Executive Summary
Any design which requires one or more devise needs a microcontroller. A microcontroller allows for communication between any number of devices and furthermore allows for the implementation of an easy to ready/easy to understand Graphical User Interface (GUI). Through using Inter-Integrated Circuit (I^2C) bus specifications, one can communicate with any number of peripheral devices. Through the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) interface, one can connect a microcontroller to a GUI for easy to read output.
# Table of Contents

- Introduction ................................................................................................................................................................3
- Objective .....................................................................................................................................................................3
- Resources ...................................................................................................................................................................3
- Implementation and Results ......................................................................................................................................3
  - Parts and Setup ......................................................................................................................................................3
    - Figure 1: Microchip PIC18F4520 .........................................................................................................................4
    - Figure 2: Analog Devices AD7745 .......................................................................................................................4
    - Figure 3: Maxim MAX232 ...................................................................................................................................5
    - Table 1: Pin Connections ....................................................................................................................................5
- Microcontroller Programming ................................................................................................................................5
  - Example Microcontroller Code ...........................................................................................................................6
- User Interface Programming ............................................................................................................................... 13
  - Figure 4: Sample GUI ....................................................................................................................................... 14
  - Example Visual Basic Code .............................................................................................................................. 14
- Conclusions .............................................................................................................................................................. 18
- References ............................................................................................................................................................... 18
Introduction

Interfacing peripheral devices with a microcontroller is an important part of any design. Microcontrollers need to exchange data with various devices and allow the user to interpret the output. One such mode of data exchange is Inter-Integrated Circuit (I\(^2\)C) communication. Using both the Analog Devices AD7745 Capacitance to Digital Converter and the Microchip PIC 18F4520 as examples, this application note will demonstrate the method of communication, via I\(^2\)C, as well as serial communication as a means to interpret capacitive sensor changes.

Objective

The objective of this application note is to demonstrate how to communicate between the Analog Devices AD7745 Capacitance to Digital Converter and the user via the Microchip PIC 18F4520 microcontroller. The concepts demonstrated in this application note can be extended to any I\(^2\)C peripheral device and a compatible I\(^2\)C microcontroller.

Resources

For this application note, the following resources should be on hand:

- Microchip MPLAB IDE v8.42 or later
- Microchip ICD2 or equivalent debugger
- Microchip PIC 18F4520 Data Sheet
- Analog Devices AD7745 Data Sheet
- RS232 Wiring Schematics
- Microsoft Visual Studio 2008 or later

The reader must also have a basic working knowledge of C programming and Visual Basic programming.

Implementation and Results

Parts and Setup

The first step in design is to wire all of the components together. The following parts are required:

- Microchip PIC 18F4520
- Analog Devices AD7745 + Expansion Board
- Breadboard
- +5 (V) Power Supply Unit
- 10 MHz Crystal Oscillator
- MAX232 Chip
- RS232 Interface
- Microchip ICD2
- 1 Yellow LED
- 1 Red LED
- 1 Green LED
- 3 220 Ω Resistors
- 4 1 μF Capacitors

The pin layout figures below help us see what needs to be connected. The primary connections are listed in Table 1: Pin Connections, and the remainder of the connections can be derived from the datasheets.

Figure 1: Microchip PIC18F4520

Figure 2: Analog Devices AD7745
Microcontroller Programming

The second step is programming the microcontroller. With the PIC18F4520, we must include a few reference files. The first is `<p18cxxx.h>`, which tells the compiler which family of microcontroller we are using. The second, `<usart.h>`, tells the compiler that we need to communicate across a serial port. The program must communicate with the User Interface (described below) in order for the user to see meaningful output. The third and most important include file is `<i2c.h>`. This file allows us to communicate with any number of I\(^2\)C compatible peripheral devices. The example code below uses the AD7745 as the only peripheral device.

The I\(^2\)C interface has a few key routines:
• OpenI2C() – Opens the bus between the master (microcontroller) and the slaves (all peripheral devices)
• StartI2C() – Begins the communication segment over the I\(^2\)C bus
• StopI2C() – Ends the communication segment over the I\(^2\)C bus
• WriteI2C() – Sends data over the bus. This is the most important I\(^2\)C routine for a couple reasons. The first Write after a Start command will hold the address of the peripheral device being written to. The second Write command will hold the address of the register to be modified on the peripheral device.
• ReadI2C() – Receives data from the bus. This routine is called after the second Write command when data input is needed from a peripheral device.
• RestartI2C() – This command reinitializes the I\(^2\)C bus line.
• IdleI2C() – This command is used in between all I\(^2\)C statements to account for any delay from the peripheral devices. Usually, the microcontroller can process commands much quicker than an I\(^2\)C device, so a delay must be put in place to ensure correct operation of the design.

