DEVELOPMENT OF EMBEDDED SOFTWARE FOR FLOWSTATION METERING USING VB.NET WITH MCCDAQ BOARD: A CASE STUDY OF PETROLEUM TRAINING INSTITUTE (PTI) DEMONSTRATION FLOWSTATION

F. K. Opara
Electronic Computer Engineering Department, Federal University of Technology, Owerri, Imo State
NIGERIA.

G. N. Okorafor.
Electronic Computer Engineering Department, Federal University of Technology, Owerri, Imo State
NIGERIA.
nwaji2000@gmail.com

C. S. Ikpeazu
Electronic Computer Engineering Department, Federal University of Technology, Owerri, Imo State
NIGERIA.

ABSTRACT

This paper presents a case study of converting an old pneumatic metering system of a Flowstation to a new system for digital display on a large screen. An analysis of the existing instrumentation system was carried out, and appropriate hardware and software platforms were selected. The system software was developed with MS Visual Studio 2010 and MS Access 2007. The Software system was tested with McDaq demo Board and the result was successful and satisfactory.

Keywords: Embedded system, Flowstation, VB.NET, MS Access, Database, Framework.

INTRODUCTION

Measurement Computing Corporation (Mcc) 16bit Data Acquisition (DAQ) can be used for Original Equipment Manufacturer (OEM), the original intended usage, and embedded computer [1]. Embedded systems are computers which are part of a special purpose device of larger unit [2, 3], or self-contained unit providing services to the unit [4]. In some cases, Embedded systems are electro-mechanical products which relate hardware and software contained in a computer which is a part of the larger system and providing non-computing features to Users [5] and in most cases relates with the environment [4, 6]. In reality, embedded systems comprises of Reactive systems, Real-Time constraints systems, and Data processing Capabilities [7]. Early embedded systems were mostly found in aircrafts, space mission crafts, and missile guidance systems [2]. However, modern Embedded systems can be found in Phones, Video systems, digital watches, microwave oven, traffic control systems, industrial automation systems and many others such as Routers, firewalls, copiers, printers, disk drives, calculators, Data Acquisition systems, and programmable logic controllers (PLCs) [2, 3]. All Embedded systems have both hardware and software constraints. Limited processing power, memory, power usage in the case of sensor networks, storage facilities as well as timing and responses [3, 8]. Embedded systems have found wide applications in nearly all the facets of Oil and Gas industry [9] such as the use of intelligent Field Devices, which are normal control loop devices but incorporated with local embedded computers and Supervisory control and Data Acquisition (SCADA) system are used in production and phase separation facilities.[10, 11, 12, 13].

Advantages of Using Embedded System:

The embedded system in this paper has many advantages over the existing pneumatic systems in the station in many parameters, such as accuracy, response time, resolution, and ability to be integrated into a lager network such as field bus. The advantages are already summarized in Table1.

Table1: Comparison between the existing and the proposed Systems

Parameter	Pneumatic System	Embedded System
Distance	Not more than 600m	Electromagnetic signal can travel to any distance with repeaters in place
Accuracy	Subjected parallax error	Digital display more accurately
Ability to integrate into Distributed Control Systems and Fildbuses	Not possible	Can be integrated into Fieldbus and Distributed Control Systems
Maintainability	Bending and Installing pipes	Wires are easily replaced and most of the maintenance are herein software based
Scalability	Scalability limited	More channels can be added with ease

The Flowstation

A flowstaion is a treatment centre in an Oil field where fluid is collected with minimum pressure loss. Here, the gas is separated from the oil and oil from water. Petroleum as produced from a reservoir is a complex mixture of hundreds of different compounds of hydrogen and carbon, all with different densities, vapour pressures and other physical characteristics, water, emulsion, and sediments. In most cases, this mixture is at elevated temperature and pressure [14, 15].

Table 2: The taxonomy of the process Variables

Pressure	Temperature
Line1 pressure	Separator1 temperature
Line2 Pressure	Separator2 temperature
Line3 pressure	Separator3 temperature
Separator1 Pressure	
Separator2 pressure	
Separator3 pressure	

Sediments from the reservoir include sand, clay, and silt; while contaminants are mostly made up of dissolved salt, Carbon dioxide (CO_2), and hydrogen sulphide (H_2S). Separator is a major component of a flowstation or surface production facilities and is manufactured in various shapes, namely; vertical, horizontal, and spherical.

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 2, No. 2, March 2012

The Petroleum Training Institute (PTI) demonstration flowstation was designed to simulate the real field situation. It comprises of the flow lines from wells, the entry manifold where the flow lines are diverted into three lines which feed the three horizontal separators, de-emulsifier plant, discharge lines and other units. The process operation of the station is not discussed in this paper.

However, as the instrumentation system is completely pneumatic, observation shows that there are only nine process variables of interest connected to meters mounted on a board at the monitoring point. The variables are line pressures, separator pressures and separator temperature. The taxonomy of the process variables are shown in Table2.

