Instrumentation I/O and Visual Basic

David Hergert
Miami University Ohio

1. Introduction

If a college wants to implement a computerized instrumentation system without using commercial software, Visual Basic provides a quick and easy solution. Visual Basic is capable of creating ActiveX controls to display data, as well as providing numerous methods to transfer and store data. Data can even be displayed on the web using ActiveX . When developing VB computerized instrumentation systems, the following must be taken into account.

- How the energy is being measured.
- How it will be brought into the computer.
- How it will display data.
- How the data is stored.
- What kind of data transfer will be needed.

Before developing a Visual Basic instrumentation system, the professor must first determine what type of transducer he is interfacing to. Next he will either buy or design the appropriate signal conditioning to bring the signal into the computer. This includes necessary A/D and DSP conversions. To bring the data into the computer, the user has a minimum of three choices:

- Use a commercial plug-in data acquisition board
- Use the serial port
- Use the printer port

While Visual Basic can easily display data, it is more difficult to read in data than QuickBasic. Displaying data can be made a relatively simple process by first creating ActiveX controls. A discussion of how to read through a serial port is first covered, then a description of creating ActiveX controls for displays. Finally a description of how Visual C++ can be used to create a DLL that allows Visual Basic to read from a port.

2. Serial I/O with Visual Basic

Many inexpensive devices exist today that allow the user to read data through the communication port. Serial data acquisition boards can input and output analog or digital signals. In addition, digital multimeters and oscilloscopes often have an RS-232port included. Some digital multimeters also include a thermocouple for temperature measurement. Often this equipment comes with instructions on how to set up data transfer using Microsoft QuickBasic. Reading and writing to a serial port in Visual Basic is done in a similar manner. When QuickBasic attempts to make a connection, the following events happen:

- The Data Terminal Ready (DTR) pin is set high.
- The Request To Send (RS) pin is sent high unless the RS option is included in the OPEN statement. A high indicates that the Request to Send detection will not be checked.
- If the Carrier Detect Option (CD) is specified, the program waits a specific period of time. The default time is 1000 milliseconds.

A simple program that suppresses the Request To Send and times out the Carrier Detect after 1000 ms is shown below. The program prints the string “5” to the COM 1 serial port and waits for an input. The input is stored in a 20 character buffer before being transferred to the memory address pointed to by A$.

```
OPEN “COM1:9600,N,8,1” FOR RANDOM AS #1
PRINT #1,”S”;CHR$(13)
A$=INPUT$(20,#t)
PRINT A$
CLOSE 1
```

The specifications are:

Baud: 9600
Parity: None
Number of bits: 8
Number of stop bits: 1

Implementing RS232 communication in Visual Basic 6.0 requires similar coding. To begin, the Comm Control component must be loaded. To do this:

Select **Project Components**
Select **Microsoft Comm Control 6.0**

This will place a telephone as one of the component icons. Select the telephone and place it on the form. Now suppose a command button is drawn on the screen. The code below replicates the QuickBasic program shown above. It uses polling with no handshaking. The input is placed in the Text1 text box. If event driven or handshaking is required, the VB Help provides some examples of how to implement this.

```
Textl.Text = “”
MSComml.CommPort = 1
‘9600 baud, no parity, 8 data, and 1 stop bit.
MSComml.Settings = “600,n,7,2”
‘Tell the control to read entire buffer when Input is used.

buffer$=””

MSComml.InBufferCount =0
MSComml.OutBufferCount =0
MSComml.InBufferSize =20
MSComml.RTSEnable = False
MSComml.DTREnable = True
MSComml.RThreshold =0
MSComml.SThreshold =0
MSComml.InputLen =0 Open the port.
MSComml.PortOpen True
MSComml.Output = "5" 'Send Command to MBD11
'Wait for data to come back to the serial port.

X = Timer
Do
'Allow the operating system to update
dummy = DoEvents()
If MSComml.InBufferCount Then
Success = True
buffer$ = buffer$ + MSComml.Input
End If

Loop Until Timer >= X + 1

If Success Then
Textl.Text = Buffer$
Else
Textl.Text = "Failed Transmission"
End If
MSComml.PortOpen = False
```

