EE634 Project Report
A Tutorial To Develope An Educational Embeded Software With Microsoft Visual Basic

Yongmian Zhang

1 Introduction

In both language size and popularity, the Visual Basic (VB) is probably the biggest and little languages. While its capabilities of rapid GUI development and easy to learn make this full-fledged window applicaiton language flourish around the programming languages.

The PC parallel port is an inexpensive and yet powerful platform for implementing projects dealing with the control of real world peripherals. The $ADC0804$ analog to digital converter (ADC) is simple, cheap, easily available 8-bit ADC. These properties in both hardware and programming language are suitable to make affordable educational project.

In this report, we will describe our system design, explain how to interface PC parallel port and ADC using VB, and show how easy to develop the graphic interface. The same technique described here can be used to develop other educational projects. To provide the background necessary for us to introduce our project, we briefly review the basics of the PC parallel port and ADC0804 in the next section.

2 Hardware Background

The ADC0804 can suite on memory locations of I/O ports to the microprocessor. ADC0804 can be packaged in a DB-25 extended case and plug to computer parallel port without need of any interfacing logic. With a maximum sampling rate of 8kHz, low-quality auddio work could be attempted. The only disadvantage of this techniques is that it is slower, which requires a few I/Oinstructions to read the one byte. It is, neversless an excellent kit for educational projects.

2.1 PC Parallel Port

The printer port provides eight TTL outputs on the data port ($Data7$ through $Data0$), five inputs on status port ($Data3$ through $Data7$) and four additional outputs on the low nibble of the Control port, that is */SELECT_IN, /AUTOFEED, INIT* and */STROBE*. The bi-directionalleads, such as *DIRECTION, IRQ Enable* on the Control Port and *IRQ* on the Status Port, provide a very simple means for advanced interrupt applications. Each printer port consists of three port addresses: data, status and control. These addresses are in sequential order. That is, if the data port is at address $0x0378$, the corresponding status port is at $0x0379$ and the control port is at $0x037A$. Table 1 listed addresses of three standard printer port (SSP).

Table 1: Addresses of three standard parallel port.

Base Addr	Data Addr	Status Addr	Control Addr
0x378	0x378	0x379	0x37A
0x3BC	0x3BC	0x3BD	0x3BE
0x278	0x278	0x279	0x37A

2.2 ADC0804 Interface Circuit

Figure 2 shows the schematic of basic circuit of $ADC0804$. The operation of the $74LS157$ simply acts as four switches. When the $\bar{A}/B$ input is low, the A input which is low niblle of input are selected. That is, $1A$ passes through to $1Y$, $2A$ passes through to $2Y$ etc. When $\bar{A}/B$ is high, the B inputs are selected and 4 high bits from $1B$ to $4B$ sent out to coresponding Y pins from $1Y$ to $4Y$. The Y outputs are connected to the Parallel Port's status port as shown in Figure 2, the 4 Pins, i.e., *Busy, Ack, Paper Out, Select*, represent the most significant nibble of the status register.