With these routines, you can communicate with any I\(^2\)C compatible peripheral device. The initialize() and get_data() subroutines below illustrate the I\(^2\)C operation. The rx_handler() subroutine illustrates the serial (usart) operation.

Example Microcontroller Code

```c
#include "pl8cxxx.h"
#include "usart.h"
#include "ADC.h"
#include "i2c.h"
#include "stdio.h"
#include "stdlib.h"
#pragma config LVP=OFF
#pragma config WDT=OFF
void rx_handler (void); //Declare the ISR function
void initialize (void); //Declare the Initialize subroutine for the C-D Converter
void get_data (void); //Declare the retrieve data subroutine for the C-D Converter

/* PARAMETER DECLARATION TABLE */
// The following table describes the boundaries for each type of object that can be sensed.
// All of the boundaries are integer variables with HEX representation. (0xVALUE)
// Water is represented by:  (1) – (6)
// Leaf is represented by:  (7) – (8)
```
// Finger is represented by: (9) - (10)
// (1) MIST Lower boundary
int a = 0x000000;
// (2) MIST Upper boundary, DRIZZLE Lower boundary
int b = 0x000000;
// (3) DIZZLE Upper boundary, LIGHT RAIN Lower boundary
int c = 0x000000;
// (4) LIGHT RAIN Upper boundary, HEAVY RAIN Lower boundary
int d = 0x000000;
// (5) HEAVY RAIN Upper boundary, DOWNPOUR Lower boundary
int e = 0x000000;
// (6) DOWNPOUR Upper boundary
int f = 0x000000;
// (7) LEAF Lower boundary
int g = 0x000000;
// (8) LEAF Upper boundary
int h = 0x000000;
// (9) FINGER Lower boundary
int i = 0x000000;
// (10) FINGER Upper boundary
int j = 0x000000;
//-------------------------------------------------------------------
long int count;
int adc_result, dloop;
unsigned char data;
long int loop;
long int CapData, temp1, temp2, temp3;
double Offset, Offset2;
int cast;
unsigned char bits = '0x00';
unsigned char bits2 = '0x00';

unsigned char StatusReg = '0xFF'; // Creates a byte for use in analyzing the
status register of the AD7745
unsigned char CapDacValue = '0x00'; // Temporary byte for reading the CAPDAC value
to ensure proper operation

unsigned char CapHIGH = '0x00'; // Byte to receive MSB byte capacitive data
unsigned char CapMED = '0x00'; // Byte to receive the next (middle) capacitive
data
unsigned char CapLOW = '0x00'; // Byte to receive LSB byte capacitive data
unsigned char VtHIGH = '0x00'; // Byte to receive MSB byte capacitive data
unsigned char VtMED = '0x00'; // Byte to receive the next (middle) capacitive
data
unsigned char VtLOW = '0x00'; // Byte to receive LSB byte capacitive data

/*------------------------------------------------------------------
Name: main()
Input: none
Output: none
Purpose: Give the program a starting point
--------------------------------------------------------------------------------*/
void main()
{
    TRISC = 0x00; //turn on tri-state register and make all output pins
PORTC = 0x00; //make all output pins LOW
OpenI2C( MASTER, SLEW_OFF);
SSPADD = 0x3F;

OpenUSART (USART_TX_INT_OFF & USART_RX_INT_ON &
    USARTASYNCH_MODE & USART_EIGHT_BIT &
    USART_CONT_RX & USART_BRGH_LOW, 63);
RCONbits.IPEN = 1; /* Enable interrupt priority */
IPR1bits.RCIP = 1; /* Make receive interrupt high priority */
INTCONbits.GIEH = 1; /* Enable all high priority interrupts */
OpenADC(ADC_FOSC_32 & ADC_RIGHT_JUST & ADC_12_TAD,
    ADC_CH0 & ADC_INT_OFF, 0); //open adc port for reading
ADCON1 =0x00; //set VREF+ to VDD and VREF- to GND (VSS)
TRISD = 0x04;
PORTDbits.RD0 = 0; //GREEN
PORTDbits.RD1 = 0; //YELLOW
PORTDbits.RD3 = 0; //RED
dloop = 0;
while(1);
PORTDbits.RD0 = 1; //GREEN
PORTDbits.RD1 = 1; //YELLOW
PORTDbits.RD3 = 1; //RED
}