THE METHODOLOGY

This research involves both hardware selection and software development. The methodology used is as stated below:

The hardware aspect:

*A visit to the flowstation for a comprehensive visual inspection; *Physically tracing the process piping; *Physically tracing the pneumatic tubing connecting all the instruments; *Identifying process variables of interest; and *Selection of hardware platform for the new system.

The description of the hardware is presented later in this paper.

The software aspect:

*Development of the system interface software; *Development of the sensor database; and *Testing of the complete system using Demo board a software that accompanies Instacal, the MccDaq driver

This work intends to replace all pneumatic transmitters in the station with sensors. In the same vein, all pneumatic transmission lines (tubing) will be replaced with twisted pair (TP) cables. Pneumatic gauges will be replaced with an electronic display board. The data acquisition board is used as an embedded computer which inputs sensors' outputs into the computer in which the software hereby developed is installed.

DEVELOPMENT OF THE SYSTEM

Embedded System Board For the embedded hardware platform, the Measurement Computing USB 1616FS was selected. It can sample 16 analogue input channels simultaneously at the rate of 50Kilo samples per second per channel through one ADC for each channel. It has 8 digital I/O channels. It operates with both 5V and 10V sensors. Also the choice of this USB 1616FS is adequate because the flowstaion needs only 9 analogue input channels and 1 digital output channel. The board is made up of one USB Microcontroller, 16 ADCs with buffers, and 32x16 6bit SRAM and few other units [17]. Measurement Computing Corporation (MCC) has open Universal library with reusable codes and declaration [18, 19]. The universal library that accompanies instacal [19], the device driver, contains many examples, whose codes can be modified to suit a particular solution.

Software Development:

The Software development process was split into two Phases, owing to the use of two development environments. The Interface software was developed with Visual Studio 2010 (VB.NET) and the Sensors database (Dbase) was developed with Microsoft Access 2007.

Development of Sensors Database:

This procedure was carried out following data acquisition system database design. Databases are classified as small, medium, and large. Generally, there are three popular platforms for Dbase development, namely, Oracle, Sequential Query Language (SQL), and Access [20]. Their areas of application and capacity building are shown in Table3. The Flowstaion metering station requires on nine sensors as can be seen in table4. There are nine fields and nine records. This is a small Dbase system. In general, all flowstation fall in small dbase systems. Hence, MS Access 2007 was used to create the sensors relational database and datasets as shown in table4.

Table3: Database Platform Taxonomy

Platform	Area of Application
Oracle	Large
SQL	Medium
MS Access	Small

The ID field contains the serial numbering of the sensors. This was done through the auto numbering of MS Access. The second field describes the type of process; while the third field describes the type of process variable. The fourth field is the channel number of each sensor; while fifth field is for instantaneous sensors, outputs. The sixth and the seventh fields record the sensors' lowest and highest outputs respectively. The eighth and ninth fields are for the minimum and the maximum values of the process variables respectively.

Table4: Database form in MS Access 2007 Version

StnSensors

S/N	Sensor Name	Description	Channel Number	Signal	Low Volt	High Volt	Low Val	High Val
1	Line1press	pressure sensor	1					
2	Line2press	pressure sensor	2					
3	Line3press	pressure sensor	3					
4	Sep1press	pressure sensor	4					
5	Sep2press	pressure sensor	5					
6	Sep3press	pressure sensor	6					
7	Sep1temp	Temp. sensor	7					
8	Sep2temp	Temp. sensor	8					
9	Sep3temp	Temp. sensor	9					

Interface Software Development:

The interface software developed with VB.NET 2010 consists of three class forms and a module. VB.NET is an object oriented programming language (OOP). It is rich in reusable codes.

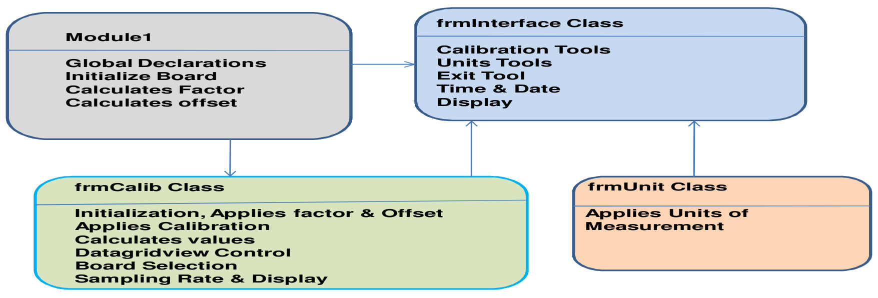


Figure1: The Software Structure

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 2, No. 2, March 2012

The parent class frmInterface, handles everything regarding to users interface. Module1 is used for global variables declarations, global functions such as factor and offset computation and global board initialization. frmCalib is call by frminterface when sensors calibration is required, recalculate and reload factors. frmUnit is called by frmInterface for the imputation of units of measurement. These relationships is shown in the structure of Figure1