If we look at the ADC's operation, on a high to

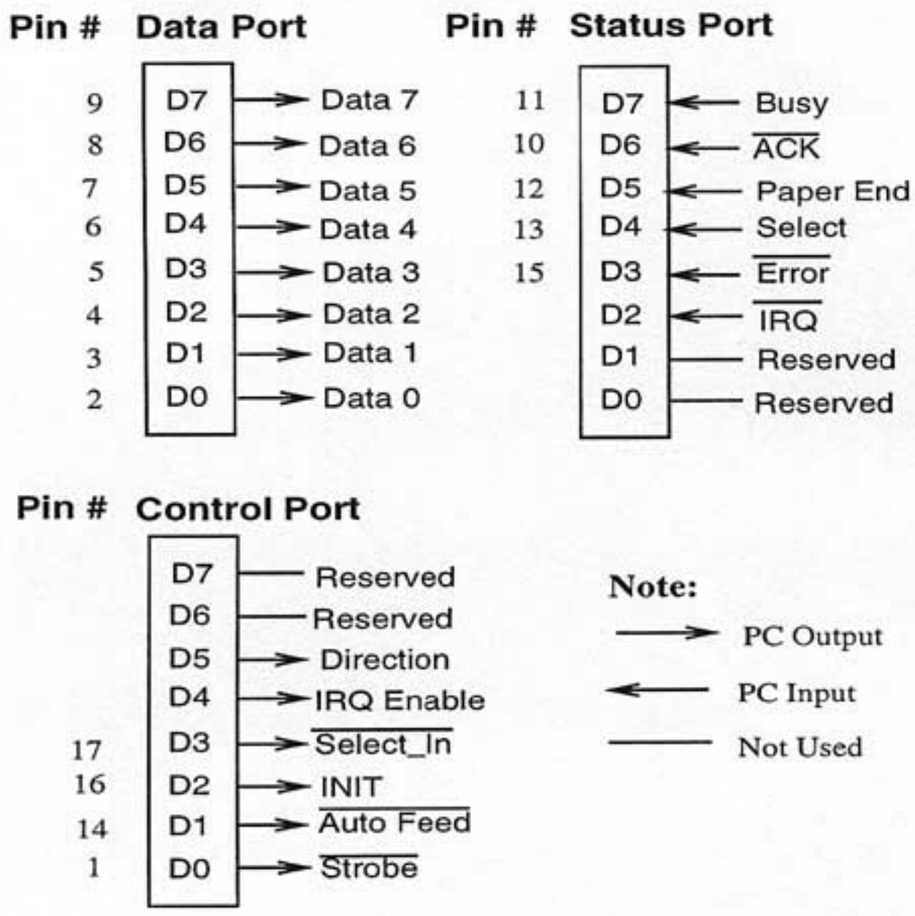

Figure 1: Pin assignments on the 25 pin connector and the bit assignments on the three ports.

low tansition of the the WR input the internal Sucessive Approximation and Shift Registers are reset. The conversion process will start when a low to high transition is made on the WR input. Once $ADC0804$ make a digital conversion of the analog voltage on it's pins, the multiplexer $74LS157$ is used to read a nibble data at a time, and then it switches to the other nibble and reads it. The software can be used to read the nibble from Status port. After receiving two nibbles, software constructs the two nibbles into a byte raw value and display it on screen. The following table 2 shows the bits and their position in the status register. The Status port is read and the most significant bit, corresponding to the *Busy* bit (pin 1 on D pin) is inverted using the exclusive-or function.