/*------------------------------------------------------------------
Name:    rx_int()
Input:   void
Output:  none
Purpose: Default interrupt code
------------------------------------------------------------------*/

#pragma code rx_interrupt = 0x8
void rx_int (void)
{
    _asm goto rx_handler _endasm
}
#pragma code
#pragma interrupt rx_handler

/*------------------------------------------------------------------
Name:    rx_handler()
Input:   void
Output:  none
Purpose: Handle all interrupts from the VB App
------------------------------------------------------------------*/

void rx_handler (void)
{
    unsigned char cc;
    cc = getcUSART(); //get a single character off the USART line
    while(BusyUSART());
    PORTDbits.RD3 = 1; //RED
    if(cc == '1')
    {
        initialize();
        while(BusyUSART());
        putcUSART (CapHIGH); //put a single character on the USART line
        while(BusyUSART());
        putcUSART (CapMED); //put a single character on the USART line
    }
while(BusyUSART());
putcUSART (CapLOW); //put a single character on the USART line
while(BusyUSART());
PORTDbits.RD0 = 1;    //GREEN
}
else if(cc = 'g')
{
    get_data();
    while(BusyUSART());
    putcUSART (StatusReg); //put a single character on the USART line
    while(BusyUSART());
    putcUSART (CapHIGH); //put a single character on the USART line
    while(BusyUSART());
    putcUSART (CapMED); //put a single character on the USART line
    while(BusyUSART());
    putcUSART (CapLOW); //put a single character on the USART line
    while(BusyUSART());
    putcUSART (VtHIGH); //put a single character on the USART line
    while(BusyUSART());
    putcUSART (VtMED); //put a single character on the USART line
    while(BusyUSART());
    putcUSART (VtLOW); //put a single character on the USART line
    while(BusyUSART());
}
PORTDbits.RD3 = 0;    //RED
PIR1bits.RCIF = 0; //reset the ISR flag.
}

/*------------------------------------------------------------------
Name:    initialize()
Input:   none
Output:  none
Purpose: Send the appropriate I2C commands to initialize the
         C-D Converter
------------------------------------------------------------------*/
void initialize (void)
{
    StartI2C();    // Start Sequence
    IdleI2C();
    WriteI2C(0x90); // AD7745 Address + WRITE
    IdleI2C();
    WriteI2C(0xBF); // RESET COMMAND TO AD7745
    IdleI2C();
    StopI2C();     // Stop Sequence

    for (loop=0x001; loop<0x834; loop++); // Delays 210 us after the AD7745 Reset
         (required by C-D converter)