This program begins by defining the communications port as COM 1. The settings are then specified as 9600 baud, no parity, 8 data bits, and 1 stop bit. Next the input and output buffers are flushed (set to 0) and the buffer size is set to 20 characters. Request to Send is disabled with RTSEnable set to False and DTR is enabled with DTR enable set toTrue. The serial port is opened with the PortOpen = True statement. After the string S is sent, a timer is started. If the wait time for an input from the device is greater than 1 second (Timer >= X±1), no further waiting is performed, and the buffer string is empty. If an input is received, it is retrieved from MSCoinml.Input and sent to Buffer$. If no response is received (Success=0), the Text1 box contains "Failed Transmission". Finally the port is closed with the statement MSComml.PortOpen = False.

If more handshaking is needed, the MSComm object can be used. Since the first instance on a form is MSComml, the following code can be used to check for Clear To Send (CTS) or Data Set Ready (DSR).

```
Private Sub MSComin_OnCommO
If MSComml.CommEvent = comEvCTS Then
'Insert code here
End If
If MSComml.CommEvent comEvDSR Then
'Insert code here
```

End If
End Sub

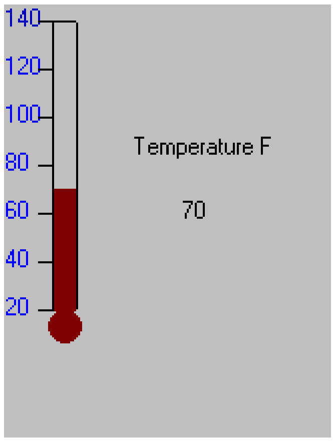
The time between readings in Visual Basic can be somewhat erratic. It also can be rather slow compared with C or QuickBasic. If a high frequency with a constant sample rate needs to be read, a low cost serial data acquisition board with a buffer can be used. The MBD board is capable of reading a *62.5Khz* signal into a buffer and transferring the contents to Visual Basic.

3. Graphic Display

There are VB components that can help display data, such as the slider and progress bar. If a custom gauge (such as a thermometer) needs to be implemented, the Shape control could be used. To fill a display, the LINE Fill command can be used with QuickBasic colors, as shown below:

Line (x, base)-(x + width, y), **QBColor(4), BF**

The example above fills the rectangle with a red color and draws a box from the bottom to the current temperature (represented by y). The base variable represents the pixel location of the bottom of the meter, width is the width of the thermometer. The BF option draws a filled rectangle. Figure 1 shows a sample screen that displays current temperature. The Line command was used to fill with a red color.



4. Active X Display

Perhaps the easiest and most versatile way to display data using meters, charts, etc. is to create ActiveX controls that the student can insert in their VB programs or web pages. In this way, the student is responsible for bringing the data into the PC and analyzing it, but has an array of components available to display data. While it is beyond the scope of this paper to provide a thorough coverage of ActiveX controls, the intent is to show how flexible and powerful they are. Aitken (1999) gives a complete description of implementing ActiveX controls. To create a simple ActiveX control, choose **File/New/ActiveX Control.**

The program below uses an ActiveX control to implement a user-controlled thermometer on the screen. After compiling, it can be inserted into other Visual Basic programs by selecting

