St. Cloud State University

Department of Electrical and Computer Engineering

Final Project Report

Real-time Digital Signal Processing using FIR filter, Wireless Digital Oscilloscope, and Spectral Analyzer

ECE 422 Microcontroller System Design– Prof. Yi Zheng

December 17, 2021

By Suhaib Abugdera

Table of Contents: **page number**

Background

This final project is to design and develop a real-time digital signal processing using FIR filter and wireless digital oscilloscope and spectral analyzer using PIC24FV16KM202, Bluetooth, and C# GUI. The project is based on the labs and lectures learned during class time. The design contains several components that are used to make the project a success and requires some techniques like transmitting and receiving data from PIC C to GUI C# and from GUI to PIC. Also, a technique of Zedgraph is used for real-time plotting and finally, the bootloader is used to load the operating system or runtime environment to add programs to memory and provide access for components. These techniques were learned from previous labs and lectures. Several features were learned like, registers, GPIO, memory, timer, PWM, internal and external interrupts, ADC, and UART communication. For this project, C language and C# GUI will be used to design and implement a given specification using the skills that were learned from previous labs and lectures and apply it in this midterm project to indicate that the outcome of the course is achieved, and the students can solve problems using these skills.

Objectives

To design and develop a real-time digital signal processing using FIR filter and wireless digital oscilloscope and spectral analyzer using PIC24FV16KM202, Bluetooth, and C# GUI. The system development is based on a communication protocol that works between the PIC and C# via Bluetooth to digitize analog signals and plot time-domain signal and frequency signal.

System requirements and design specifications

Basic requirements:

The required devices in this system are as follow:

1. Breadboard
2. A power supply
3. PIC24FV16KM202
4. ICD-U64
5. PCW C compiler
8. Bluetooth module
9. Bluetooth adapter (doesn't need if the laptop is used)
10. 74LS20 NAND

11. Wire jumpers

The design specifications are based on levels from basic to advanced levels as follow:

The Basic requirements are to digitize an analog signal, plot digitized data in a continuous mode without distortion. The system must be implemented on a breadboard with knowledge from previous labs. Additional design specifications are listed from basic to higher levels like the following:

Level I:

Based on Lab 9 and Lab 10, on the GUI to select:

1) The PIC digitizes a rectangular wave of 10, 50, 100 Hz that may be generated by a function generator, or another PIC, or PWD (without interrupt), with a sampling frequency of 2 kHz. Using textbox and GUI buttons for the various functions. Note that:

a- A very high sampling frequency for a very low-frequency signal requires an FIR filter having long taps. Thus, you may not be able to implement an FIR filter with 2 kHz for a 10 Hz signal with 64 taps. In that case, you may want to down the sampling 2 kHz to a lower sampling such as 200 Hz so that the filter is acceptable, and the processing time is within the time limit.

b- The procedure is common in practice where the hardware is fixed, but the software can be adjusted for different signals.

2) Design and implement three FIR LPFs to filter the digitized data for extracting 10, 50, and 100 Hz sinusoid waves, and plot the results on GUI continuously. Note that the filter coefficients may be the same for the three filters, but the sampling frequencies are different in the PIC.

3) The specification of the filter design is open, but the filtered waveform should be close to expected with minimum distortion and minimum other harmonics. The designs of the filters must be documented with justifications.

4) The GUI display of the filtered output should be as fast as possible. Use a protocol for transmission and plot. The plot should spontaneously represent the waveform situation. Do not use DAC or LCD.

5) There are GUI buttons to select an FIR filter, start the plot and stop the plot. The coefficients of the filter must be 16 bits.

6) Download your filter coefficients from C#, once one of three filters is selected. You may download every coefficient by using two bytes and then recombine them in the PIC using shift operation. The coefficients are NOT the part of the PIC code, but the part of the C# code.
7) Display both sampled input and filtered data on the same graph.
8) Correct unit label for time and voltage.

Level II:

Based on Level I (addition)

1) Add three BPF having center frequencies: 30, 150, 300 Hz to extract sinusoid waves of 30, 150, and 300 Hz, with give rectangular waves of 10, 50, and 100 Hz, and plot the results on GUI continuously. Note that the filter coefficients may be the same for the three filters, but the sampling frequencies are different in the PIC.
2) Download your filter coefficients from C#, once a filter of three is selected.
3) The specification of the filter design is open, but the filtered waveform should be close to expected with minimum distortion and minimum other harmonics. Document your filter design with justifications,
4) Selected proper sampling frequencies an FIR filter: 200 Hz, 1 kHz, 2 kHz. The selection of the sampling frequency should be independent hardware to above by using textbox and GUI buttons.
5) The GUI display of the filtered output should be as fast as possible. Use a protocol for transmission and plot. The plot should spontaneously represent the waveform situation.
6) There are GUI buttons to select an FIR filter, start the plot and stop the plot.
7) Display both sampled input and filtered data on the same graph.
8) Correct unit label for time and voltage.

Level III:

Display the maximum, the minimum, and the frequencies of the time plots on designated GUI textboxes, sliding bar for trigger control, correct X and Y axes labeling.

Level IV:

DFT implementation and plot using C# for both input of the ADC and output of the filter. There will be two graphic plot boxes:

1) time plots of input and output of the digital filter,

2) the Fourier Transforms of the input and output of the filter.

Level V:

To be able to use Bluetooth to download the PIC code using the boot loader.

Design and Implementation

The design was approached with a very simple code as done in the previous labs. Starting with defining the PIC24FV16KM202 with PIC C and defining devices and initializing the global variables needed for the code design. After initializing and defining devices as was done in previous labs, the code design is starting with creating a loop for normalization as follow:

The loop is to digitize data with 8 bits so that it can transfer data byte by byte or char (is ASCII) this will allow the system to be very fast (making the program very simple without interrupt) and go through the loop one to digitize ADC converter and configure ADC, then inside loop two command one ADC to a character and digitize 200 one after another(just digitize into a character array. Then, transmit one at a time to C# after doing normalization.

Before sending the coefficients from C#, the testing for the chip is conducted to see if the data will be transferred between GUI and the chip. See appendix (A) for a screenshot of a tera terminal sending data. This is an easy way to make sure that the code in PIC C is really working and move to C# GUI to do the rest.

Running average trigger (it can be used depending on how the system is working) by adding a trigger to PIC C; choosing 2.5V; the first data status = 0; if the data is less than 2.5, just keep status =1; if bigger than 2.5, do Status =2 and then, enter that loop to digitize 200.

If the first data is above 2.5, then make status =0 that will continue to stay positive, stay status = 0 until the data is less than 2.5, then change the status =1. See Figure (1) below for the state machine diagram.

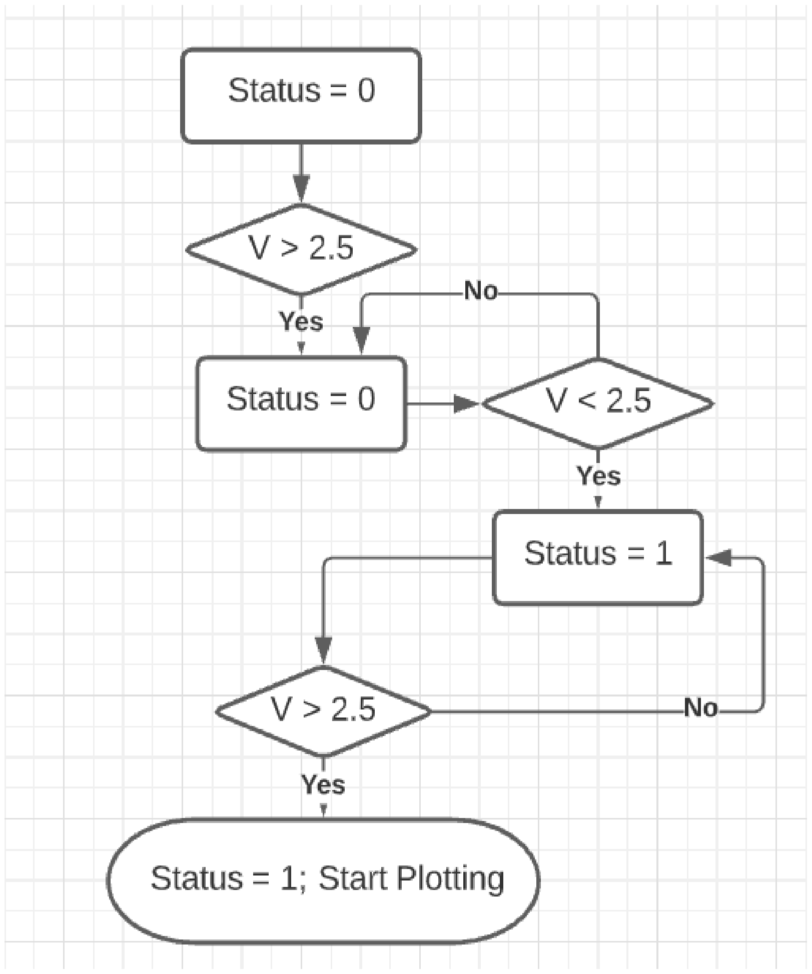
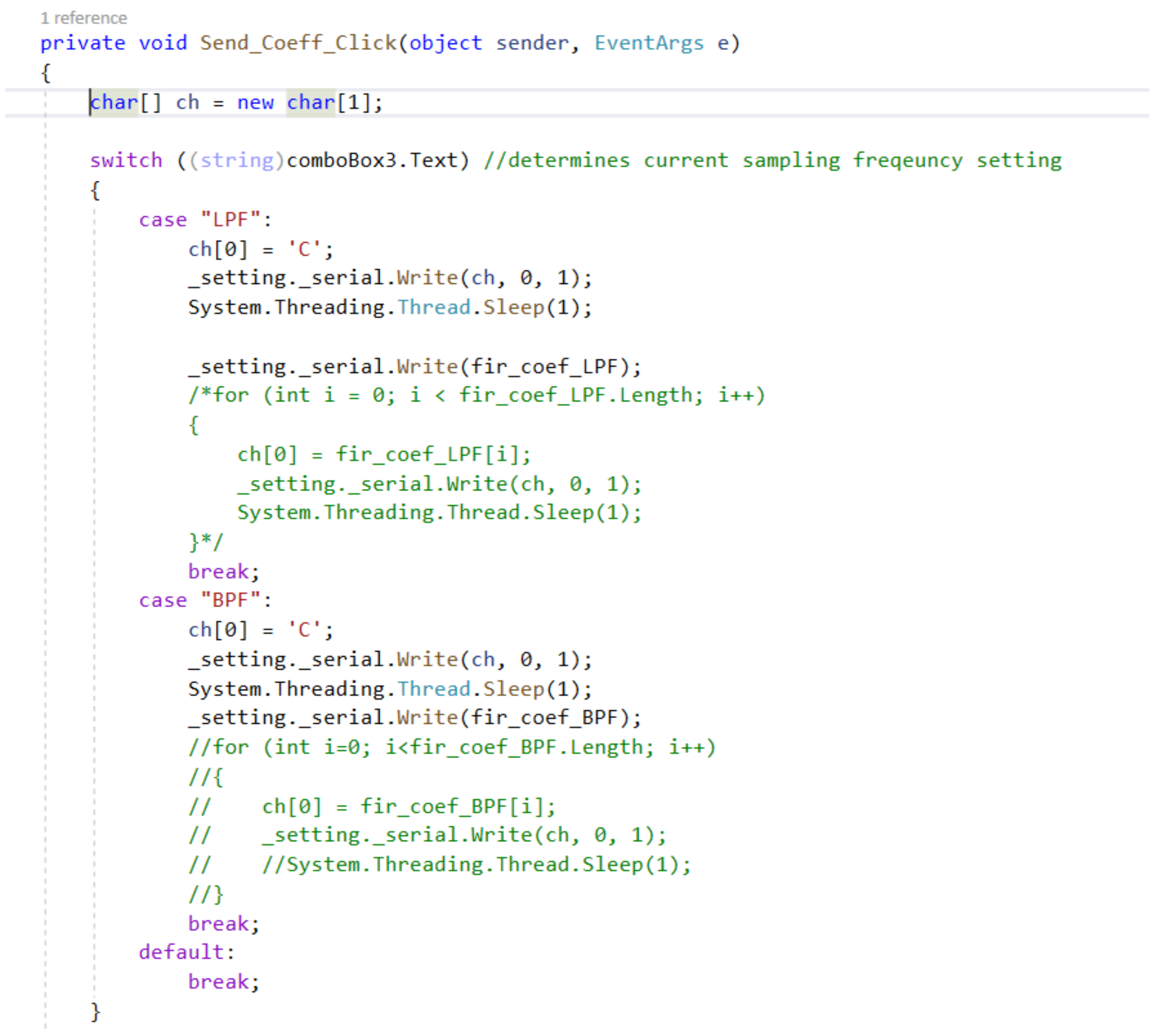


Figure (1) State diagram for the trigger

Some glitches will occur, but by doing trigger and using running average, the current data will be equal to the average of the previous 3 or 2. Start to sample frequency and timer inside the interrupt sample data to array to 6 data then start transmission. The maximum frequency needs to be in a different loop not on the time.