    StartI2C();
    IdleI2C();
    WriteI2C(0x90);
    IdleI2C();
    WriteI2C(0x09); // Sets ADDRESS POINTER REGISTER of AD7745 to EXC SETUP REG
    IdleI2C();
    WriteI2C(0x4B); // Sets EXC SETUP REG to use amplitude +- Vdd/2, EXCON=1,
                    and use EXCA
    IdleI2C();
StopI2C();
StartI2C();
IdleI2C();
WriteI2C(0x90);
IdleI2C();
WriteI2C(0x07); // Sets ADDRESS POINTER REGISTER of AD7745 to CAP SETUP REG
IdleI2C();
WriteI2C(0x80); // Sets CAP SETUP REG to enable capacitive channel 1 in single-ended mode. CHOP disabled
IdleI2C();
StopI2C();
StartI2C();
IdleI2C();
WriteI2C(0x90);
IdleI2C();
WriteI2C(0x08); // Sets ADDRESS POINTER REGISTER of AD7745 to VOLTAGE SETUP REG
IdleI2C();
WriteI2C(0x81); // Sets VOLTAGE SETUP REG to enable capacitive channel 1 in single-ended mode. CHOP disabled
IdleI2C();
StopI2C();
StartI2C();
IdleI2C();
WriteI2C(0x90);
IdleI2C();
WriteI2C(0x0B); // Sets ADDRESS POINTER REGISTER of AD7745 to CAP DAC A REG
IdleI2C();
WriteI2C(0xF1); // Sets CAP DAC A REG to full value of capacitance ~16 pF
IdleI2C();
StopI2C();
StartI2C();
IdleI2C();
WriteI2C(0x90);
IdleI2C();
WriteI2C(0x0A); // Sets ADDRESS POINTER REGISTER of AD7745 to CONFIGURATION REG
IdleI2C();
WriteI2C(0xA1); // Sets CONFIGURATION REG to enable CONTINUOUS CONVERSION at 62 ms conversion time
IdleI2C();
StopI2C();
while(StatusReg!=0x00) get_data(); //Get the initial value of the Capacitive Sensor
```c
/*------------------------------------------------------------------
Name:    get_data()
Input:   none
Output:  none
Purpose: Send the appropriate I2C commands to get data from the C-D Converter
*/
```
void get_data (void)
{
    StartI2C(); //-------- THIS SEQUENCE POLLS THE STATUS REGISTER
    IdleI2C();
    WriteI2C(0x90); // AD7745 Address + WRITE
    IdleI2C();
    WriteI2C(0x00); // Sets ADDRESS POINTER REGISTER of AD7745 to STATUS REG
    IdleI2C();
    RestartI2C(); // Restart command to begin reading
    IdleI2C();
    WriteI2C(0x91); // AD7745 Address + READ
    IdleI2C();
    StatusReg = ReadI2C(); // Reads the STATUS REG value and stores it in char
    "StatusReg"
    IdleI2C();
    NotAckI2C(); // Since reading only ONE BYTE, NO ACKNOWLEDGE is sent back
to prevent auto-increment of address pointer
    IdleI2C();
    StopI2C();
    for (loop = 1; loop < 4000; loop++)
    {
        if (StatusReg>0x07 && StatusReg<0x10)
        {
            // If the EXCERR bit goes high in the Status Register, the EXC cannot
            // be driven
            // We want to then stop looping and perhaps turn on a red LED to
            indicate an error
            PORTDbits.RD3 = 1; //RED
        }
        else if ((StatusReg == 0x00)||(StatusReg == 0x01)||(StatusReg == 0x02)) // Capacitive data is ready to be read
        {
            PORTDbits.RD1 = 1; //YELLOW
            if(StatusReg == 0x00)
            {
                StartI2C();
                IdleI2C();
                WriteI2C(0x90); // AD7745 Address + WRITE
                IdleI2C();
                WriteI2C(0x01); // Sets ADDRESS POINTER REGISTER of AD7745 to CAP
                DATA REGISTER 1
                IdleI2C();
                RestartI2C();
                IdleI2C();
                WriteI2C(0x91); // AD7745 Address + READ
                IdleI2C();
                CapHIGH = ReadI2C();
                IdleI2C();
                AckI2C();
                IdleI2C();
                CapMED = ReadI2C();
                IdleI2C();
                AckI2C();
                IdleI2C();
                CapLOW = ReadI2C();
            }
        }
    }
}
IdleI2C();
AckI2C();
IdleI2C();
VtHIGH = ReadI2C();
IdleI2C();
AckI2C();
IdleI2C();
VtMED = ReadI2C();
IdleI2C();
AckI2C();
IdleI2C();
VtLOW = ReadI2C();
IdleI2C();
NotAckI2C();
IdleI2C();
StopI2C();
}
else if(StatusReg == 0x01)
{
StartI2C();
IdleI2C();
WriteI2C(0x90); // AD7745 Address + WRITE
IdleI2C();
WriteI2C(0x04); // Sets ADDRESS POINTER REGISTER of AD7745 to CAP
DATA REGISTER 1

IdleI2C();
RestartI2C();
IdleI2C();
WriteI2C(0x91); // AD7745 Address + READ
IdleI2C();
VtHIGH = ReadI2C();
IdleI2C();
AckI2C();
IdleI2C();
VtMED = ReadI2C();
IdleI2C();
AckI2C();
IdleI2C();
VtLOW = ReadI2C();
IdleI2C();
NotAckI2C();
IdleI2C();
StopI2C();
}
else if(StatusReg == 0x02)
{
StartI2C();
IdleI2C();
WriteI2C(0x90); // AD7745 Address + WRITE
IdleI2C();
WriteI2C(0x01); // Sets ADDRESS POINTER REGISTER of AD7745 to CAP
DATA REGISTER 1