OPERATION

When the software icon is double clicked, frminterface is displayed with black backcolour for beauty and clarity in reading of the displayed characters. At this juncture, the pixel control is activated as well as loading of time and date in timer click event. On load event and default units saved in settings are loaded and displayed. The next operation is to select board, by clicking on sensor on the menu bar and selecting calibration to display frmcalib. The default is the Demo board, software that comes with the board driver. The numeric up-down in frmcalib is used to select the particular board, board1 in this work. Calibration is carried out on the data grid view on frmcalib. By clicking apply, the new calibration value is used under timer click (Sampling) and display codes in frmcalib to obtain process variable values for display on frminterface. Same calibration procedure will take place in the case of new variable ranges At this juncture; the database is updated with calibration parameters. The default units are saved an area in registry called settings in this sequence; application (Metering), location (Unit setting), value (The particular Unit), and default value (measurement unit for the variable). Whenever new units are required frmUnit is called by clicking sensor on the menu bar, selecting unit to display the form. The required units are typed into their respective textboxes, click apply to save in settings. Since the values are in the registry, the program has to restart anytime the units are altered

When the above procedures are completed, also, when these procedures are not needed; the sensors are sampled, a 9x9 array is created using timer tick event of frmcalib. Computation of factors, offset, and process variable values are carried out using Module1 and frmcalib codes. The computed values are displayed on the interface form. The database is updated accordingly. These procedures are summarized in the flowchart (Figure2).

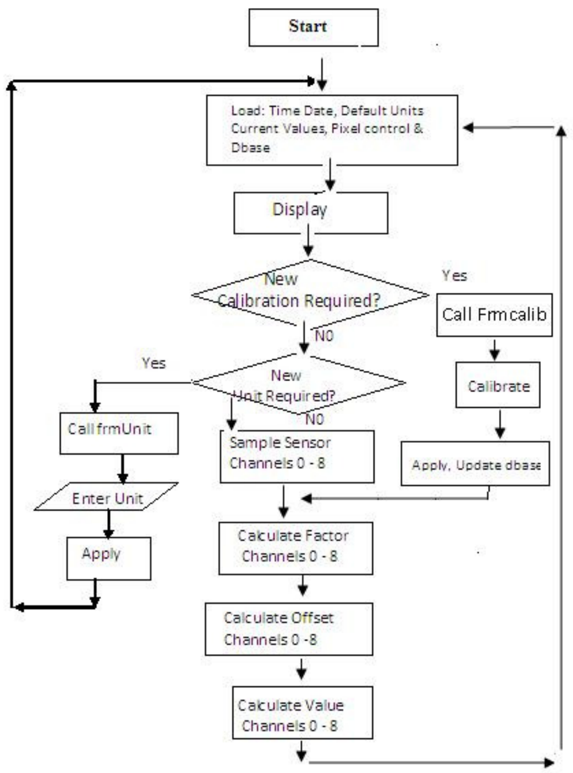

Start
Load: Time Date, Default Units Current Values, Pixel control & Dbase
Display
New Calibration Required?
Yes
NO
Call Frmcalib
Calibrate
Apply, Update dbase
New Unit Required?
Yes
NO
Call frmUnit
Enter Unit
Apply
Sample Sensor Channels 0 - 8
Calculate Factor Channels 0 - 8
Calculate Offset Channels 0 -8
Calculate Value Channels 0 - 8

Figure2: The Flowchart

Design Methodology Forms:

The first stage was the creation of Visual Basic Windows Application project, named metering. The next is the defining of the forms and there are four design forms, frmInterface Design, frmCalib Design, and frmUnit Design were created for this project. Also designing with VB.NET requires dropping an object from the toolbox on the form and manipulation of its properties using the property table [21, 22, 23]. The following properties were mostly used: Name, Size, Text, Font, Forecolor, Backcolor, Enable, and Anchor. The codes are written in frmInterface, frmCalib, Module1, and frmUnit