Finally, building GUI C# to send coefficients to the chip and provides the user with options to choose LPF to extract 10, 50, and 100 Hz signals and BPF to extract 30, 150, and 300 Hz signals from input square wave signal. In C# GUI, the FDA tools coefficients are moved there and to transmit all data from GUI is by sending the coefficient one time in an array as seen in the figure (2) below for the array in GUI that does send the coefficient to PIC microcontroller. There is an uncommented for loop to send the coefficients. This was the first trial, and it does send coefficients very slow, and it takes 8 seconds to run. Changing the for loop with the array was very fast.



```
1 reference
private void Send_Coeff_Click(object sender, EventArgs e)
{
    char[] ch = new char[1];

    switch ((string)comboBox3.Text) //determines current sampling freqeuncy setting
    {
        case "LPF":
            ch[0] = 'C';
            _setting._serial.Write(ch, 0, 1);
            System.Threading.Thread.Sleep(1);

            _setting._serial.Write(fir_coef_LPF);
            /*for (int i = 0; i < fir_coef_LPF.Length; i++)
            {
                ch[0] = fir_coef_LPF[i];
                _setting._serial.Write(ch, 0, 1);
                System.Threading.Thread.Sleep(1);
            }*/
            break;
        case "BPF":
            ch[0] = 'C';
            _setting._serial.Write(ch, 0, 1);
            System.Threading.Thread.Sleep(1);
            _setting._serial.Write(fir_coef_BPF);
            //for (int i=0; i<fir_coef_BPF.Length; i++)
            //{
            //    ch[0] = fir_coef_BPF[i];
            //    _setting._serial.Write(ch, 0, 1);
            //    //System.Threading.Thread.Sleep(1);
            //}
            break;
        default:
            break;
    }
```

Figure (2) Sending coefficients code in GUI.

The input signal is digitized using 8-bit ADC and processed using 16-bit coefficients from the FDA tool of MATLAB. Also, users can choose different sampling frequencies and enter a wide range of offset and shift values to control output from the system. Moreover, users are provided with a GUI program developed in C# which helps users to easily control the system and visualize the output in the GUI. See figure (3) below for the system block diagram. After this simple code and normalization for the first level, the C# GUI started to be designed to plot the first part and then, continue to do the next levels by making sure the communication protocol is working.

System Block Diagram (Hardware)

In figure (3) below shows a block diagram for overall system components for this project. The diagram shows how these components are connected and communicated with each other as input, output, or both.

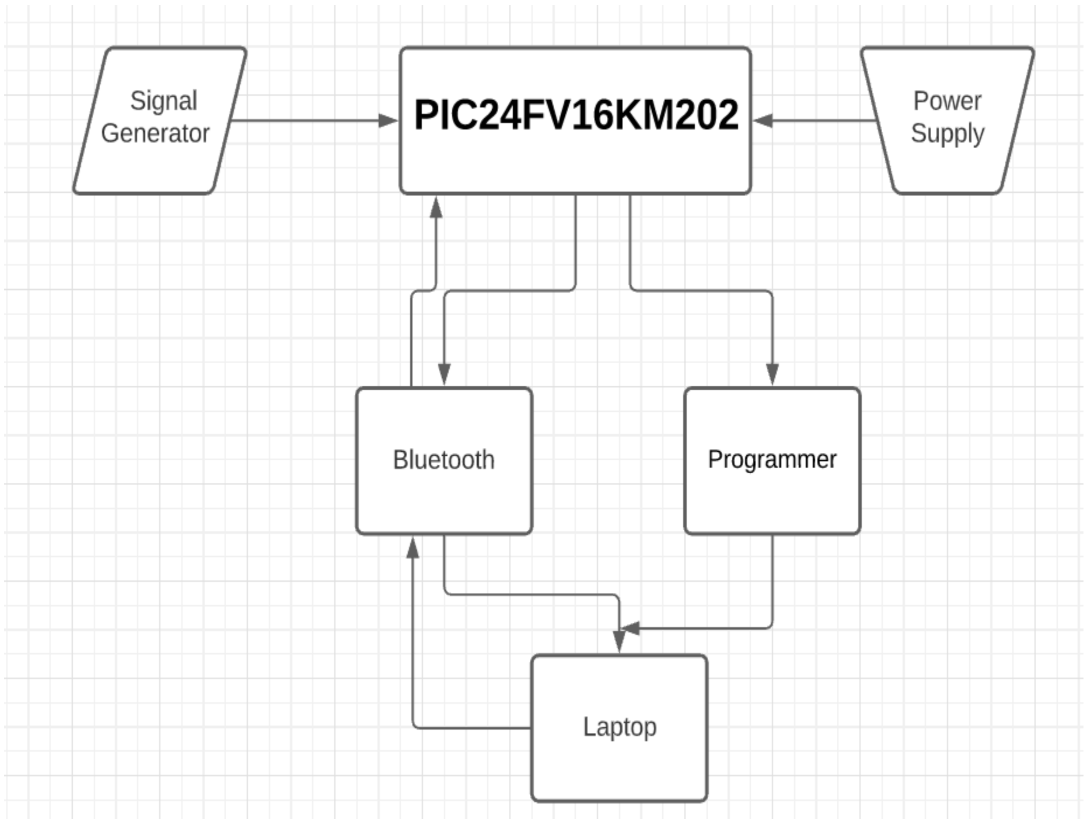
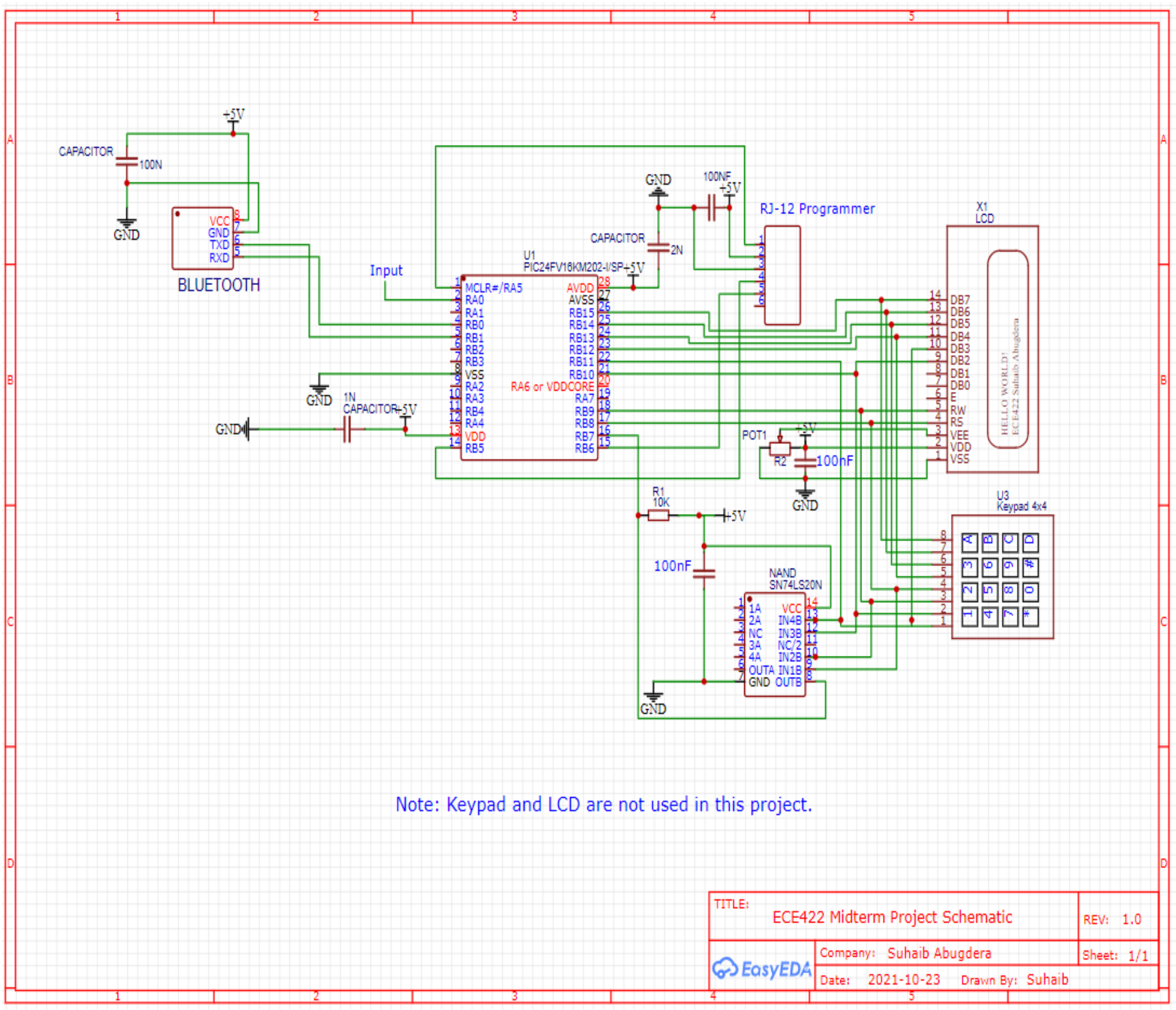


Figure (3) System Block Diagram

The system block diagram in figure (3) above shows the component hardware implemented in this project. Each of the six components in the diagram has the functionality of the microcontroller. The project collects signals from the function generator through the input on PIC24F pin 2 and plots it on GUI which is built in the laptop by sending the data via Bluetooth.

Starting with the power supply is connected to the breadboard and to the terminal to provide 5v and 3.3v to the chip and to download the code from PIC C to the chip. For Bluetooth module, is operating by setting the Bluetooth device on the computer to communicate with the chip using graphic user interface along with computer and Visual Studio to receive the information via Bluetooth. The hardware mentioned is in the system schematic in figure (4) below.

System Schematic of Hardware



Figure(4) System Schematic of Hardware.

The schematic in figure (4) above is attached with a pdf file within the project to provide a clear image with zooming in and out to make the schematic clear.

As seen from the schematic, the PIC24FV16KM202 chip is connected to 5v and ground provided by the power supply; the pins from 1 to 5 in the programmer are also connected to the chip to pin RA5, RB5, RB6, and VDD to function as input and output information exchange like downloading the compiled code into the chip. Also, the Bluetooth module is connected to RB0 and RB1 of the PIC24F and to the power supply 5v and the ground. The keypad 4X4 and LCD are not used in this project, but they have been used in previous labs.

The Bluetooth module is connected to RB0 and RB1 as mentioned are an important part for receiving and transmitting data as specified in the datasheet that those pins work with TXD and RXD. The Bluetooth module is a very important device in this project; it allows the user to communicate with the terminals in the laptops or computers and visual studio GUI after setting them in the computer.

The implementation hardware of the system schematic in figure (4) above is shown below in figure (5) to provide an understanding of the physical connection onto the breadboard.