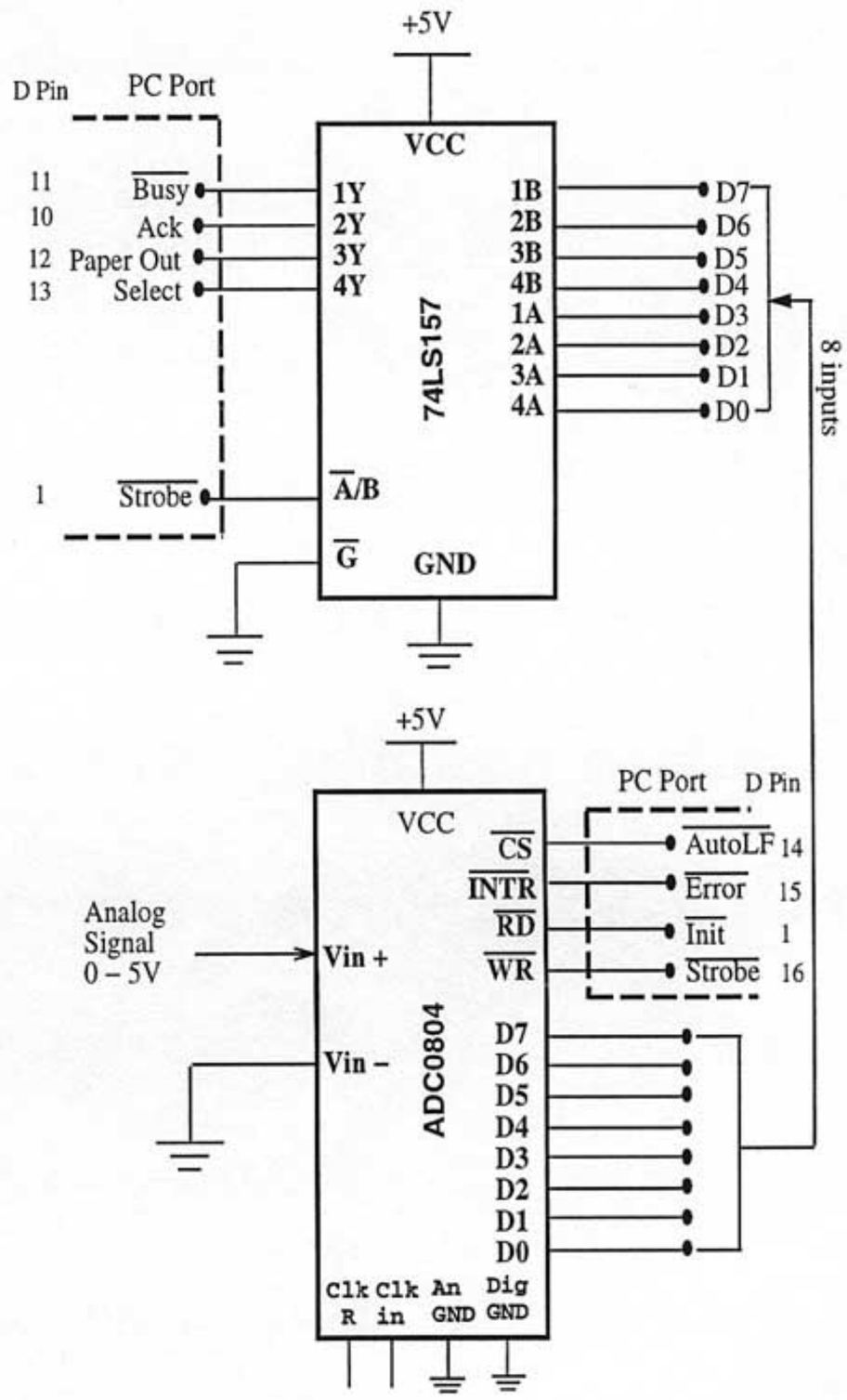

Figure 2: The connection of A/D converter to the PC parallel port.

Table 2: The nibbles from Status port.

Status Bit	Low Nibble	High Nibble
$S4$	$D0$	$D4$
$S5$	$D1$	$D5$
$S6$	$D2$	$D6$
$\bar{S}7$	$D3$	$D7$

3 Interface to Hardware

3.1 Accessing Parallel Port in VB

To write directly from Status port and Control port, we need the functions in programming language which has ability to access port registers and control over the parallel port signals. Most of programming languages, such as Assembly, C/C++, Pascal/Delphi, Qbasic, to directly manipulate bits of port registers, and instruct CPU to write and read data from specified locations. Unfortunately, this also has downside: it has potential risk of crash the system if a critical memory or port address is overwritten by mistake. Miscrosoft eliminated the existance of memory access function in VB. However, almost all programming language support libraries of standard code modules for commonly used functions. VB also provides an entry to access procedures from a dynamic link library (DLL) written in other programming language to fit our own requirement.