PROJECT/COMPONENTS/BROWSE and clicking on the **OCX** filename. The program uses the **Time** icon to increment temperature display events. On its own, this control does not do anything, but when inserted in a VB program, it provides a simple way to display readings.

```
Public x As Integer
Dim y
Dim xprev
Public Property Let Value(ByVal temp As Variant)
Label1.Caption = temp
End Property
Public Property Let Max(ByVal maxm As Variant)
Textl.Text = maxm
End Property
Public Property Let Min(ByVal minm As Variant)
Text2.Text minm
End Property
Private Sub Timer1_TimerO
‘Place scale
Label3.Caption = Text1.Text
Label9.Caption = Text2.Text
Label8.Caption = lnt((Val(Text1.Text) - Val(Text2.Text)) / 6) + Val(Text2.Text)
Label7.Caption = Int((Val(Text1.Text) - Val(Text2.Text)) * 2 / 6) + Val(Text2.Text)
Label6.Caption = Int((Val(Text1.Text) - Val(Text2.Teit)) * 3 / 6) + Val(Text2.Text)
Label5.Caption = Int((Val(Text1.Text) - Val(Text2.Text)) * 4 / 6) + Val(Text2.Text)
Label4.Caption = Int((Val(Text1.Text) - Val(Text2.Text)) * 5 / 6) + Val(Text2.Text)
‘Draw tick marks
Line (y - 8, 152)-(y, 152)
Line (y - 8, 128)-(y, 128)
Line (y - 8, 104)-(y, 104)
Line (y - 8, 80)-(y, 80)
Line (y -8, 56)-(y, 56)
Line (y - 8, 32)-(y, 32)
Line (y - 8,8)-(y, 8)
xprev =x
x = 152 - (Val(Label1.Caption))
‘Set limits
If x <9 Then x =9
If x>152 Then x= 152
‘Define fill position
xx = 152 + ((144 / (Val(Text2.Text) - Val(Text1.Text)) * (Label1.Caption -20)))
‘If fill position has changed, draw new fill position
If xprev <> x Then Line (y, 152)-(y + 10, 9), QBColor(7), BF
Line (y, 152)-(y + 10, xx), QBColor(4), BF
End Sub

Private Sub UserControl_InitializeO
Text2.Text =0
Label3.Caption =0
```

```
Label9.Caption =0
Label8.Caption =0
Label7.Caption =0
Label6.Caption =0
Label5.Caption =0
Label4.Caption =0
Label3.Caption = Text1.Text
Label1.Caption =20
Text1.Text = 140
Text2.Text = 20
'Set VGA screen size (holdover from QuickBasic)
ScreenWidth = 640
ScreenHeight = 480
y=25
x=152
Timer 1.Enabled = True
End Sub
```

The program uses Public Property Let to create properties that allow data to be sent to the control. The Labels in the above program are used to specify the scale markings. For ActiveX controls, UserControl_Initialize replaces Form Load in a standard VB program. This program was first implemented in QuickBasic (hence the 640x480 resolution), then ported into a Visual Basic ActiveX control. Data is passed to the thermometer through properties Value, Max and Min. Value specifies the temperature, Max the Maximum position on the scale, and Min the minimum position on the scale. For example, if the control is called TemperatureGauge, data can be passed from a new form to the gauge through TemperatureGaugel.Value (note that Visual Basic add a "1" to indicate that the first type of this control is to be loaded on the *form).*

A few simple revisions allow other displays to be created. Shown below is a meter (similar to an ammeter on an instrument panel). In this case the property value containing the data is called Value.

```
Private Sub Timer1_TimerO
Dim bx
Dim xprev
P1= 3.14159265359
'Bottom of Meter Position
XPOS= 120
YPOS= 120
'bx is the current position
xprev =bx
Label1.Visible True
Label4.Visible = True
Label2.Visible = True
Label3.Visible = True
Label1.Caption = Min
Label2.Caption = Max
Label3.Caption = (Min + (Max - Min) /2)
Label4.Caption = Value
'bx is the normalized meter position*PI
```

```
bx= (Value - Min) * (PI) / (Max - Mm)
‘Define limits
If bx < 0 Then bx = 0
If bx  > PI Then bx = PI
‘Draw meter
Line (XPOS - 80, YPOS - 70)-(XPOS + 300, YPOS + 120), QBColor(3), B
Circle (XPOS + 90, YPOS + 80), 100, , 0, PI
‘If meter position has changed, clear screen and redraw to erase old needle position
If xprev <>bx Then
Cls
Line (XPOS - 80, YPOS - 70)-(XPOS + 300, YPOS + 120), QBColor(3), B
Circle (XPOS + 90, YPOS + 80), 100, , 0, PI
EndIf
‘Draw needle
Line (XPOS + 90, YPOS + 80)-(XPOS +90+ 100 * Cos(-PI + bx), YPOS +80 + 100 * Sin(-PI
+ bx)), QBColor(4)

‘Draw tick marks
For XA =0 To PI Step PI/2
Line (XPOS +90+95 * Cos(PI + XA), YPOS +80+95 * Sin(-PI + XA))-(XPOS +90+ 103 *
Cos(-PI + XA), YPOS +80+ 103 * Sin(-PI + XA))
Next
End Sub
```

One of the features of ActiveX controls is that they can be incorporated into web pages, via a web editor such as Microsoft Front Page. The web page www.users.muohio.edu/hergerd/ASEE also contains code to implement a strip chart recorder.