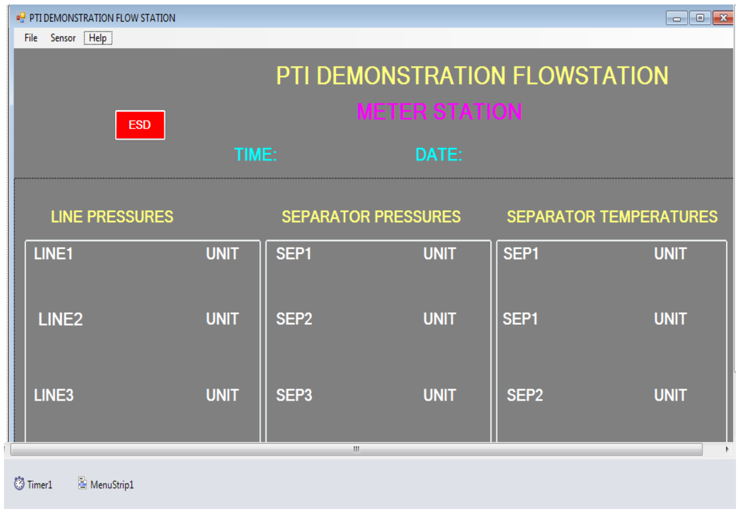


Figure3: Interface Design

frmInterface Codes

Public Class frmInterface

```
Private Sub frmInterface_Activated(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Activated
        Dim pxw, pxh As Double
        pxw = My.Computer.Screen.Bounds.Width
        pxw = (pxw - Panel1.Width) / 2
        If pxw < 0 Then pxw = 0
        pxh = My.Computer.Screen.Bounds.Height
        pxh = (pxh - Panel1.Height) / 2
        If pxh < 0 Then pxh = 0
        Panel1.Location = New Point(pxw, pxh)
    End Sub
Private Sub frmInterface_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        frmCalib.Show()
        frmCalib.Hide()
        frmUnit.Show()
        frmUnit.Hide()
        CalculateFactor()
        lblUnit1.Text = GetSetting("Metering", "Unitsettings", "unit1", "psi")
        lblUnit2.Text = GetSetting("Metering", "Unitsettings", "unit2", "psi")
        lblUnit3.Text = GetSetting("Metering", "Unitsettings", "unit3", "psi")
        lblUnit4.Text = GetSetting("Metering", "Unitsettings", "unit4", "bar")
        lblUnit5.Text = GetSetting("Metering", "Unitsettings", "unit5", "bar")
        lblUnit6.Text = GetSetting("Metering", "Unitsettings", "unit6", "bar")
        lblUnit7.Text = GetSetting("Metering", "Unitsettings", "unit7", "degC")
        lblUnit8.Text = GetSetting("Metering", "Unitsettings", "unit8", "degC")
        lblUnit9.Text = GetSetting("Metering", "Unitsettings", "unit9", "degC")
    End Sub

Private Sub Timer1_Tick(By Val sender As System. Object, By Val e As System. Event Args) Handles Timer1.Tick
        lblTime2.Text = Format(Now, "long time")
        lblDate2.Text = Format(Now, "long date")
        If Button3.Enabled = True Then
            Button3.Enabled = False
        Else
            Button3.Enabled = True
        End If
    End Sub
Private Sub Calibration Tool Strip MenuItem_Click (By Val sender As System. Object, ByVal e As System. Event Args) Handles Calibration Tool Strip Menu Item. Click
        frmCalib.Show()
End Sub

Private Sub UnitsToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles UnitsToolStripMenuItem.Click
        frmUnit.Show()
    End Sub
    Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click
        End
    End Sub
End Class
```

frmCalib:

This is an additional class form with datagridview dragged into it from VB.NET ToolBox. mIt connects the database in MS Access. It handles sensor calibration, sampling rate, and loading updates of the data base. frmCalib is shown in figure4.

Offset:

The standard output of most sensors is 4mA – 20mA. When terminated with 250ΩResistor in An instrument produces 1V – 5V. However, there is another standard whichOutputs 0 – 20mA, which translates to 1 – 10V respectively when terminated with 500ΩResistor [16]. In the first popular standard, the zero value of the processVariable Corresponds to 4mA (1V). This is usually termed offset in data acquisition

Calibration Factor:

In process Control Instrumentation, there is a signal range for the minimum value and the maximum value which correspond to the minimum and maximum sendor output respectively as shown in figure. Hence these definitions are applied.

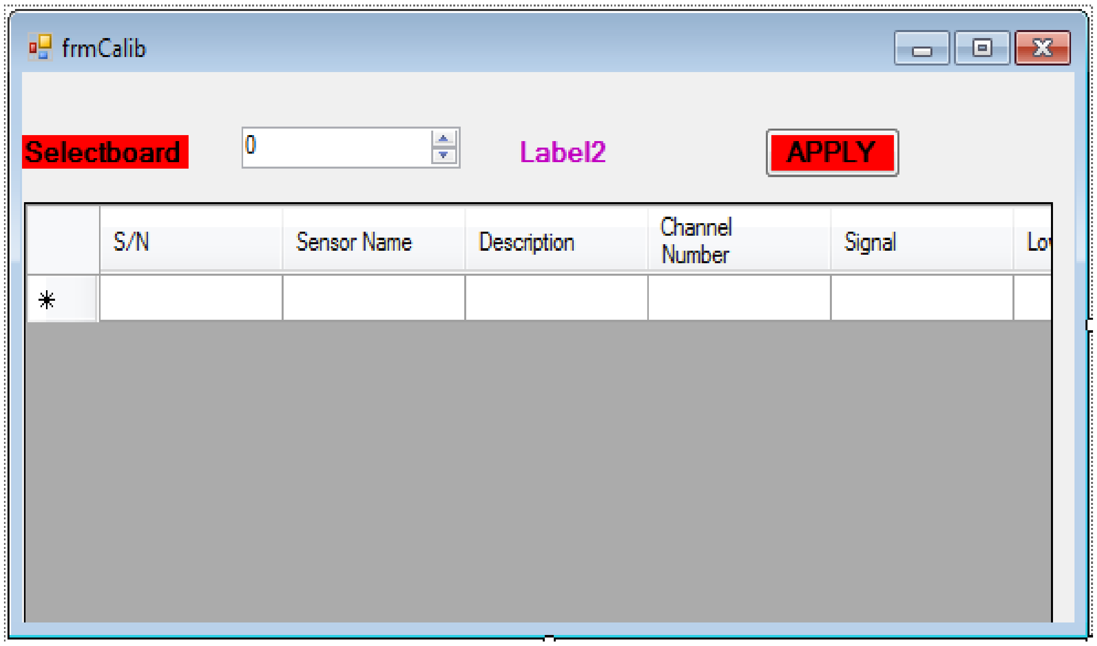
HighVolt = the Maximun value of Sensor's output