Since C/C++ provides the built in functions *inp()* and *outp()* which can manipulate bits on port registers, we can utilize Microsoft Visual Studio to create our own DLL file to cater VB applications. To build a DLL, open the VC++ working studio, create a new project, give it the name as you want, and choose its type as "Dynamic Link Library". Then insert the following source file and header file to the project and build it to a 32 bit DLL file, which contains a collection of our own version of *inp()* and *outp()* that can be used by any number of different programs. Particularly that when your program calls a function in a DLL, window will automatically load the DLL into memory.

```
/*This is header file ioport.h*/
#include <stdio.h>
#include <conio.h>
#include <windows.h>
int __declspec(dllexport) __stdcall
inp32(unsigned short portAddr);

void __declspec(dllexport) __stdcall
outp32(unsigned short portAddr, int value)
```

```
/*This is source file ioport.c*/
#include "ioport.h"

/*read a byte from hardware port*/
int __declspec(dllexport) __stdcall
inp32(unsigned short portAddr)
{
    int value;
    value = _inp(portAddr);
    return value;
}

/*output a byte to hardware port*/
void __declspec(dllexport) __stdcall
outp32(unsigned short portAddr, int value)
{
    _outp(portAddr, value);
}
```

Since **__declspec(dllexport)** is used on an exported function in our DLL file, the decorated name (the symbol function name with an underscore(_)) is exported. In order to be calling DLL function from VB applicaitons, each modules in the DLL file have to be defined as **__stdcall**. The *__stdcall* name decoration prefixes the symbol name with an underscore(_) and appends the symbol with an at sign(@) character followed by the number of bytes in the argument list (the required stack space). You can check the decorated name in the map file after DLL file is built if *map file* of linker setting on *project* manu in VC++ is on. For DLLs to be called by programs written in the 32-bit version of VB, the alias technique in VB program is needed to refer to the memory location that is already referred to by a module in DLL, as shown in follows. Now we can use *inp()* and *outp()* functions to control over the port registers in our VB applicaitons.

```
Public Declare Function inp Lib
"ioport.dll" Alias "_inp32@4" (ByVal _
PortAddress As Integer) As Integer

Public Declare Sub outp Lib "ioport.dll" _
Alias "_outp32@8" (ByVal PortAddress As _
Integer, ByVal Value As Integer)
```

3.2 Interface to Parallel Port

To make above hardware work, first we must initialize the 74157 multiplexer to switch either inputs low nibble or high nibble of 8 inputs from ADC. We use

functions *outp()* and *inp()* created in previous section to manipulate PC parallel port control and status bits.

We will read the least significant first, thus we have to place $\bar{A}/B$ low. Since the strobe is hardware inverted, thus we must set Bit 0 of the control port to get a low on Pin 1 in parallel port. Therefore we write $0x02$ to control register and simutaniously bring WR on ADC0804 from low to high to start the conversion process, and latch 8 bits data to 74157. It may be necessary to add delays in the process. That is

```
'Bring WR from low to high,
'and low nibble (A pin on
'74157) is selected
outp Ctrl, &H02
Delay(some_time)
```

Once the low nibble is selected, we can read the leat significant nibble from status port. Since the *Busy* bit is hardware inverted, we use *XOR* with *0x80* to toggle it back. We are only interested in the most significant nibble of the results, thus we *and* the results with *0xF0* to mask off the least significant nibble, and shift the nibble we have just read to the position of least significant nibble of variable. The code segment is shown as follows.

```
'Get low nibble and right shift it
'4 bits to the low nibble position
lowNibble = (inp(Status) Xor &H80) \ 16
```

To get the most significant nibble, we have to switch the 74157 multiplexer to select input B. Thus we write $0x03$ to control register, and at the same time, this also bring WR from high to low and finish one byte conversion. Now we can read the most significant nibble and put the two nibble together to form one byte of digital value. The digital value can be returned to graphical interface to display it.

```
'WR fron high to low and B bits
'on 74157 selected.
outp Ctrl, &H03
highNibble = (inp(Status) Xor &H80) And &HF0
'restore control register
outp Ctrl, &H06
'Form one byte of digital value and
'return it to calling function
ByteValue = (higNibble Or lowNibble)
```