5. Accessing Ports in Visual Basic

Since the first Windows edition of Visual Basic, there has been no provision to read and write to ports. In earlier versions of Visual C++ it was possible to write a DLL to implement a port command in Visual Basic. Starting with Version 4, this provision was not included in Visual C++. A fix for this bug is to force an intrinsic optimization *(*/Oi*)* and use the inp and outp command located in the conio.h header file.

To implement this library, begin by creating a DLL file in Visual C++ with **File/New**. After the DLL is loaded, choose **Settings** from the **Project** menu. From the dialog box, choose **C/C++** and click Customize. In the **Project Options** box, add **/Oi**. The Visual C++ code can then be added. Sample code to implement port 1/0 is shown below:

```
#include <conio.h>
int OutPort(int x,int y)
{
outp(x,y);
return y;
}

#include <conio.h>
```

```
int InPort(int x)
{
int y;
y=inp(x);
return y;
}
```

A complete VC++ DLL project for port I/O is included on the web site. After creating the DLL, it can be loaded into the **Visual Basic General Declarations** section. Sample code to do this is shown below. This code assumes PORT.DLL is in the windows/temp directory.

```
Private Declare Function InPort _
Lib "c:\windows\temp\Port.dll"_
(ByVal x As Integer) As Integer

Private Declare Function OutPort _
Lib "c:\windows\temp\Port.dll"_
(ByVal x As Integer, ByVal y As Integer) As Integer
```

To send a byte to the printer port, the following code can be used:

```
z = OutPort(&H378, 18)
```

This function also returns the data sent to the port.

To read in from the printer port, enter the code:

```
y = InPort(&H378)
```

Note.. This assumes the printer port is at 378 hex. A check on this can be made by looking up the LPT1 port in System Properties located in Control Panel.

