lstDevice.Items.Item(i))
    Next i
    'Close the file
    sw.Close()
    End Sub
```

Code attached to command button FIND

```
Private Sub btnFind_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles btnFind.Click
    'Declare variables
    Dim DeviceWanted, DeviceListed, DeviceRecord As String
    Dim count, i As Integer
    'Function Trim removes spaces from the string
    DeviceWanted = Trim(txtDevice.Text)
    'Find the number of items in the list box
    count = lstDevice.Items.Count
    'Loop to compare the records in the list box
                                    with the wanted device
    For i = 0 To count - 1
        lstDevice.SelectedIndex = i
        DeviceRecord = lstDevice.SelectedItem
        DeviceListed = Trim(Microsoft.VisualBasic.Left
                                        (DeviceRecord, 9))
        'If found place the record in the text box
                                      and display message
        If DeviceWanted = DeviceListed Then
            txtDevice.Text = lstDevice.Items.Item(i)
            MsgBox("Device found!")
            'If found exit search
            GoTo finish
        End If
    Next i
    'If device is not found display message
    txtDevice.Text = ""
    MsgBox("Device not found")
finish:
End Sub
```

Code attached to command button Show Pinouts

```
Private Sub btnPinouts_Click(ByVal sender As
              System.Object, ByVal e As System.EventArgs)
                                Handles btnPinouts.Click
    'Declare counter j as static integer
    Static j As Integer
    'Display the appropriate image in the label Pinouts
    Select Case j
    Case 1
        lblPinouts.Image = My.Resources.SE1_OP_AMP_PIC
    Case 2
```

```
        lblPinouts.Image = My.Resources.SE2_OP_AMP_PIC
    Case 3
        lblPinouts.Image = My.Resources.SE3_OP_AMP_PIC
    End Select
    'increment counter
    j = j + 1
    'if value of couter j reaches the last number,
                                                reset to 1
    If j = 4 Then j = 1
End Sub
```

The code should be clear given the comments. The operation of the program is also simple.

- To add a record type the data in the text box and click the ADD button.
- To delete a record select it by clicking on it in the list box and then click the DELETE button.
- To load data from a file click the LOAD button. Select the file and click OPEN. Note that the records are added to the existing records in the list box.
- To save the records in a file click the SAVE button.
- To search the database for a particular device type the designation of the device in the text box (e.g. 741) and click the Find Device button.
- To show the pinout diagram click repeatedly the Show Pinouts button until you find the appropriate diagram.

Placing additional pinout diagrams.

The pinout diagrams will of course have to be drawn using a program like Microsoft Paint. Alternatively if these diagrams can be found from manufacturer's manuals then they can be scanned and saved, each one as a separate file. Once you have the diagrams stored in files then follow the instructions below to add the diagrams in the resources of the program. This has to be done at program design time:

- Double click ***My Projects*** in the Solution Explorer window,
- Click the ***resources*** tab in the window opened in the main area,
- Click the triangle (arrow) to the right of the ***Add Resources*** button,
- Select ***Add Existing File***,
- In the code for the ***Show Pinouts*** button add the additional code for the diagrams being added. (do not forget to change the value of the counter ***j*** for the total number of diagrams in the resources, last line before the ***End Sub***).

Executable files

Once you have debugged the program and you are satisfied that it works, you do not want to go through the process of loading Visual Basic Express every time you want to run the program. When you debug and save your program Visual Basic compiles your source code and creates an executable (.exe) file for you. You can find this file if you dig into the directories. This file can then be copied (on your desktop for example) and run by simply double clicking on it. The executable file can be found under the directories where you have saved your program e.g.

\OP-AMP-Data\Bin\OP-AMP-Data.exe

Please note that executable files may not run on another machine that has not had the NET framework installed on it.

The last program we have looked at shows the power of Visual Basic 2005. With less than two pages of code we have implemented a program that creates a database with facilities to Add, Delete, Edit, Save, Load, Search and Display diagrams.

In this short course on Visual Basic programming you have learned to write programs which:

- allow the user to input data,
- process the input data,
- present the user with results in alphanumeric and in graphical form,
- create files on a mass storage device and store data in them,
- open files to read data from them.

In short you have learned to program the computer to carry out the most important processes that computer programs in general do.