IdleI2C();
RestartI2C();
IdleI2C();
WriteI2C(0x91); // AD7745 Address + READ
IdleI2C();
CapHIGH = ReadI2C();
        IdleI2C();
AckI2C();
        IdleI2C();
        CapMED = ReadI2C();
        IdleI2C();
AckI2C();
        IdleI2C();
        CapLOW = ReadI2C();
        IdleI2C();
NotAckI2C();
        IdleI2C();
        StopI2C();
    }
    PORTDbits.RD1 = 0;  //YELLOW
    }

**User Interface Programming**

The third step is programming the User Interface. When programming an interface, there are two key components; the Graphical User Interface (GUI) and the backend code. In Figure 4: Sample GUI, the user can see a very simple and easy to use interface. Fields are grouped together based on their function. All program control is placed in the “Controls” group box, and all feedback data is placed in the “Data” group box. The remaining parts of the program are visual representations/interpretations of the data. The front end GUI is designed to be as simple and easy to use as possible. The backend, on the other hand, contains much more code to interface with the microcontroller. The setRS232() routines allow serial communications between the microcontroller and the Visual Basic application.
**Example Visual Basic Code**

```vbnet
Public Class Main
    Inherits System.Windows.Forms.Form
    Dim setRs232 As New Rs232
    Dim serial_in As String
    Dim x As Integer
    Dim HexVal As String
    Dim start As DateTime
    Dim time As TimeSpan
    Dim BaseCapacitance As Decimal
    Declare Sub Sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

    Private Sub Main_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        With setRs232
            .Port = 1
            .BaudRate = 2400 'Configured for a 10 MHz clock
            .DataBit = 8
            .StopBit = Rs232.DataStopBit.StopBit_1
            .Parity = Rs232.DataParity.Parity_None
            .Timeout = 10000
        End With
        setRs232.Open()
    ButtonReset_Click(sender, e)
```
End Sub

Private Sub ButtonStart_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ButtonStart.Click
    ChartCapGraph.Series("Series1").Points.Clear()
    ChartCapGraph.Legend.Enabled = False
    ChartCapGraph.ChartAreas("Default").AxisX.Title = "Time (s)"
    ChartCapGraph.ChartAreas("Default").AxisY.Title = "Capacitance (fF)"
    Dim Z As Integer
    Dim Max As Decimal
    Dim Min As Decimal
    Dim Capacitance As Decimal
    Dim Temperature As Decimal

    start = DateTime.Now
    For Z = 1 To (Val(TextBoxTimer.Text) * 11.111111)
        setRs232.Write("g")
        serial_in = setRs232.InputStreamString
        x = Asc(CChar(serial_in))
        If (x < 16) Then
            HexVal = "0" + Hex$(x)
        Else
            HexVal = Hex$(x)
        End If
        TextBoxStatus.Text = "0x" + HexVal
        setRs232.Read(1)
        serial_in = setRs232.InputStreamString
        x = Asc(CChar(serial_in))
        If (x < 16) Then
            HexVal = "0" + Hex$(x)
        Else
            HexVal = Hex$(x)
        End If
        TextBoxStatus.Text = "0x" + HexVal
        setRs232.Read(1)
        serial_in = setRs232.InputStreamString
        x = Asc(CChar(serial_in))
        If (x < 16) Then
            HexVal = HexVal + "0" + Hex$(x)
        Else
            HexVal = HexVal + Hex$(x)
        End If
        TextBoxStatus.Text = "0x" + HexVal
        setRs232.Read(1)
        serial_in = setRs232.InputStreamString
        x = Asc(CChar(serial_in))
        If (x < 16) Then
            HexVal = HexVal + "0" + Hex$(x)
        Else
            HexVal = HexVal + Hex$(x)
        End If
        TextBoxStatus.Text = "0x" + HexVal

        TextBoxHex.Text = "0x" + HexVal
Capacitance = CDec("&H" + HexVal) * (0.0000004882812791) - 4.096