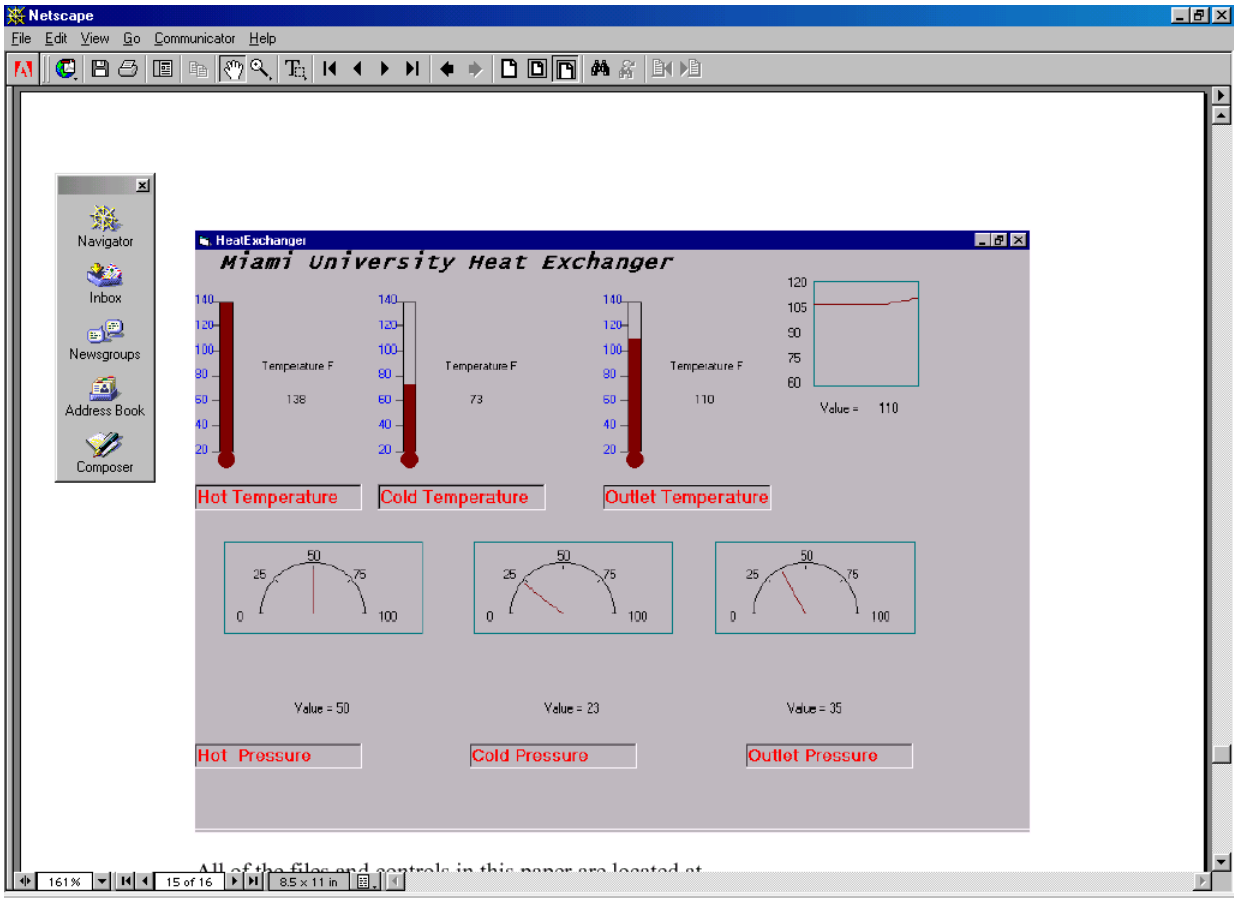
The PORT.DLL library, VC++ code and sample Visual Basic program can be downloaded from the address:

www.users.muohio.edu/hergerd/ASEE

V. Garde (1998) and Hergert provide numerous hardware and software examples to use the printer port for analog and digital reading and writing.

6. Conclusions

This paper shows how to read signals from both the serial and parallel port into Visual Basic. With some working knowledge of VB, it is relatively easy to create a flexible, powerful data acquisition system. Users can easily create their own port DLL using Visual C++ and the techniques described above. In addition, ActiveX controls can be used to create a LabView like display both as a stand alone EXE file and as an HTML page. Shown below is a heat exchanger instrumented with an 8 channel data acquisition board using the VB Active X controls described above.



Bibliography

1. Aitken, P. Internet Programming with Visual Basic 6 in 21 Days, Sams, (1999), pgs 31-109.
2. Hergert, D.Thiebeault, N. PC Architecture:From Assemble Language to C, Prentice Hall, (1987), pgs 64-65.
3. MBD, MK1 1 Microcontroller User’s Manual, MBD, (1994).
4. Microsoft Quick Basic Language Reference Manual, Microsoft, (1987), pgs 296-299,
Microsoft, FIX: Port **1/0** Not in **DLL Version of CRT for VC++ 4.0,** Microsoft Web Page, 2000.
5. URL: http://support.niicrosoft.comIsupportIkb/articles/Q 1 52/0/30.asp?LN=EN-US&SD=gn&FR=0&qry=port%2OVC+±&rnk= 1 &src~DHCS_MSPSSgnSRCH&SPR =VCC; FIX: Port I/O Not in DLL Version of CRT for VC++ 4.0.
6. Perry, G. **Teach Yourself Visual Basic 6 in** 21 Days, Sams, (1999).
7. V. Garde, D. **Programming the Parallel Port, R&D** Books, (1998).

DAVID HERGERT

David Hergert has a B.S.M.E from the University of Cincinnati and a Ph.D. in Industrial Engineering from Pacific Western University. He has taught in the engineering technology department at Miami University for 16 years. His areas of interest are instrumentation and control.