To summarize, the routine for one byte conversion will look like:

```
Public Function ReadAD() As Integer
   Dim dataLow As Integer
   Dim dataHigh As Integer
   Dim rawData As Integer

   Do While (inp(stat) And &H8)
   Loop   'wait until INTR low
   outp ctrl, &H2 'CS and RD low, WR high
   Delay (1)
   ' get low nibble, need invert s7
   dataLow = (inp(stat) Xor &H80) \ 16
   outp ctrl, &H3   'CS, WR, RD low
   Delay (1)
   'Get high bits and mask off unwanted
   'bits, s7b need invert
   dataHigh = (inp(stat) Xor &H80) And &HF0
   outp ctrl, &H6 'CS Low, WR and RD high
   Delay (1)
   'form one byte value
   ReadAD = dataHigh Or dataLow
End Function
```

4 Acquiring and Visulization

The Figure 3 illustrates the functional view of analog to digital converting system we made with VB. The software runs three synchronous tasks:

- Acquiring data from the hardware
- Shipping data to displayer
- Access event from user input

These tasks exchange data and communicate with each other by a software main control loop.

The heart of the application is the sampling main calling loop, which is the runtime interface to AD Converter and continously ship data to graphic window. The simplified code is listed as follows. The outside infinite main loop is implemented by VB timer control.

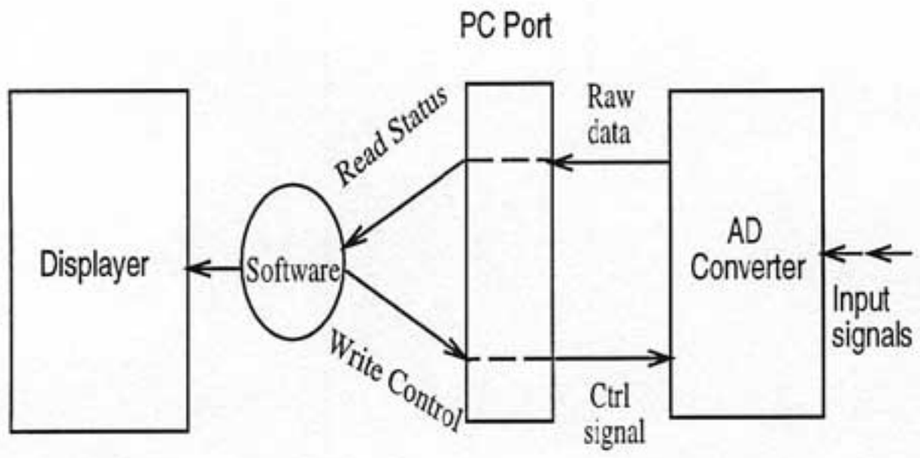

Figure 3: The functional diagram of the system

In order to ease the user's view, the control timer can be set from 0 to certain seconds based on the machine speed, to delay erasing graphic data after one window sampling has done.

```
Sub Sampling_Timer()
    Dim voltage As Double
    Dim i As Integer

    For i = 0 To samples - 1
       voltage = ReadAD() * maxVoltage / 256
       DrawResults i, voltage
    Next i
    If StopButtonPressed Then
       exit sub
    End If

End Sub
```

To add graphical elements to our systems, a little development effort was needed with VB graphic facilities, and can still produce nice results. Following code segment is our data visualization routine. Samples are scaled to fit into one window. If running mode set to infinite, each window will display required number of samples. Figure 4 gives a flow chart for this module.

```
Private Function DrawResults(i as Integer,_
            value As Double)
  If i = 0 Then
    x0 = origX
    y0 = ScreenHeight + origY - _
          (ScreenHeight * value) / maxVoltage
  Else
    x1 = origX + i * (maxDispWidth / samples)
    y1 = ScreenHeight + origY - _
       (ScreenHeight * value) / maxVoltage
    Line (x0, y0)-(x1, y1), drawColor
    x0 = x1
    y0 = y1
  End If
End Function
```