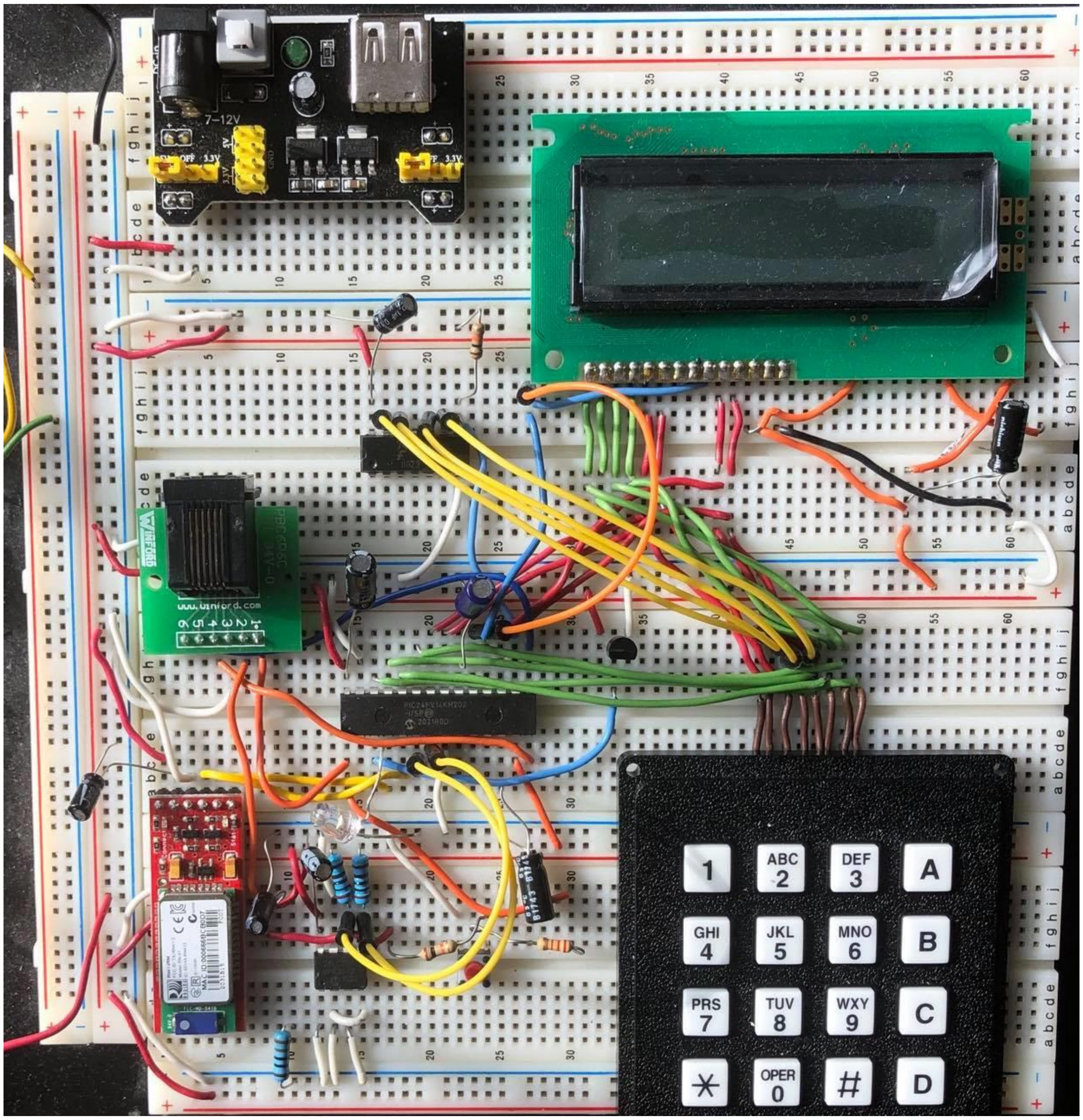
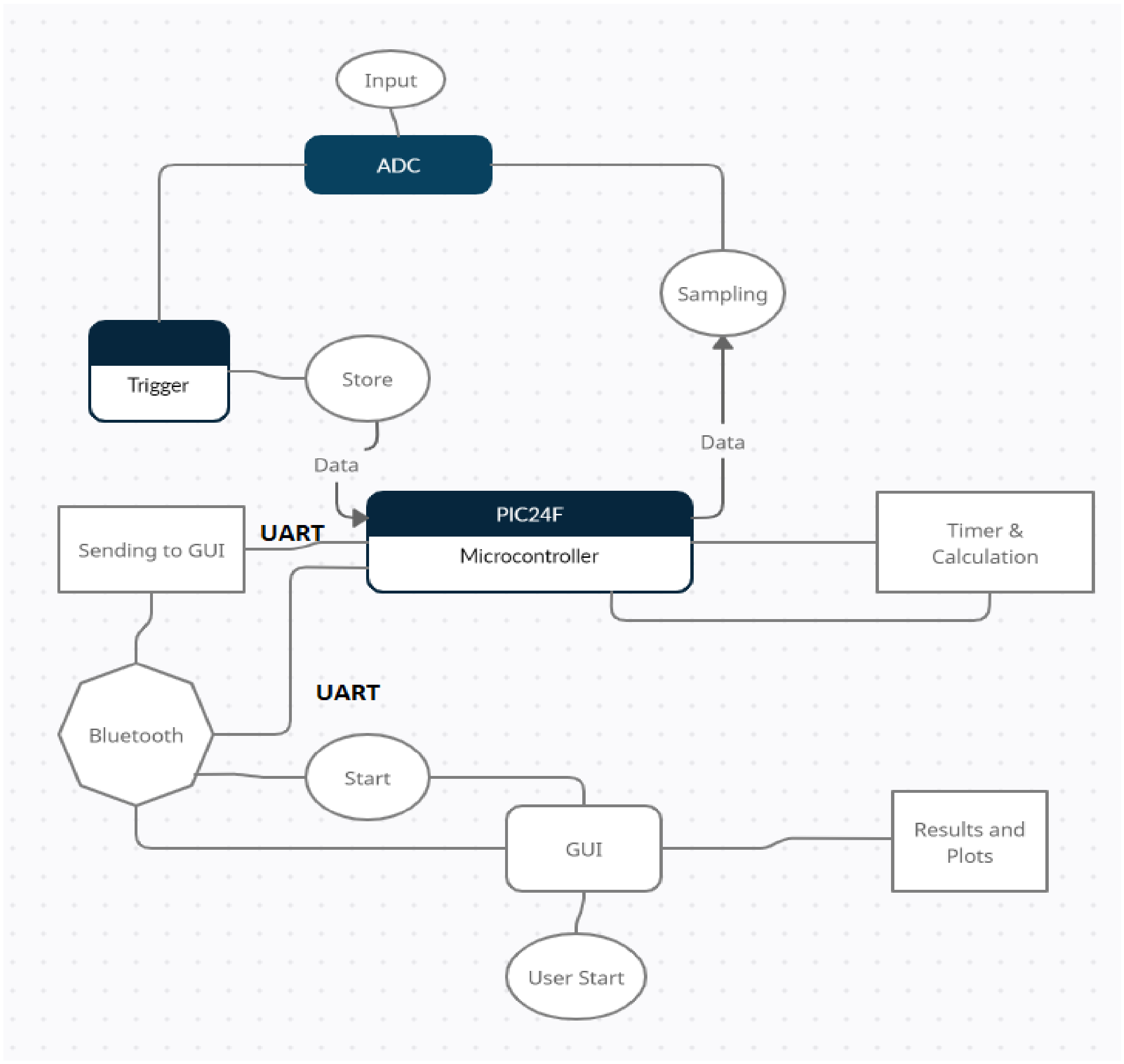


Figure (5) System Hardware Implementation

Software Components

System Flowchart of the Software:



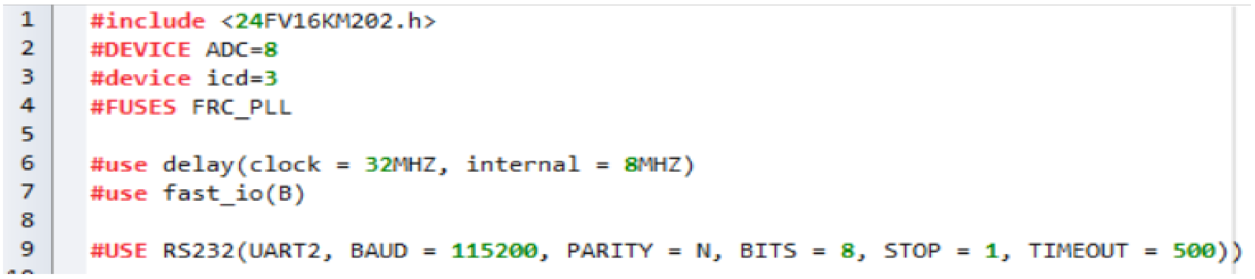
Figure(6) System Flowchart.

The flowchart in figure (6) above is simplified for one input used as a general function of the system. It begins with an input of the signal generator to the PIC24F in the ADC pin with storing and triggering when needed. Inside the chip, a UART's used to transmit and receive serial data. The sampling data from ADC will go through timer calculation and normalization to suit the right frequency and send the data to GUI via Bluetooth using communication protocol and these data will be controlled by GUI to plot the result and send back the coefficients and control signals with FDA tools' coefficients to receive other data.

Software implementation

For the software implementation code, it is cumbersome, and it will not be clear if the code is typed here, but the code will be provided in screenshots or snips for the specified design below; also, some code will be added in the appendices at the end of the report.

Including and defining devices:

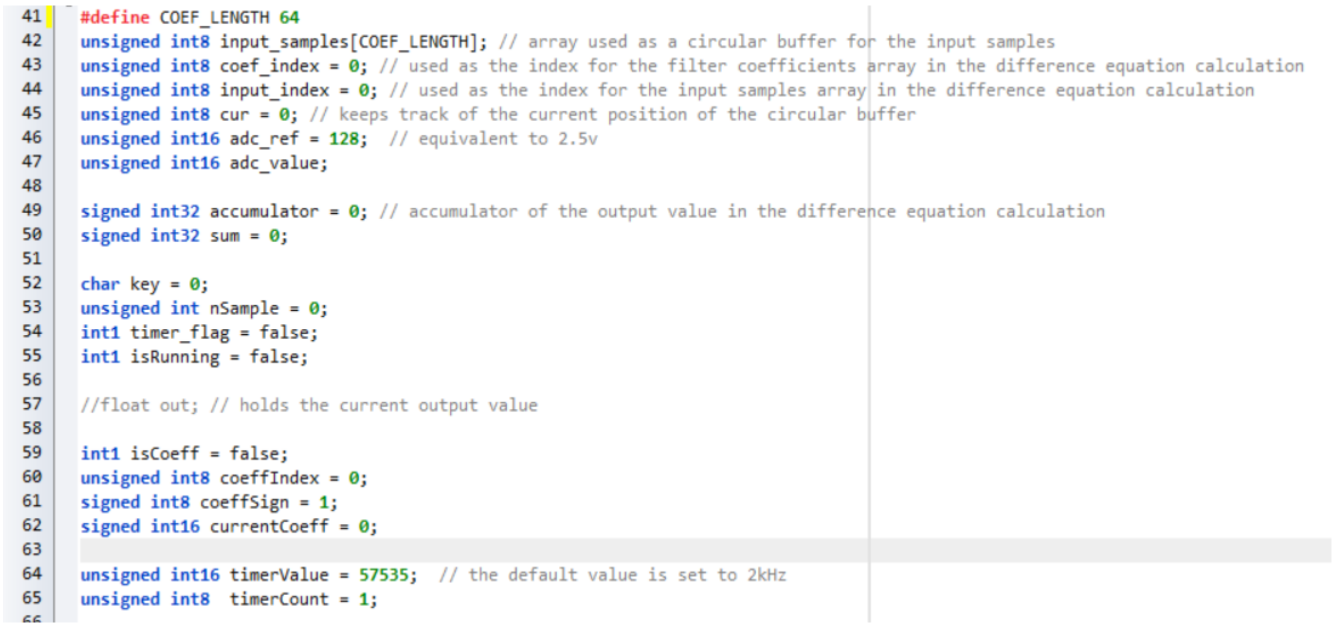


```
#include <24FV16KM202.h>
#DEVICE ADC=8
#device icd=3
#FUSES FRC_PLL

#use delay(clock = 32MHZ, internal = 8MHZ)
#use fast_io(B)

#USE RS232(UART2, BAUD = 115200, PARITY = N, BITS = 8, STOP = 1, TIMEOUT = 500))
```

Figure(7) Defining devices

Figure (7) above shows including the microcontroller and devices calls in main(); for ADC device is set to pin 2 which is 8 bits and there is icd, which is a great method for debugging without interrupt. The delay clock is set to 32MHZ with an internal 8MHZ as in the datasheet. Also, use RS232 for UART and define the baud, parity, bits, and timeout.