LowVolt = the minimum value of Sensor's output

HighValue = the maximum expected value of the process variable (Sometimes referred to as engineering value)

LowValue = the minimum expected engineering value

Factor is the slope of the graph (HighValue – LowValue)/(HighVolt – LowVolt) Which in Datagridview = Currentcell(9,j) –currentCell(8,j)/Currentcell(7,j)- currentcell(6,j) Where(i, j) define columns and rows respectively.



.Figure4: Datagidview Calib Design

frmCalib Codes:

The load event Sub, the DataGridView, and the Timer tick sub are reusable codes from MccDaq Universal Library [19] modified for this application.

Public Class frmCalib

Imports Microsoft.VisualBasic

```
Imports System
Imports System.IO

Public Class frmCalib

Private Sub frmCalib_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'TODO: This line of code loads data into the 'FlwStnSensorsDataSet.StnSensors' table. You can move, or remove it, as needed.
        Me.StnSensorsTableAdapter.Fill(Me.FlwStnSensorsDataSet.StnSensors)
        initializeboard()
        CalculateFactor()
    End Sub

Private Sub DataGridView1_CellContentClick(ByVal sender As System.Object, ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) Handles DataGridView1.CellContentClick
        Dim colIndex, rowIndex As Integer
        colIndex = DataGridView1.CurrentCell.ColumnIndex
        rowIndex = DataGridView1.CurrentCell.RowIndex

        If colIndex = 7 Then
            DataGridView1.Item(5, rowIndex).Value = DataGridView1.Item(4, rowIndex).Value
        End If

        If colIndex = 8 Then
            DataGridView1.Item(6, rowIndex).Value = DataGridView1.Item(4, rowIndex).Value
        End If
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick

        Dim signalRange As String = "5volts"
        Dim row As Integer
        Dim p As Integer
        Dim HighChan As Integer
        Count = NumPoints    ' total number of data points to collect
        Rate = 390    ' per channel sampling rate ((samples per second) per channel)
        Options = MccDaq.ScanOptions.Default
        If signalRange = "5volts" Then
            Range = MccDaq.Range.Bip5Volts     ' set the range to 0-5volts
        Else
            Range = MccDaq.Range.Bip10Volts     ' set the range to 0-10volts
        End If

        AnaDaqBoard.BoardConfig.GetNumAdChans(HighChan)

        For p = 0 To HighChan - 1
            ULStat = AnaDaqBoard.AIn32(p, Range, MemHandleArray(p), 0) 'ULSTAT stores error info
from board initialisation
            AnaDaqBoard.ToEngUnits32(Range, MemHandleArray(p), VoltCount(p))
            volts(p) = CSng(VoltCount(p))
        Next

        row = DataGridView1.CurrentCell.RowIndex
        For i = 0 To 8
            DataGridView1.Item(4, i).Value = volts(i)
        Next

'Display values on line1press, line2press, line3press, sep1press, sep2press, sep3press, sep1temp,
sep2temp, and sep3temp
        frmInterface.lblRead1.Text = volts(0) * factor(1)
        frmInterface.lblRead1.Text = Val(frmInterface.lblRead1.Text) - offset(1)
        frmInterface.lblRead1.Text = Format(Val(frmInterface.lblRead1.Text), "##0.##")
        frmInterface.lblRead2.Text = volts(1) * factor(2)
        frmInterface.lblRead2.Text = Val(frmInterface.lblRead2.Text) - offset(2)
        frmInterface.lblRead2.Text = Format(Val(frmInterface.lblRead2.Text), "##0.##")
        frmInterface.lblRead3.Text = volts(2) * factor(3)
        frmInterface.lblRead3.Text = Val(frmInterface.lblRead3.Text) - offset(3)
        frmInterface.lblRead3.Text = Format(Val(frmInterface.lblRead3.Text), "##0.##")
        frmInterface.lblRead4.Text = volts(3) * factor(4)
        frmInterface.lblRead4.Text = Val(frmInterface.lblRead4.Text) - offset(4)
        frmInterface.lblRead4.Text = Format(Val(frmInterface.lblRead4.Text), "##0.##")
        frmInterface.lblRead5.Text = volts(4) * factor(5)
        frmInterface.lblRead5.Text = Val(frmInterface.lblRead5.Text) - offset(5)
        frmInterface.lblRead5.Text = Format(Val(frmInterface.lblRead5.Text), "##0.##")
        frmInterface.lblRead6.Text = volts(5) * factor(6)
        frmInterface.lblRead6.Text = Val(frmInterface.lblRead6.Text) - offset(6)
        frmInterface.lblRead6.Text = Format(Val(frmInterface.lblRead6.Text), "##0.##")
        frmInterface.lblRead7.Text = volts(6) * factor(7)
        frmInterface.lblRead7.Text = Val(frmInterface.lblRead7.Text) - offset(7)
        frmInterface.lblRead7.Text = Format(Val(frmInterface.lblRead7.Text), "##0.##")
        frmInterface.lblRead8.Text = volts(7) * factor(8)
        frmInterface.lblRead8.Text = Val(frmInterface.lblRead8.Text) - offset(8)
        frmInterface.lblRead8.Text = Format(Val(frmInterface.lblRead8.Text), "##0.##")
        frmInterface.lblRead9.Text = volts(8) * factor(9)
        frmInterface.lblRead9.Text = Val(frmInterface.lblRead9.Text) - offset(9)
        frmInterface.lblRead9.Text = Format(Val(frmInterface.lblRead9.Text), "##0.##")
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
        Me.StnSensorsTableAdapter.Update(Me.FlwStnSensorsDataSet.StnSensors)
        CalculateFactor()


    End Sub

    Private Sub NumericUpDown1_ValueChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles nBoard.ValueChanged
        initializeboard()
    End Sub
End Class
```