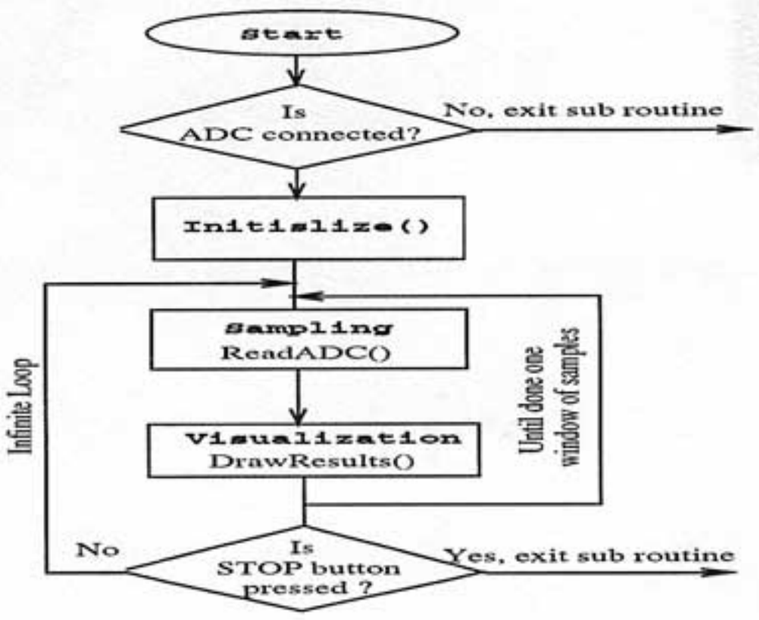

Figure 4: The program flow chart of sampling module

5 User Graphic Interface

Little code was needed for user's interface, since VB provides prebuilt graphic user interface components and easily manipulate layout and component properties. VB also lets you interactively run the application while building it. Figure ?? shows the completed user interface for this project, which consists of 9 event buttons and one data display window. Samples shows on display window are from audio inputs.

We built many necessary features into the user interface that could easily be accomplished with VB. User is allowed to choose sampling mode by click sample mode button, as shown in Figure ??. The dialog box provides user to set sampling time. The dialog box also gives us the option to choose running mode either just sampling the number of samples that user input or infinite sampling mode. User can stop it any time by simply pressing the stop button if infinite sampling mode is chosen. Users are allowed to change default address *378H* to their LPT base address by simply choosing one from drop-down list. The Save button provide user to log samples to PC for further analysis. The attributes of display window can be changed by pressing buttons of forground color, background color, reduce screen and enlarge screen.

The input voltage ranges are detected by software automatically and whenever the input range has