Initializations Design and Code with Explanations:

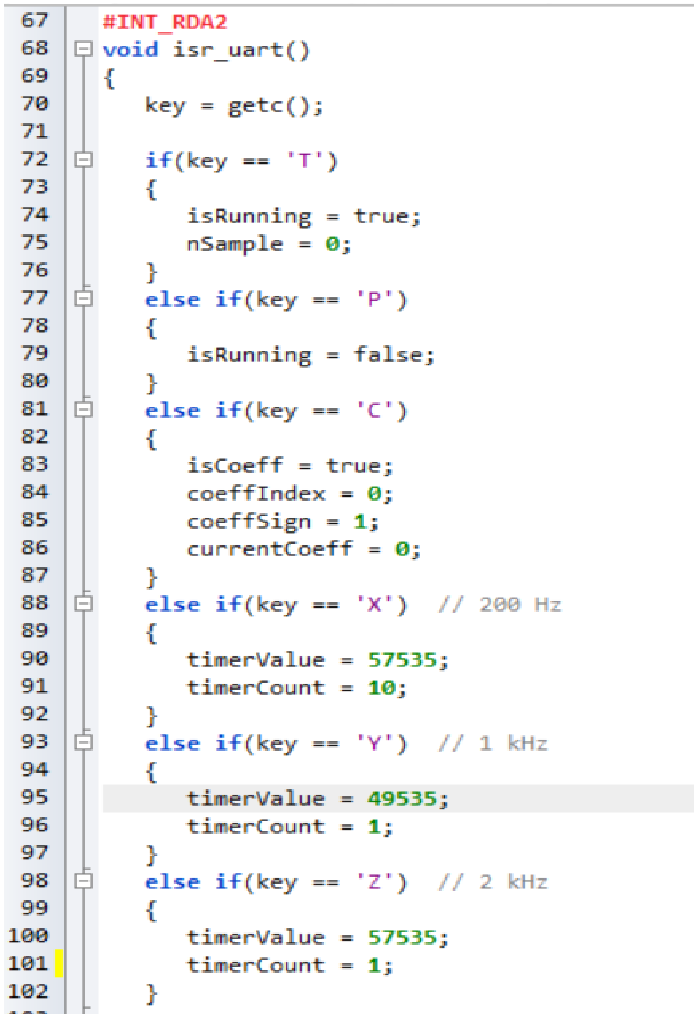


```
#define COEF_LENGTH 64
unsigned int8 input_samples[COEF_LENGTH]; // array used as a circular buffer for the input samples
unsigned int8 coef_index = 0; // used as the index for the filter coefficients array in the difference equation calculation
unsigned int8 input_index = 0; // used as the index for the input samples array in the difference equation calculation
unsigned int8 cur = 0; // keeps track of the current position of the circular buffer
unsigned int16 adc_ref = 128;  // equivalent to 2.5v
unsigned int16 adc_value;

signed int32 accumulator = 0; // accumulator of the output value in the difference equation calculation
signed int32 sum = 0;

char key = 0;
unsigned int nSample = 0;
int1 timer_flag = false;
int1 isRunning = false;

//float out; // holds the current output value

int1 isCoeff = false;
unsigned int8 coeffIndex = 0;
signed int8 coeffSign = 1;
signed int16 currentCoeff = 0;

unsigned int16 timerValue = 57535;  // the default value is set to 2kHz
unsigned int8  timerCount = 1;
```

Figure (8) System Initialization Code.

Figure (8) above shows the initializations for configuration. Setting 64 tabs for the length of coefficients. Initializing array to be used as a circular buffer for the input samples and indexes for the difference equation calculation and ADC reference 16 bits to 128. Also, a 32 bits accumulator of the output value in the difference equation calculation. The timer is initialized to set the flags. Setting the sampling data and the timer value and timer flag. The float out has some variables to be initialized and is used to hold the current output value in lines 59 to 65 with the default value set to 2kHz.

For isr_uart() function below in figure (9) below is important for determining the timer values. UART is responsible to receive and transmit data from PIC24F, Bluetooth, and GUI.



```
#INT_RDA2
void isr_uart()
{
   key = getc();

   if(key == 'T')
   {
      isRunning = true;
      nSample = 0;
   }
   else if(key == 'P')
   {
      isRunning = false;
   }
   else if(key == 'C')
   {
      isCoeff = true;
      coeffIndex = 0;
      coeffSign = 1;
      currentCoeff = 0;
   }
   else if(key == 'X')  // 200 Hz
   {
      timerValue = 57535;
      timerCount = 10;
   }
   else if(key == 'Y')  // 1 kHz
   {
      timerValue = 49535;
      timerCount = 1;
   }
   else if(key == 'Z')  // 2 kHz
   {
      timerValue = 57535;
      timerCount = 1;
   }
```

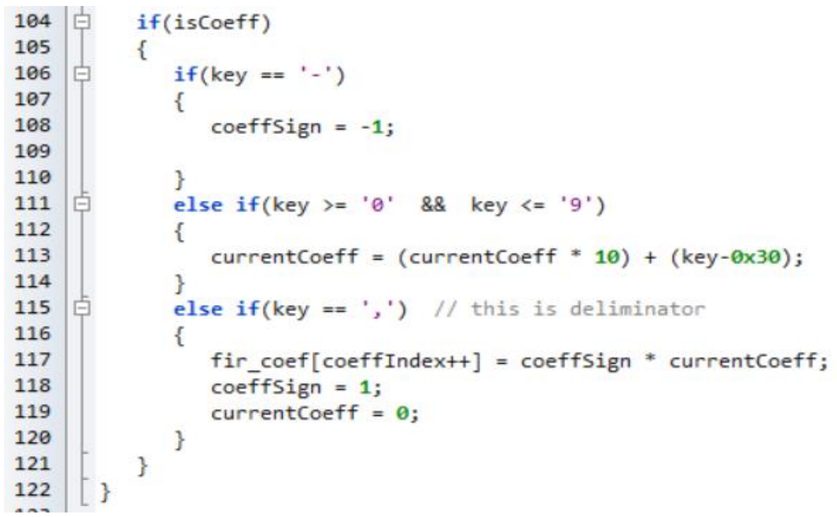
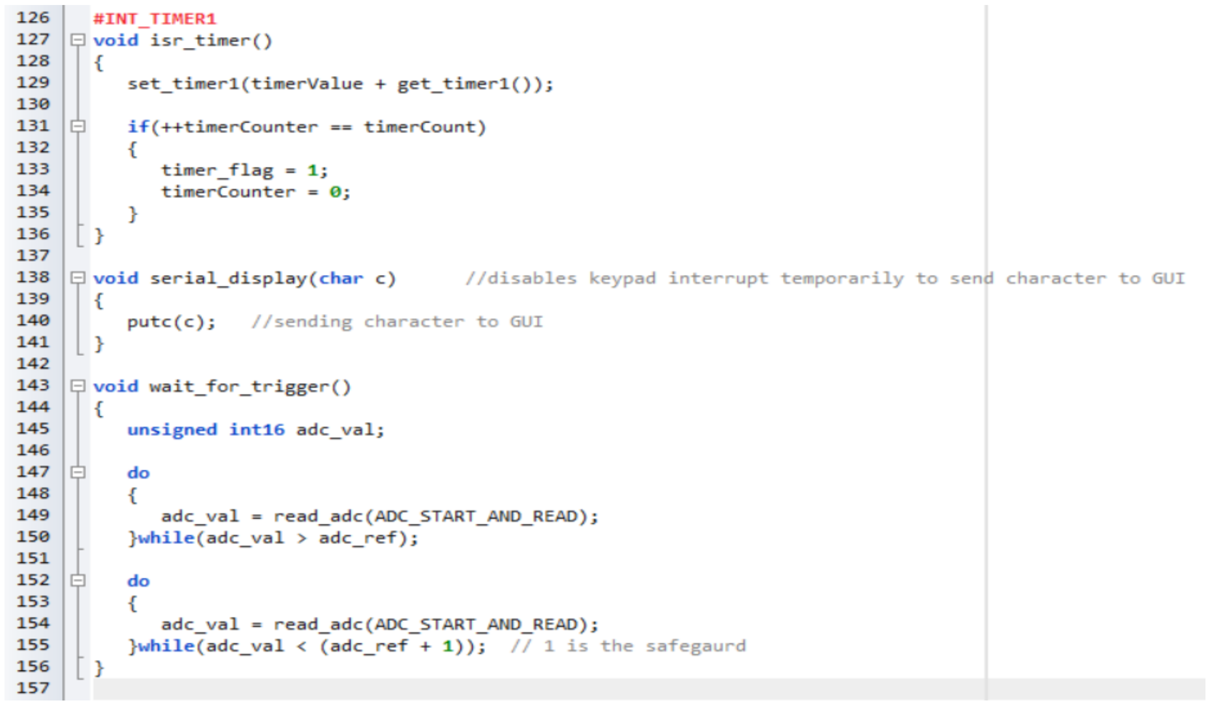

```
    if(isCoeff)
    {
       if(key == '-')
       {
          coeffSign = -1;

       }
       else if(key >= '0'  &&  key <= '9')
       {
          currentCoeff = (currentCoeff * 10) + (key-0x30);
       }
       else if(key == ',')  // this is deliminator
       {
          fir_coef[coeffIndex++] = coeffSign * currentCoeff;
          coeffSign = 1;
          currentCoeff = 0;
       }
    }
 }
```

Figure (9) function isr_uart code sample

The function is important to communicate and signal if it is running or not using to keys "P" and "T" for stop sampling and start sampling that communicates with C# GUI for the click bottoms start and stop sampling. Also, it determines the timer values in the switch statement. Other signals X, Y, and Z to set the frequency to 200hz, 1kHz, and 2kHz. The coefficients that go back and forth from GUI to PIC, need to be converted to ASCII for digits 0 to 9, comma, and minus sign.

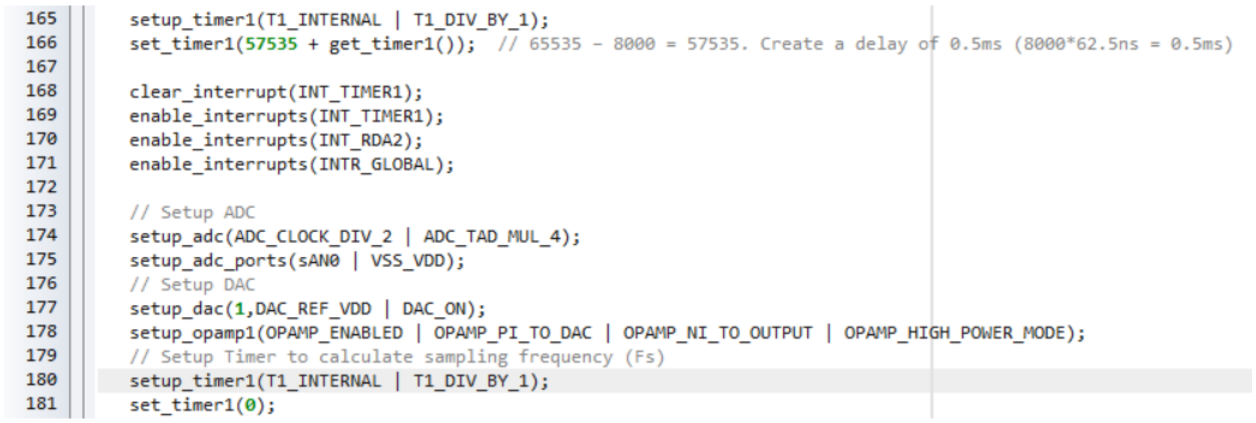
The timer interrupt, serial display, and trigger using ADC value are implemented in the code as in figure (10) below. The three void functions are timer to count the timer and set the flag, serial display that sends control signal "C" to GUI to start sending characters and trigger based on the ADC value that sets its reference and compares it with its value so that it can read the ADC.



```
#INT_TIMER1
void isr_timer()
{
   set_timer1(timerValue + get_timer1());

   if(++timerCounter == timerCount)
   {
      timer_flag = 1;
      timerCounter = 0;
   }
}

void serial_display(char c)      //disables keypad interrupt temporarily to send character to GUI
{
   putc(c);   //sending character to GUI
}

void wait_for_trigger()
{
   unsigned int16 adc_val;

   do
   {
      adc_val = read_adc(ADC_START_AND_READ);
   }while(adc_val > adc_ref);

   do
   {
      adc_val = read_adc(ADC_START_AND_READ);
   }while(adc_val < (adc_ref + 1));  // 1 is the safegaurd
}
```

Figure (10) Timer interrupt, serial display, and trigger

For ADC setup and interrupt, see figure (11) below for ADC setup and interrupt code sample.