' Convert the Capacitance to a readable value

setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = "0" + Hex$(x)
Else
    HexVal = Hex$(x)
End If

setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = HexVal + "0" + Hex$(x)
Else
    HexVal = HexVal + Hex$(x)
End If

setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = HexVal + "0" + Hex$(x)
Else
    HexVal = HexVal + Hex$(x)
End If

TextBoxHex2.Text = "0x" + HexVal

Temperature = CDec("&H" + HexVal) / 2048 - 4096
Temperature = Math.Round(Temperature, 2)
TextBoxTemperature.Text = Temperature.ToString + " (C)"

Capacitance = Math.Round(Capacitance, 5)
If (Z = 1) Then
    Max = Capacitance
    Min = Capacitance
End If
If (Capacitance > Max) Then
    Max = Capacitance
End If
If (Capacitance < Min) Then
    Min = Capacitance
End If
TextBoxCapacitance.Text = (Capacitance * 1000).ToString + " (fF)"
time = DateTime.Now.Subtract(start)
ChartCapGraph.Series("Series1").Points.AddXY(time.TotalSeconds,
Capacitance * 1000)
TextBoxCapacitance.Refresh()
TextBoxHex.Refresh()
TextBoxStatus.Refresh()
TextBoxHex2.Refresh()
TextBoxTemperature.Refresh()

If ((Math.Abs(Capacitance - BaseCapacitance) * 1000) < 25) Then
ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 25) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) < 65) Then
    TextBoxWhat.Text = "One Drop"
ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 65) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) < 105) Then
    TextBoxWhat.Text = "Two Drops"
ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 105) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) < 145) Then
    TextBoxWhat.Text = "Three Drops"
ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 145) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) < 195) Then
    TextBoxWhat.Text = "Four Drops"
ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 195) And
((Math.Abs(Capacitance - BaseCapacitance) * 1000) < 500) Then
    TextBoxWhat.Text = "Covered"
ElseIf ((Math.Abs(Capacitance - BaseCapacitance) * 1000) > 500) Then
    TextBoxWhat.Text = "Finger/Hand"
End If
TextBoxWhat.Refresh()
Next
Sleep(1)

ChartCapGraph.ChartAreas("Default").AxisY.Minimum = Min * 1000
ChartCapGraph.ChartAreas("Default").AxisY.Maximum = Max * 1000
ChartCapGraph.ChartAreas("Default").AxisX.Minimum = 0
ChartCapGraph.ChartAreas("Default").AxisX.Maximum = time.TotalSeconds

Dim temp As String
temp = PictureBoxWiper.ImageLocation
temp = temp.Remove(0, 39)
temp = temp.Remove(1, 4)
Dim y As Integer
y = temp
y = y + 1
If y = 6 Then y = 1
PictureBoxWiper.ImageLocation = "M:\My Documents\480\Final\App\App\Pics" + y.ToString + ".jpg"
PictureBoxWiper.Refresh()
End Sub

Private Sub ButtonReset_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ButtonReset.Click
    setRs232.Write("i") 'Used to initialize the C-D Converter (interrupt for the Microcontroller)
    TextBoxCapacitance.Clear()
    TextBoxHex.Clear()
    TextBoxStatus.Clear()
    setRs232.Read(1)
    serial_in = setRs232.InputStreamString
    x = Asc(CChar(serial_in))
    If (x < 16) Then
        HexVal = "0" + Hex$(x)
    Else
        HexVal = Hex$(x)
    End If
setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = HexVal + "0" + Hex$(x)
Else
    HexVal = HexVal + Hex$(x)
End If
setRs232.Read(1)
serial_in = setRs232.InputStreamString
x = Asc(CChar(serial_in))
If (x < 16) Then
    HexVal = HexVal + "0" + Hex$(x)
Else
    HexVal = HexVal + Hex$(x)
End If

BaseCapacitance = CDec("&H" + HexVal) * (0.0000004882812791) - 4.096

End Sub
End Class

Conclusions

Through both serial and I²C communications, a microcontroller can communicate with any number of peripheral devices and communicate the data to the user. The serial interface allows for communication between the microcontroller and the user and allows for a much cleaner and easier to interpret stream of data. The I²C interface allows the microcontroller to control various peripheral devices through a specific set of commands and routines.

References

Analog Devices AD7745 Data Sheet


Microchip PIC18F4520 Data Sheet