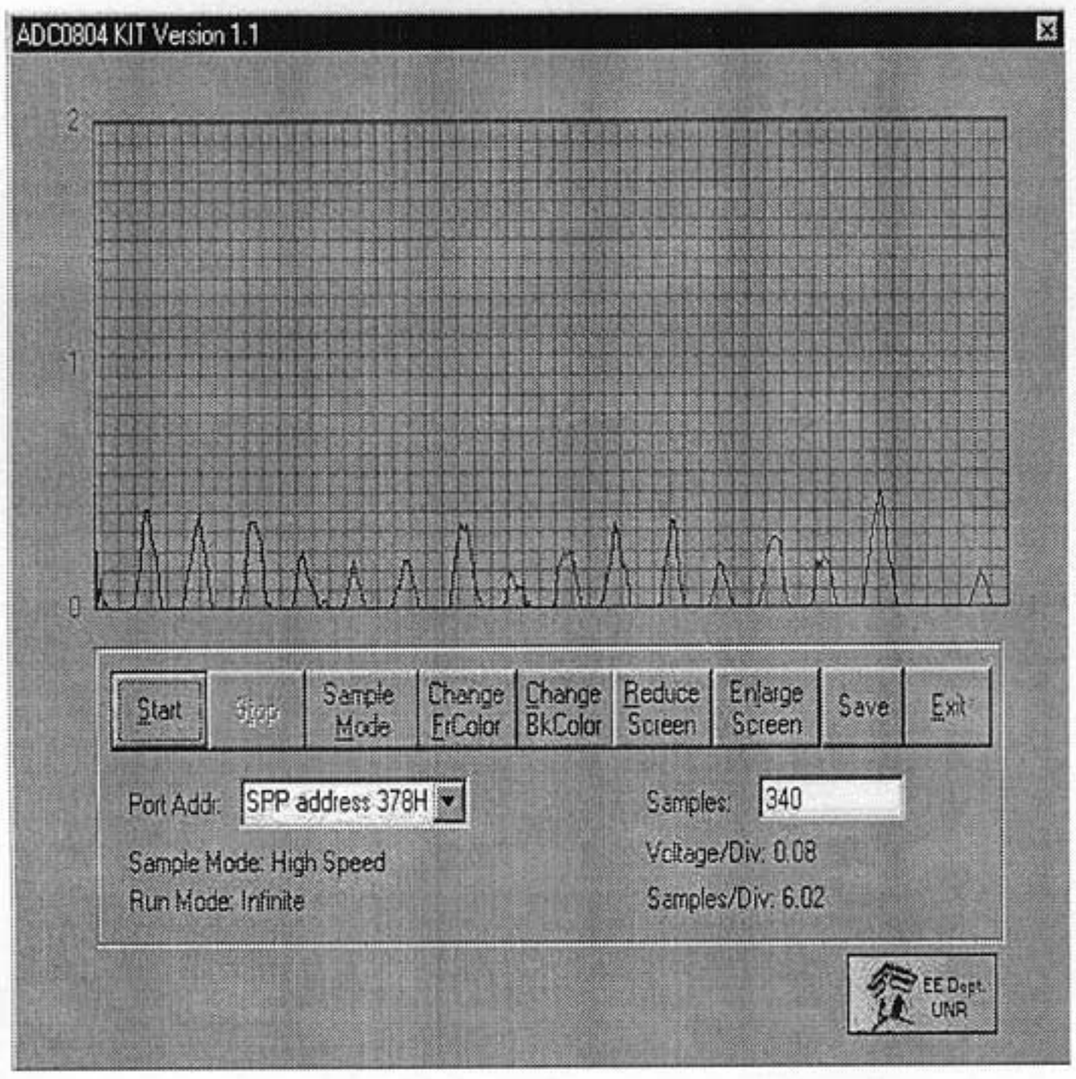

Figure 5: The graphic user interface

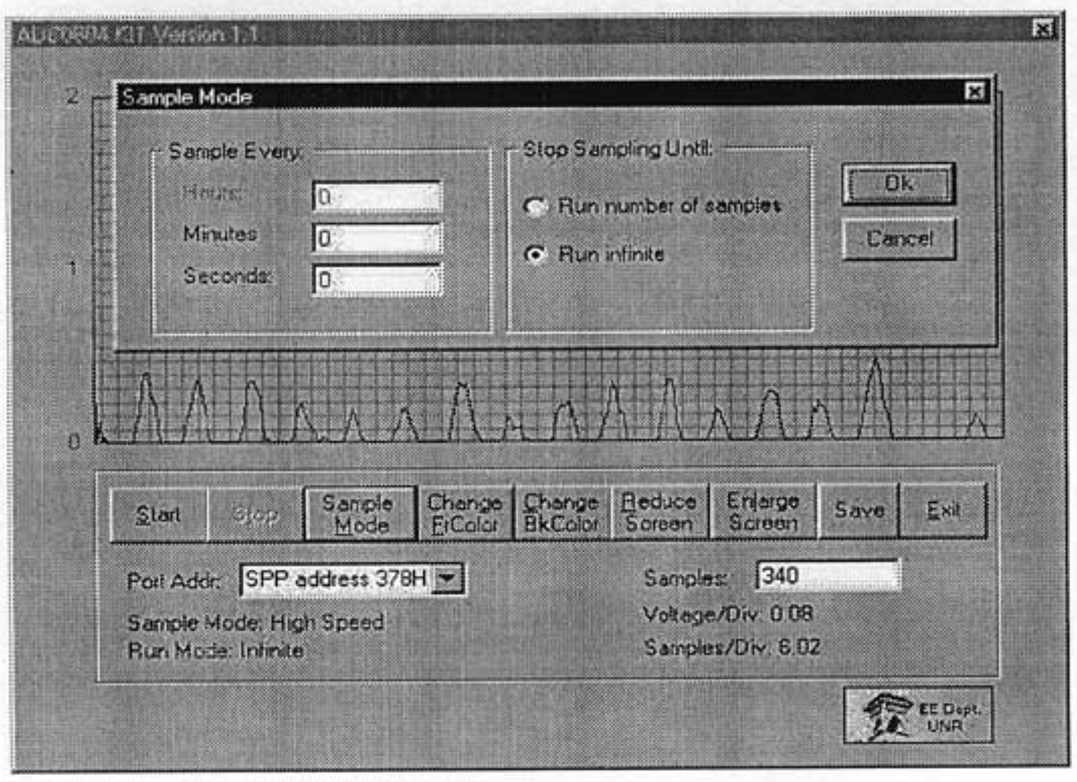

Figure 6: The dialog box to choose sampling mode

changed, at the same time, the voltage scaler on y-axis is also update. The voltage per division on Y-axis is given on message window. Samples are scaled to fit on one display window. The parameter value of *Samples/Division* will be recalculated when changing the number of samples per window or also the window size. We use float value to represent samples per division in order to accurately reflect the change of window size.

6 Implementation

Listed are major routines for interfacing to PC parallel port, to handle creating windows for display and writing text and attributes to the windows.

1. *cmdStart_click()*: This routine is to reponse for user's start button. The function performs to check if existance of port address and then check sampler is ready or not. Finally start sampling based on the sampling mode and running mode.

2. *StartSample()*: A infinite loop is implement for continuously sampling data from ADC. The sampling can be stopped by user's pressing ESC key.

3. *ClearDrawing()*: Redraw sampled results with setting color. If background color is passed to this function, it acts to erase drawing on the screen.

4. *DrawResult()*: Graphically display sampled digital data on the screen. All sampled data are scaled to fit one screen at a time.

5. *Delay()*: implemented as a dummy loop to delay a certain number of instruction cycles.

6. *InitSampler()*: Power up the sampler and check if sampler is connected to PC parallel port.

7. *ReadAD()*: Read raw data from AD converter. First read low nibble and then get high nibble and final form a one byte value.

8. *GetVoltageRange()*:Get input voltage ranges of ADC sampler. For ADC0804, the maximum input voltage is 2V or 20V.

9. *CheckPortExist()*: Check if user specified port address is exist. Write two bytes and read then back. If the reads match the writes, the port exists.

10. *GetPortBaseAddr()*: Access user's input for change the port base address.

11. *DefaultPortSetting()*: Set port base address as $0x378$

7 Summary

This artical demonstrates how to use easy available 8-bit ADC and utilize powerful PC parallel port to develope home brew projects for educational purpose. Visual Basic is a popular window programming language for everage programmers. However, there is a hitch to directly manipulate the PC port registers. We illustrated how to use Visual C++ studio with few lines of C code to create a DLL file to cater for our VB application. How to interface to parallel port and AD converter was also described. The same technique described here can be utilized to develop other educational projects related with accessing PC parallel port.