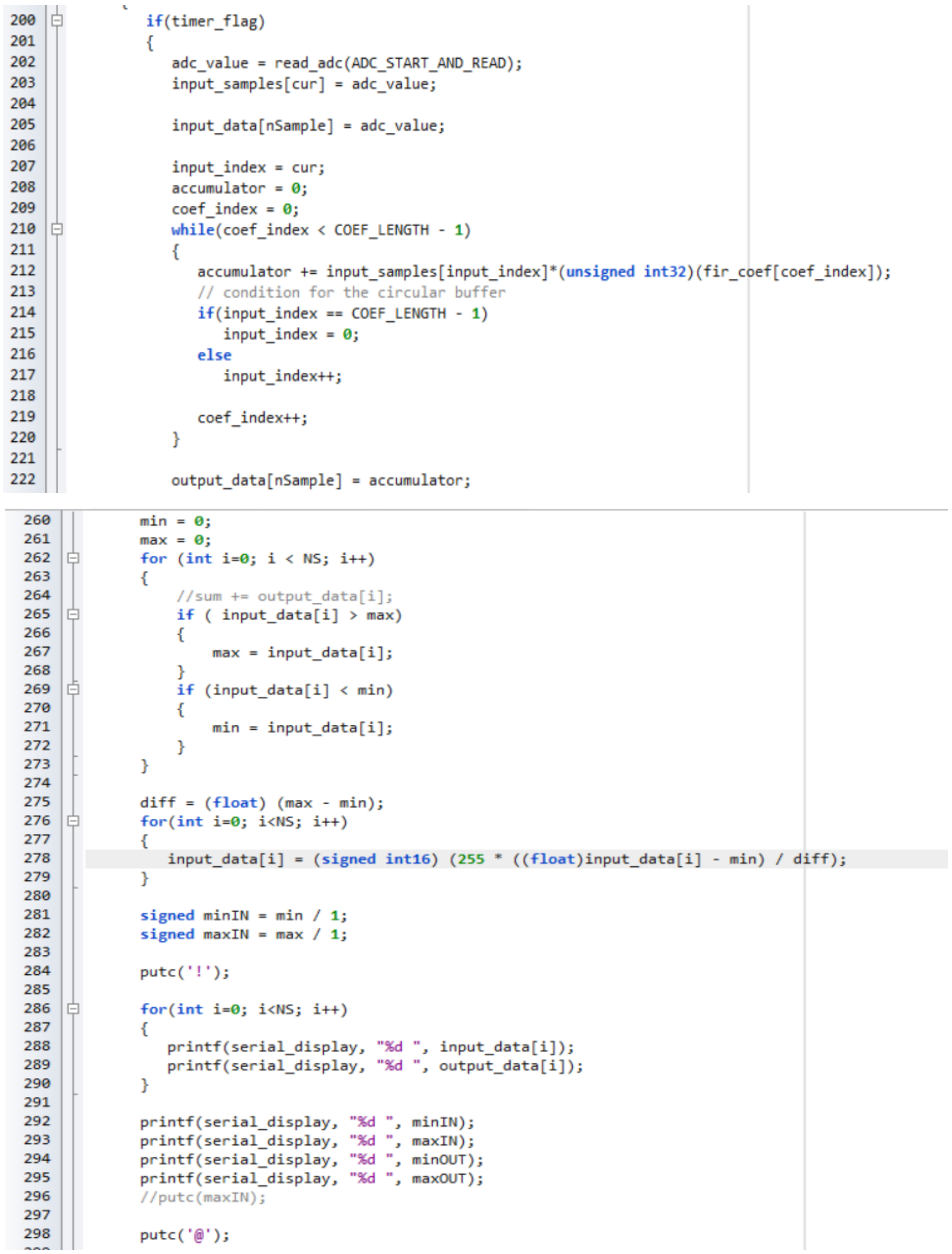


```
165    setup_timer1(T1_INTERNAL | T1_DIV_BY_1);
166    set_timer1(57535 + get_timer1());  // 65535 - 8000 = 57535. Create a delay of 0.5ms (8000*62.5ns = 0.5ms)
167
168    clear_interrupt(INT_TIMER1);
169    enable_interrupts(INT_TIMER1);
170    enable_interrupts(INT_RDA2);
171    enable_interrupts(INTR_GLOBAL);
172
173    // Setup ADC
174    setup_adc(ADC_CLOCK_DIV_2 | ADC_TAD_MUL_4);
175    setup_adc_ports(sAN0 | VSS_VDD);
176    // Setup DAC
177    setup_dac(1,DAC_REF_VDD | DAC_ON);
178    setup_opamp1(OPAMP_ENABLED | OPAMP_PI_TO_DAC | OPAMP_NI_TO_OUTPUT | OPAMP_HIGH_POWER_MODE);
179    // Setup Timer to calculate sampling frequency (Fs)
180    setup_timer1(T1_INTERNAL | T1_DIV_BY_1);
181    set_timer1(0);
```

Figure (11) ADC setup and interrupt code sample

ADC setup is important to allow to take samples of the input data. The ADC is set to pin 2 “RA0”; see the datasheet for PIC24F, page 6. The timer and interrupt are clearly set in lines 165 to 171. The timer is configured to function as a time counter. After running on the counter incrementation, the values are stored into registers and then the timer resets and the interrupt enables to run a specific function; then clearing the timer and set to run.

Normalization to digitize the 150 samples in the code in figure (12) below. It is simply a loop to check if the data is less than 150, it reads and stores it in the data array, and the control signal "!" and "@" in lines 284 and 298 are the start and end sampling to communicate with C#.



```
200         if(timer_flag)
201         {
202            adc_value = read_adc(ADC_START_AND_READ);
203            input_samples[cur] = adc_value;
204
205            input_data[nSample] = adc_value;
206
207            input_index = cur;
208            accumulator = 0;
209            coef_index = 0;
210            while(coef_index < COEF_LENGTH - 1)
211            {
212               accumulator += input_samples[input_index]*(unsigned int32)(fir_coef[coef_index]);
213               // condition for the circular buffer
214               if(input_index == COEF_LENGTH - 1)
215                  input_index = 0;
216               else
217                  input_index++;
218
219               coef_index++;
220            }
221
222            output_data[nSample] = accumulator;
```

```
260      min = 0;
261      max = 0;
262      for (int i=0; i < NS; i++)
263      {
264         //sum += output_data[i];
265         if ( input_data[i] > max)
266         {
267            max = input_data[i];
268         }
269         if (input_data[i] < min)
270         {
271            min = input_data[i];
272         }
273      }
274
275      diff = (float) (max - min);
276      for(int i=0; i<NS; i++)
277      {
278         input_data[i] = (signed int16) (255 * ((float)input_data[i] - min) / diff);
279      }
280
281      signed minIN = min / 1;
282      signed maxIN = max / 1;
283
284      putc('!');
285
286      for(int i=0; i<NS; i++)
287      {
288         printf(serial_display, "%d ", input_data[i]);
289         printf(serial_display, "%d ", output_data[i]);
290      }
291
292      printf(serial_display, "%d ", minIN);
293      printf(serial_display, "%d ", maxIN);
294      printf(serial_display, "%d ", minOUT);
295      printf(serial_display, "%d ", maxOUT);
296      //putc(maxIN);
297
298      putc('@');
```

Figure (12) Normalization

The code above in figure (12) is used for normalization. Using while loop in lines 210 to 222 to set and collect data for accumulator and find the maximum and minimum of the input data in lines 260 to 273 and then find the difference in line 275 to be used in input equation in line 278. After the control signal "!" to start sampling, the print functions are set in for loop to print input, output data, and maximum inputs and outputs which will be explained in the test and results that show how they are displayed in GUI.

Serial Communication Design:

Serial communication is a bidirectional system that has a direction from Bluetooth to computer to allow the control via Bluetooth. The Bluetooth module is connected to the chip and to the computer terminal to allow receiving or transmitting data. Using UART communication functions, which allows the user to process data.

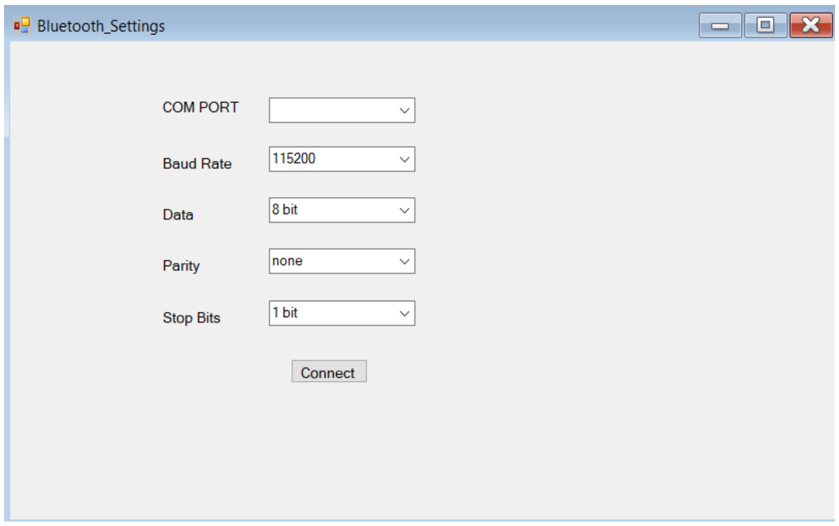
When the chip PIC24F receives the data, it goes to an interrupt to receive the coming data and allow it to be processed in the loop.

The interrupt reads the byes that are sent to the receiver. If the buffer data aren't the input values, the receiving data will behold into an array. If the key is received, the buffer is increased once at a time and it will be assigned to a register to be processed, and then a null character will be placed at the end of the data array as an indicator for the end of the array.

So, the while loop means to run this block of code if data is received; and will convert from a string to an integer and clear the flag.

Bluetooth Setting Design for GUI C#:

The Bluetooth design is designed on a graphic user interface (GUI) program using C# to establish a serial connection between a PC and a microcontroller using a Bluetooth module; see figure (13) for the Bluetooth settings, which used to communicate or connect with the computer.


Bluetooth_Settings

COM PORT	
Baud Rate	115200
Data	8 bit
Parity	none
Stop Bits	1 bit

Figure (13) Bluetooth Settings.

The design of the above settings for Bluetooth is given in the tutorial, which is about serial communication. The design is simple and easy to build. The Combo Boxes are for the description to run the module, for example, COM5 is the terminal to connect to the laptop port that the module is connected to. The other Combo Boxes are given specifically for this project. The code is implemented automatically by C# language on the Visual Studio platform. The design is made by C# automatically by clicking on the components needed for the module. Components like Combo Box, labels, and buttons. Labels are to name the operation for the Compo Boxes; the Combo Boxes are the parameters for the connection description and the button is pressed to connect the port. The code is modified for this project that doesn't require a lot of code.

C# GUI and Plot Design and Code:

One of the requirements for this project is the GUI design and plot, which is designed to interface the basic level of the project with the GUI. Using Visual Studio platform to design the GUI using C# language. The given tutorial for serial communication is useful to establish this design. See the final shape of the GUI design in the appendix (B). The purpose of this part is to display the results and plots of LPF, BPF, and DFT. See the flowchart for the GUI in figure (14) below.