The Module:

frmModule1 is used for declaration of global variables and objects used by more than one class. In this paper, the classes referred to are, frmInterface and frmCalib. It also contains the method for factor calculation and Board initialization. The codes are shown below.

frmModule1 Codes:

The declarations and Board initialization sub are reusable from the Universal Library modified for this application [19].

```
Imports MccDaq
Module Module1
    Public parameter() As Double
    Public IniDigboard As String
    Public AnaDaqBoard As MccBoard
    Public regname As MccDaq.CounterRegister
    Public FOutDivider, FOutSource, Compare1, Compare2, LoadValue, TimeOfDay, ChipNum As
Integer
    Public Const NumPoints As Integer = 30000  ' Number of data points to collect
    Public Const FirstPoint As Integer = 0     ' set first element in buffer to transfer to array
    Public ADData(NumPoints) As System.UInt16  ' dimension an array to hold the input values
    Public UserTerm As Short                  ' flag to stop acquisition manually
    Public MemHandle, MemHandle2 As Integer
    Public MemHandleArray(16) As Integer
    Public ULStat, ULStat2 As ErrorInfo
    Public high_Value, low_Value, high_Volt, low_Volt As Double
    Public factor(16) As Double
    Public offset(16) As Double
    Public Range As New MccDaq.Range
    Public Options As New MccDaq.ScanOptions
    Public Rate As Integer
    Public Count As Integer
    Public VoltCount(16) As Double
    Public volts(16), avolts(32) As Single

    Public Sub initializeboard()
       AnaDaqBoard = New MccDaq.MccBoard(frmCalib.nBoard.Value)
       Dim AnaBoardName As String = AnaDaqBoard.BoardName
       AnaDaqBoard.FlashLED()
       ULStat = MccDaq.MccService.DeclareRevision(MccDaq.MccService.CurrentRevNum)
       Mem Handle = MccDaq.MccService.WinBufAlloc32Ex(NumPoints)
       If Mem Handle = 0 Then
          AnaBoardName = ULStat. Message
       End If
       frmCalib.lblmsg.Text = Ana Board Name
    End Sub
    Public Sub Calculate Factor()
       For J As Integer = 0 To 8

          If IsDBNull(frmCalib.DataGridView1.Item(8, J).Value) Then
frmCalib.DataGridView1.Item(8, J).Value = 0
          If IsDBNull(frmCalib.DataGridView1.Item(7, J).Value) Then
frmCalib.DataGridView1.Item(7, J).Value = 0
          If IsDBNull(frmCalib.DataGridView1.Item(6, J).Value) Then
frmCalib.DataGridView1.Item(6, J).Value = 0
```

```
        If IsDBNull(frmCalib.DataGridView1.Item(5, J).Value) Then frmCalib.DataGridView1.Item(5, J).Value = 0
        high_Value = Val(frmCalib.DataGridView1.Item(8, J).Value)
        low_Value = Val(frmCalib.DataGridView1.Item(7, J).Value)
        high_Volt = Val(frmCalib.DataGridView1.Item(6, J).Value)
        low_Volt = Val(frmCalib.DataGridView1.Item(5, J).Value)
        If (high_Volt - low_Volt) = 0 Then high_Volt = 1
        factor(Val(frmCalib.DataGridView1.Item(0, J).Value)) = (high_Value - low_Value) / (high_Volt - low_Volt)
        offset(Val(frmCalib.DataGridView1.Item(0, J).Value)) = factor(Val(frmCalib.DataGridView1.Item(0, J).Value)) * high_Volt
        offset(Val(frmCalib.DataGridView1.Item(0, J).Value)) = offset(Val(frmCalib.DataGridView1.Item(0, J).Value)) - high_Value
    Next J
  End Sub
End Module
```

frmUnit:

This class form handles the application of measurement unit to frmInterface and is shown in figure5. Here the units of measurement for process variables are entered. The units are typed into their respective textbox, apply, and saved in setting. However, at startup, the default units loaded to the interface

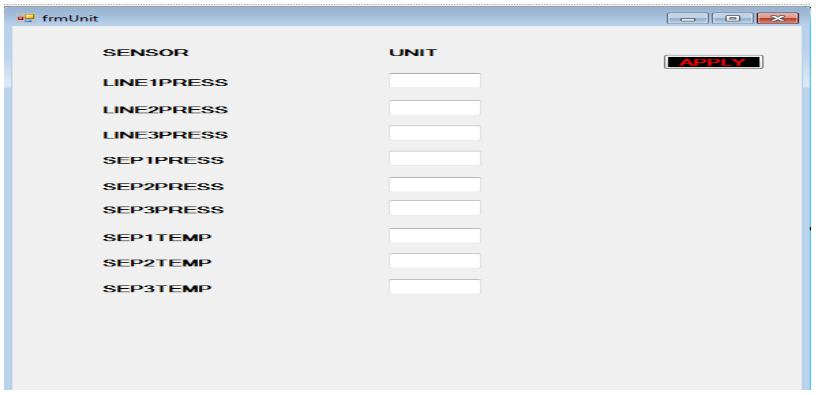


Figure5: frmUnit

frmUnit Codes

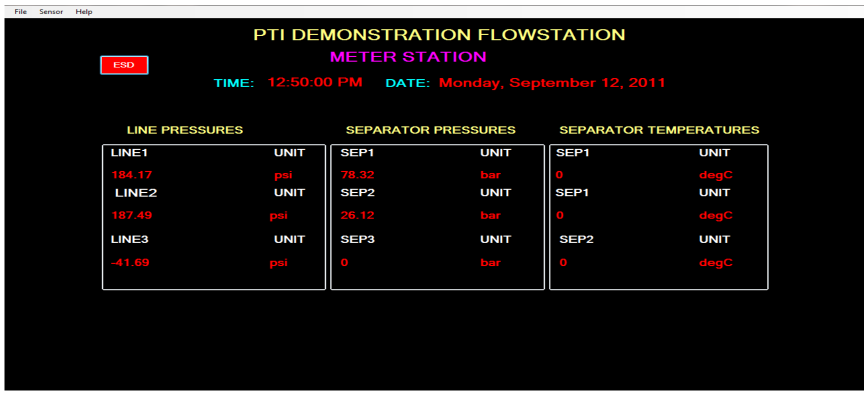
PUBLIC CLASS FRMUNIT

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    SaveSetting("Metering", "Unitsettings", "unit1", TextBox1.Text)
    SaveSetting("Metering", "Unitsettings", "unit2", TextBox2.Text)
    SaveSetting("Metering", "Unitsettings", "unit3", TextBox3.Text)
    SaveSetting("Metering", "Unitsettings", "unit4", TextBox4.Text)
```