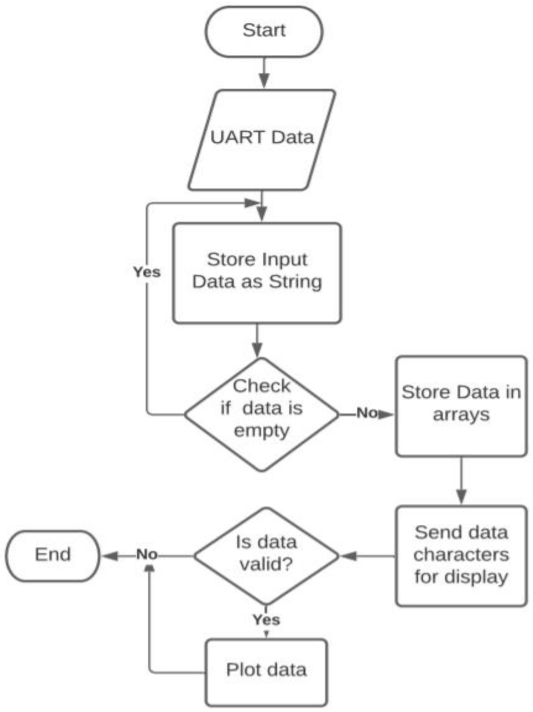


Figure (14) GUI Flowchart.

When pressing start or run the compiler will compile the data, which is stored as a string and comparing byte. The comparing happens with the null to check if the data is empty or not. The valid data will be extracted one character at a time using arrays to send them as one package. See the event chart in figure (15) below. The stored data will be plotted as shown in figure (16).

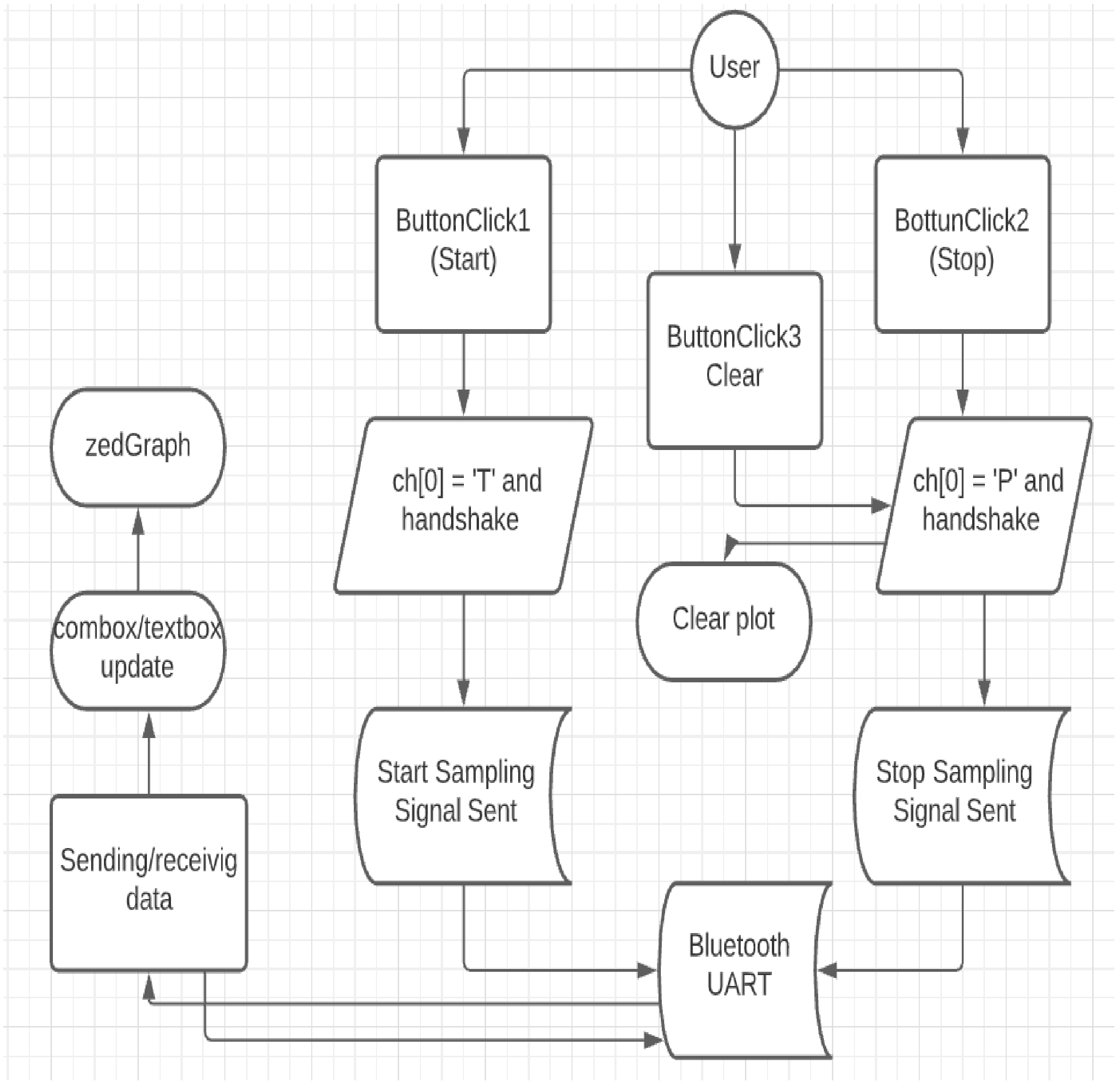


Figure (15) Event Chart

The event chart above has different events with different operations. When clicking on Button click1 after the Bluetooth is connected, the signal 'T' will be recognized and sent to PIC C for starting the sampling and handshake to empty the buffer. The button click2 is like button click1; for the button, click3 is to clear the data and plots by stopping it first and sending the clear function from zedgraph. PIC C will send the data array stored by UART via Bluetooth and update it using textbox and combo boxes that have different signals that tell which frequency is needed like frequency 100 Hz. 200 Hz and so on. The updated data will be sent to zedgraph for plotting.

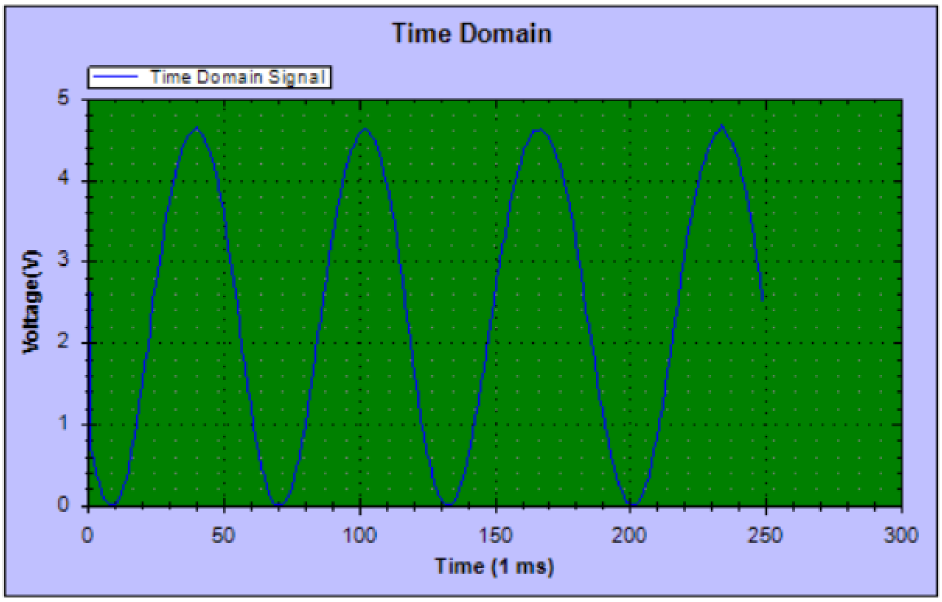


Figure (16) GUI real-time plotting.

Figure (16) above shows a frequency plotted using an even button click in GUI. It is processed as in the even chart in figure (15) above.

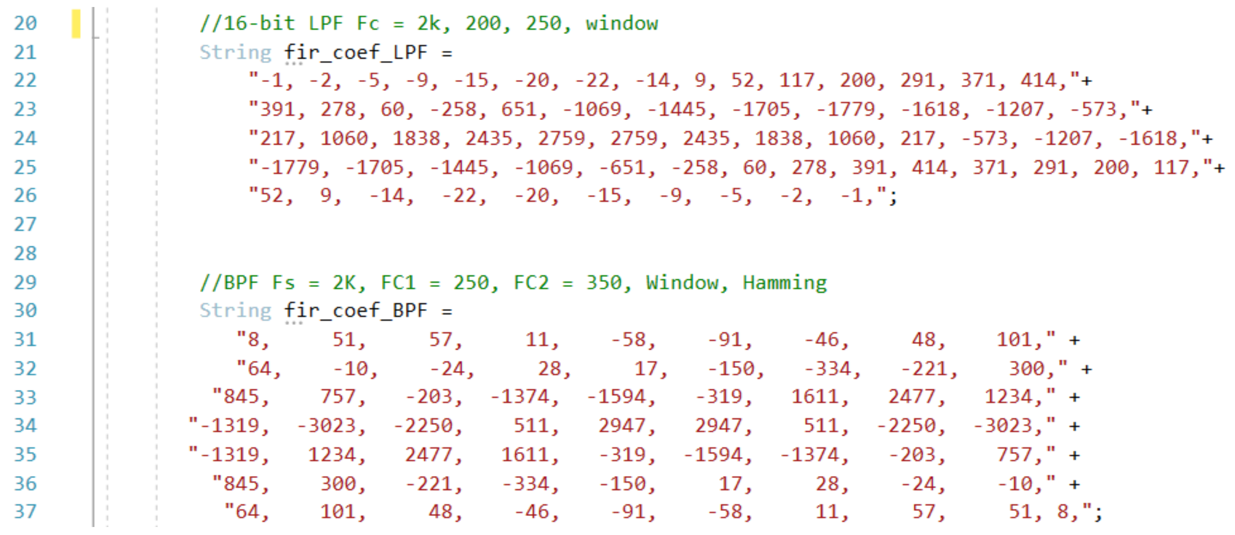
The code starts with designing the GUI from the tutorial given in lab 10 and because of its simplicity it can be done without the tutorial. All components are chosen, characters are extracted, button timer delay, code for Receiving Data and Updating Rich Text Box, text exchanged, and plot settings.

The sampling frequency code is done in the Combobox value changed that shows the sampling process with signals and measurement labels for the graph. It is a function called Send_Coeff_Click. It is responsible to give a signal from C# GUI to PIC C to start the process of sending the coefficient from GUI to the chip. See the code below in figure (17) for the sampling frequency process.

```
private void Send_Coeff_Click(object sender, EventArgs e)
{
    char[] ch = new char[1];

    switch ((string)comboBox3.Text) //determines current sampling freq. setting
    {
        case "LPF":
            ch[0] = 'C';
            _setting._serial.Write(ch, 0, 1);
            System.Threading.Thread.Sleep(1);

                    _setting._serial.Write(fir_coef_LPF);

            break;
        case "BPF":
            ch[0] = 'C';
            _setting._serial.Write(ch, 0, 1);
            System.Threading.Thread.Sleep(1);
            _setting._serial.Write(fir_coef_BPF);
            break;
        default:
            break;
           }
    switch ((string)comboBox2.Text) //determines current sampling freq.setting
    {
        case "10 Hz":
            ch[0] = 'X';
            _setting._serial.Write(ch, 0, 1);
            System.Threading.Thread.Sleep(1);
            break;

        case "50 Hz":
            ch[0] = 'Y';
            _setting._serial.Write(ch, 0, 1);
            System.Threading.Thread.Sleep(1);
            break;

        case "100Hz":
            ch[0] = 'Z';
            _setting._serial.Write(ch, 0, 1);
            System.Threading.Thread.Sleep(1);
            break;

        default:
            break;
    }
       }
```

Figure (17) Sending Coefficient function in GUI.

In figure (17) above, it shows a code that specifies which plot will be done by sending their coefficients to PIC C. the case statement determines the sampling frequency setting, and can someone choose which plot of LPF, BPF, or DFT is needed to be displayed or plotted. Also, other options are available like frequency of 10 Hz, 50 Hz, or 100 Hz. See figure (18) below for the coefficients that are sent to PIC C.