```
        SaveSetting("Metering", "Unitsettings", "unit5", TextBox5.Text)
        SaveSetting("Metering", "Unitsettings", "unit6", TextBox6.Text)
        SaveSetting("Metering", "Unitsettings", "unit7", TextBox7.Text)
        SaveSetting("Metering", "Unitsettings", "unit8", TextBox8.Text)
        SaveSetting("Metering", "Unitsettings", "unit9", TextBox9.Text)
    End Sub
End Class
```

RESULT AND DISCUSSION

The complete system was tested using MccDaq Demo Board running on Windows 7 operating system. The results were accurate to the two decimal places. The result is shown in figure6.



PTI DEMONSTRATION FLOWSTATION
METER STATION
TIME: 12:50:00 PM   DATE: Monday, September 12, 2011

LINE PRESSURES		SEPARATOR PRESSURES		SEPARATOR TEMPERATURES	
LINE1	UNIT	SEP1	UNIT	SEP1	UNIT
184.17	psi	78.32	bar	0	degC
LINE2	UNIT	SEP2	UNIT	SEP1	UNIT
187.49	psi	26.12	bar	0	degC
LINE3	UNIT	SEP3	UNIT	SEP2	UNIT
-41.69	psi	0	bar	0	degC

Figure6: Result of the Simulation

The result shown in figure6 proves that the system will work perfectly when deployed to an industrial location. Industrial sensors are manufactured to give high accuracy. There outputs are usually an electrical quantity, voltage, which was obtained from the Demo board. The display system can be a giant display board mounted outside or plasma unit mounted in doors. Since the interface form has a runtime calibration of sensors and a provision for changing process variable units. The system can be deployed to any field irrespective of the number of variable because the codes are reusable.

CONCLUSION

This work was carried out using Petroleum Training Institute (PTI), Demonstration Flowstation, Effurun, Delta State in Nigeria as a case study for developing a metering System which uses electronic display board instead of gauges for accuracy and convenience. The Software was developed, with the codes of the four VB.NET form well written, tested, and results show that what we developed is found efficient and when deployed to the Flowstation we designed it for, produces expected results.

The advantages of this design over the existing pneumatic system are: *the meters location is not limited by distance since electrical signals can be processed and transmitted to any length; *the display system is more accurate because the data is 16bit wide and resolution is high; and *the system can be integrated into Distributed control systems and Fieldbus network. The disadvantage is that both the DAQ board and computer must be housed in a damp-free and dust proof air-conditioned environment. Considering the cost of pneumatic transmitters, tubing, gauges, and maintenance, with respect to sensors, twisted pair cable, and electronic display, our system is more economical.

However, for the system to operate efficiently with minimal down time seriously recommend that the following precautions should be taken:

a. A small hut should be erected in the Flowstation to house the electronic components. A Porto cabin may as well serve the same purpose.

b. A stable power supply is needed to power the device. A solar powered system will supply a constant power with minimal fluctuations.
c. Calibration should be carried out by well trained staff. Operators should not in any way temper with the codes, this could lead system malfunctioning.

REFERENCES

1. Measurement Computing (OEM & Embedded System) http://www.mccdaq.com/products/db.3000ubs.htm
2. Florain Lecher and Daniel Walter (2006) “An Introduction to Embedded Systems”
3. Edward Lee (2002) “Embedded Software” Advance in Computers, Academic Press, London, Vol. 56
4. J.A Cook and J.S. Fraudember (2007) “Embedded SoftwareArchitecture” EECS 461.
5. Juho Jaalinoja (2004) “Requirements Implementation in Embedded Software Development” VT Publications 526.82p+aa.7p
6. David Urting (2002) “A Tool for Component Based Design of Embedded Software
7. Louis Gomes and Aniko Costa (2004) “Embedded Systems: Introductory Course Supported by Remote Experiment” , Portugal
8. Bas Graaf et al (2003) “Embedded Software Engineering: The State of the Practice” IEEE Software, Nov-Dec, 2003, pp 61 – 69
9. Mehala Ciortea (2004) “Aspects Regarding the Types of Process Control Systems” Proceedings of the International Conference on Theory and Applications of Mathematics and Informatics (ICTAMI 2004), Thesoloniki, Greece, PP 90 – 95
10. H. Dale Beggs (1984) “Gas production Operations”. OGCI and Petroskills Publication, Tulsa, OK. USA, PP 219 – 234.
11. David A. T. Donohue and Karl R. Lang (1984. “Production Technology and Reservoir Management for the Geologist” GL302, IHRDC Video Product Sales, Boston, MAUSA, PP 25 – 52
12. Flanker Kuo (2008) “ Using Industrial Ethernet in the Oil and Gas Industry” Moxa White Paper ,Moxa Inc.
13. M. Sanchez et al (2005) “Unified automation with digital plant architecture improves gas plant and gathering system operation” Hyrocarbon Processing, pp 39 – 44
14. H. Dale Beggs (1984) “Gas production Operations”. OGCI and Petroskills Publication, Tulsa, OK. USA, PP 219 – 234.
15. David A. T. Donohue and Karl R. Lang (1984. “Production Technology and Reservoir Management for the Geologist” GL302, IHRDC Video Product Sales, Boston, MAUSA, PP 25 – 52.
16. Curtis D. Johnson (1977) “Process Control Instrumentation Technology” John Wiley & Sons, NY, USA
17. Measurement Computing (2010) “USB-1616FS User’s Guide” Document Revision 7.
18. Measurement Computing “Quick Start Guide” Measurement Computing Corporation, USA.
19. Measurement Computing (2008) “Universal Library Help”
20. Asim Abbasi (2005) “Microsoft Access 2007 Step by Step “ Takven Inc. South River, NJ, USA.

21. Thearan Willis, Brayan Newsome "Beginning Microsoft Visual Studio 2010" www.axishooting.net/books/Visual%20Basic/wrox.Begining.Visual.Basic.2010pdf

22. Cameron Wakefield, Henk-Evert Sonder, Wei Meng Lee (Series Editor) (2001) "VB.NET Developer's Guide" syngress publishing Inc. MA, USA

23. Gary Cornell, Jonathan Morrison (2002) "Programming VB.NET. A Guide for Experienced Programmer" www.spsamraj.weebly.com/upload/3/4/6/4/3464757/vb.netpdf

24. O'Reilly (2002)"Programming Visual Basic .NET " Dave Grundgeiger Publisher First Edition.

25. Alfred C Thompson II "Microsoft Visual Basic .NET Projects for the Classroom"

26. Indian Community Initiative ".NET Tutorial for Beginners"

27. Robin A. Reynolds-Haertle (2002) "OOP with Microsoft Visual Basic .NET and Microsoft Visual C# Step by Step" Microsoft Press