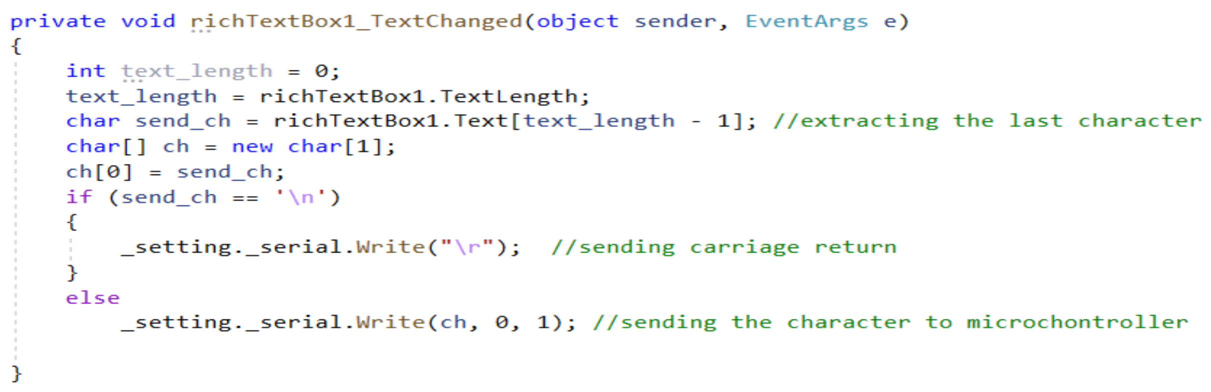


```
        //16-bit LPF Fc = 2k, 200, 250, window
        String fir_coef_LPF =
            "-1, -2, -5, -9, -15, -20, -22, -14, 9, 52, 117, 200, 291, 371, 414,"+
            "391, 278, 60, -258, 651, -1069, -1445, -1705, -1779, -1618, -1207, -573,"+
            "217, 1060, 1838, 2435, 2759, 2759, 2435, 1838, 1060, 217, -573, -1207, -1618,"+
            "-1779, -1705, -1445, -1069, -651, -258, 60, 278, 391, 414, 371, 291, 200, 117,"+
            "52,  9,  -14,  -22,  -20,  -15,  -9,  -5,  -2,  -1,";


        //BPF Fs = 2K, FC1 = 250, FC2 = 350, Window, Hamming
        String fir_coef_BPF =
           "8,      51,      57,      11,     -58,     -91,     -46,      48,     101," +
           "64,     -10,     -24,      28,      17,    -150,    -334,    -221,     300," +
         "845,     757,    -203,   -1374,   -1594,    -319,    1611,    2477,    1234," +
       "-1319,   -3023,   -2250,     511,    2947,    2947,     511,   -2250,   -3023," +
       "-1319,    1234,    2477,    1611,    -319,   -1594,   -1374,    -203,     757," +
         "845,     300,    -221,    -334,    -150,      17,      28,     -24,     -10," +
          "64,     101,      48,     -46,     -91,     -58,      11,      57,      51, 8,";
```

Figure (18) FDA tools coefficients code in C# GUI.

In figure (18) above, it shows the coefficients that were extracted from MATLAB for a cutoff frequency of 2kHz and set to window hamming plots.

Extracting character using an array is an important part that communicates with PIC C and GUI by sending a carriage return and character to the microcontroller. See figure (19) for extracting character code



```
private void richTextBox1_TextChanged(object sender, EventArgs e)
{
    int text_length = 0;
    text_length = richTextBox1.TextLength;
    char send_ch = richTextBox1.Text[text_length - 1]; //extracting the last character
    char[] ch = new char[1];
    ch[0] = send_ch;
    if (send_ch == '\n')
    {
        _setting._serial.Write("\r");  //sending carriage return
    }
    else
        _setting._serial.Write(ch, 0, 1); //sending the character to microchontroller

}
```

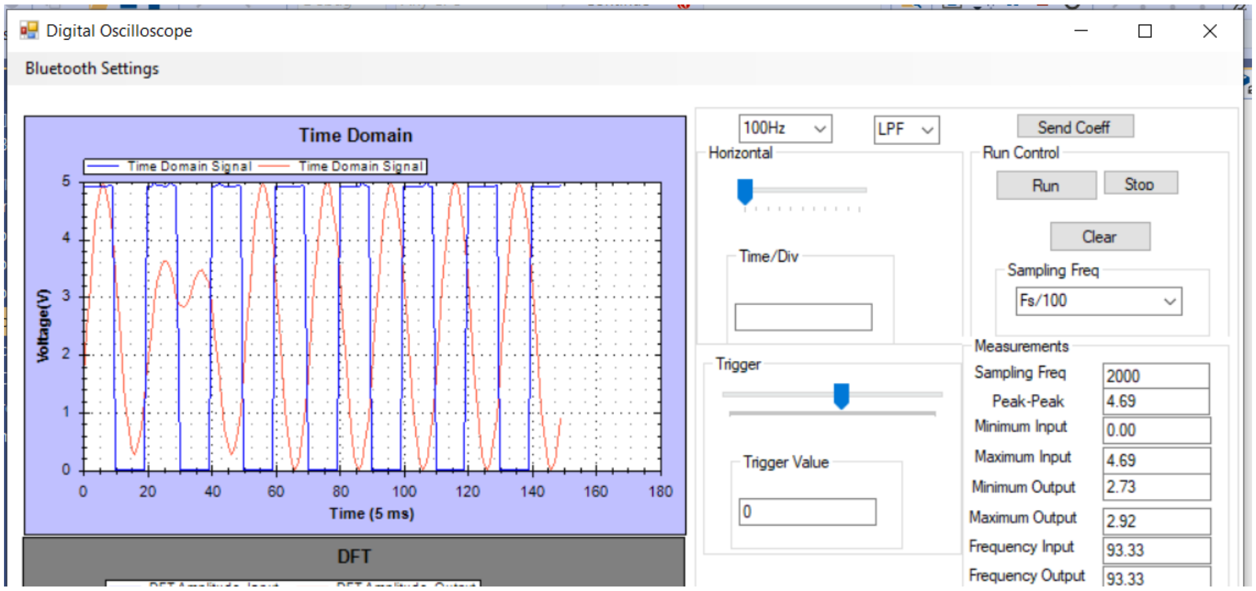
Figure (19) Extracting Characters Code.

Figure (19) above shows the code for extracting each character at a time. Using arrays to store and compare the character as ASCII. The loop is going through all characters and using null terminator; all characters don't include period ".", so, comparing them with period will be effective. Using the substring method to extract a substring from a string that begins at a specified character position and ends at the end of the string. The first character in the string is at index 0, not index 1. Comparing the data with zero to see if it is not equal to zero to add it to the plot. Also, the vertical or graph zoom is done with multiple factors as seen in the code in appendix (C). The peak to peak and maximum and minimum are done by using a function in C# called Max() and Min(). The DFT is calculated and plotted using the equations and formula provided in the class. See appendix (D) for the DFT calculation code in GUI.

Results and tests:

In this section, implementation and testing results will be shown using pictures and screenshots. The results will be discussed briefly, and further discussion will be in the analysis and discussion section below.

The project requirements of plotting LPF, BPF, and DFT with different frequencies are tested and result in correct plots and values. For LPF and BPF plots with 100 Hz, 50 Hz, and 10 Hz frequency and sampling of 2000 Hz, 1000Hz, and 200 Hz were plotted as seen in figure (20) to figure (26) below.



100Hz

LPF

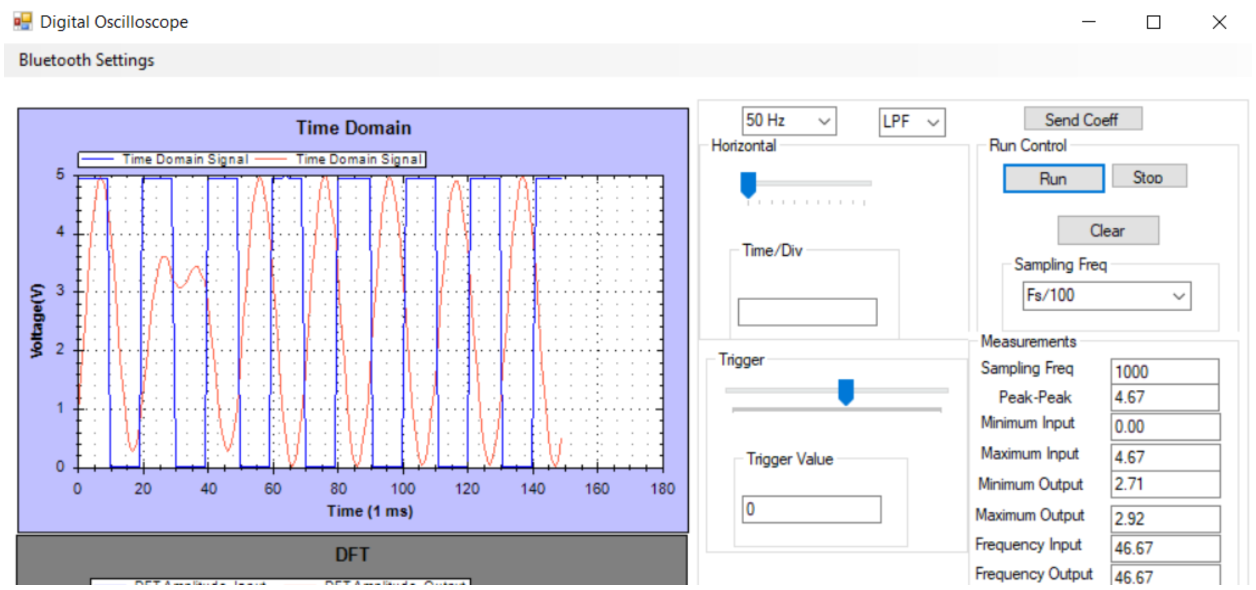
Sampling Freq: Fs/100

Trigger Value: 0

Measurements

Sampling Freq	2000
Peak-Peak	4.69
Minimum Input	0.00
Maximum Input	4.69
Minimum Output	2.73
Maximum Output	2.92
Frequency Input	93.33
Frequency Output	93.33

Figure (20) LPF with 100 Hz.

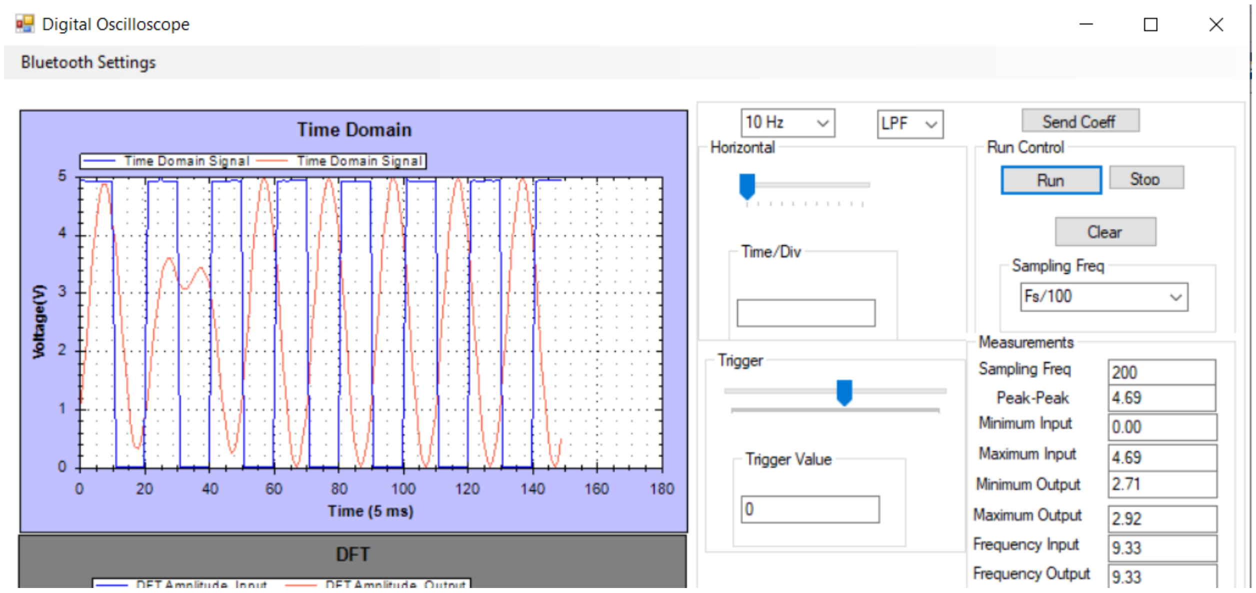


50 Hz
LPF
Sampling Freq: Fs/100
Trigger Value: 0

Measurements

Sampling Freq	1000
Peak-Peak	4.67
Minimum Input	0.00
Maximum Input	4.67
Minimum Output	2.71
Maximum Output	2.92
Frequency Input	46.67
Frequency Output	46.67

Figure (21) LPF with 50 Hz.



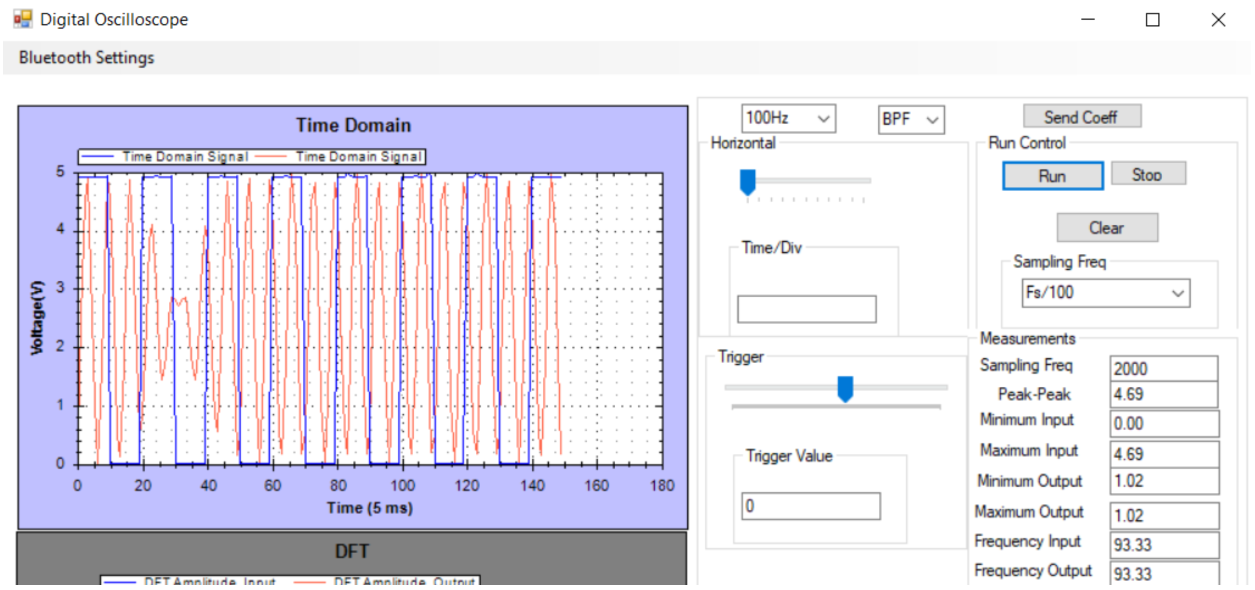
10 Hz
LPF
Sampling Freq: Fs/100
Trigger Value: 0

Measurements

Sampling Freq	200
Peak-Peak	4.69
Minimum Input	0.00
Maximum Input	4.69
Minimum Output	2.71
Maximum Output	2.92
Frequency Input	9.33
Frequency Output	9.33

Figure (22) LPF with 10 Hz.

The plots above show LPF in figure (20) with 100 Hz and 2000 Hz sampling with frequency inputs and outputs of 93.33 which should be 100 Hz. For 50 Hz LPF in figure (21) with 1000 Hz sampling and frequency inputs and outputs of 46.67 Hz which is a little far from 50 Hz. For 10 Hz LPF in figure (22) with 200 Hz sampling frequency shows frequency inputs and outputs of 9.33 Hz which is close to 10 Hz. All plots look correct, and the only issue is the dented plot on the second amplitude. This issue might be solved by fixing the normalization with an accurate timer. It is a similar issue as in BPF in figure (23) to figure (26) below.



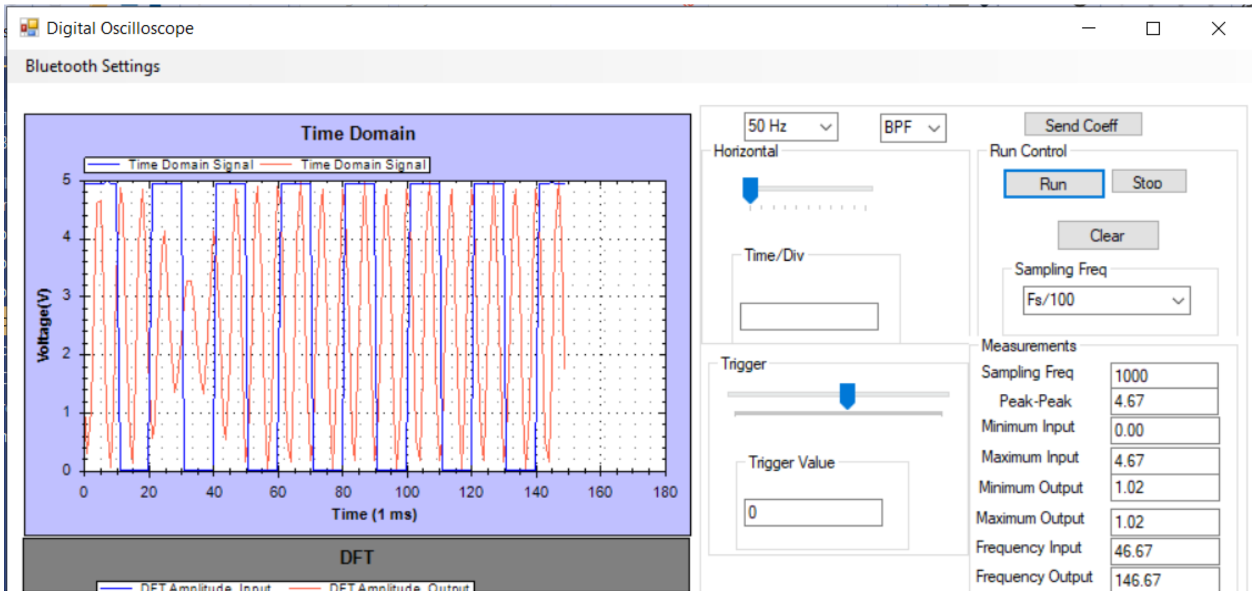
100Hz BPF

Trigger Value: 0

Sampling Freq: Fs/100

Sampling Freq	2000
Peak-Peak	4.69
Minimum Input	0.00
Maximum Input	4.69
Minimum Output	1.02
Maximum Output	1.02
Frequency Input	93.33
Frequency Output	93.33

Figure (23) BPF with 100 Hz.



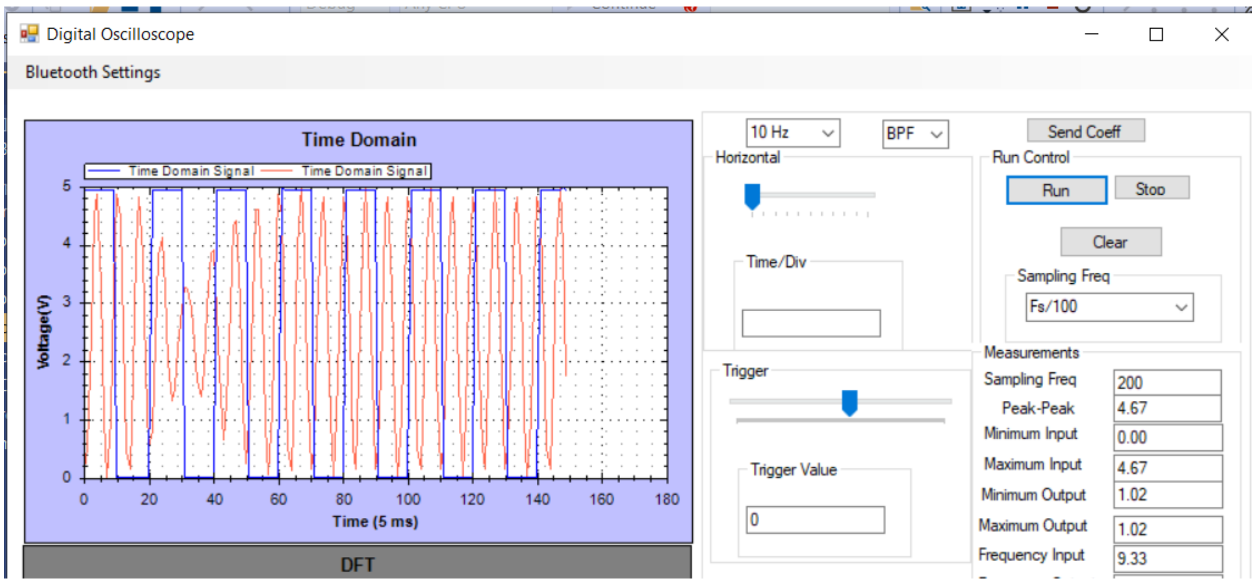
50 Hz BPF

Trigger Value: 0

Sampling Freq: Fs/100

Sampling Freq	1000
Peak-Peak	4.67
Minimum Input	0.00
Maximum Input	4.67
Minimum Output	1.02
Maximum Output	1.02
Frequency Input	46.67
Frequency Output	146.67

Figure (24) BPF with 50 Hz.



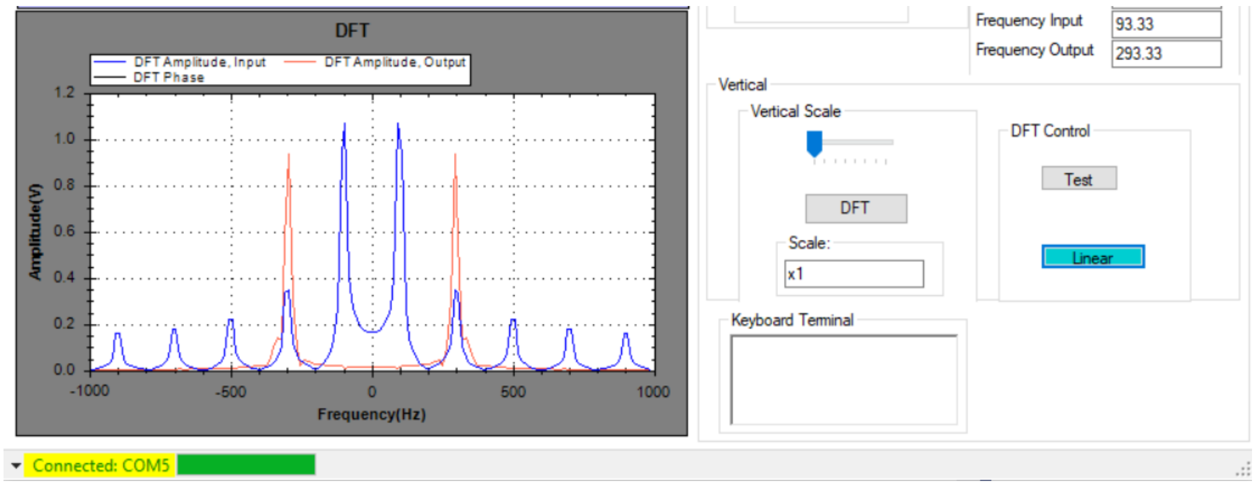
10 Hz
BPF
Fs/100
Trigger Value: 0

Sampling Freq	200
Peak-Peak	4.67
Minimum Input	0.00
Maximum Input	4.67
Minimum Output	1.02
Maximum Output	1.02
Frequency Input	9.33

Figure (26) BPF with 10 Hz.

The PIC digitizes a rectangular wave of 10, 50, 100 Hz that may be generated by a function generator. The results are a sine wave of LPF and BPF. The plots are also shown peak to peak value, which is 4.61V that is close to 5V, maximum is 4.61V, and the minimum is zero. The trigger slide bar is used to trigger the amplitude from 1V to 5V, but it seems to have an issue executing it. The vertical scale or Zoom for the plot has multiple zooming factors

The DFT plots are shown in figure (27) to figure (29) are the plots of phase in dB and amplitude of DFT in Volt with graph control 2 of zedgraph with the red plot which is DFT output and blue plot of the input.

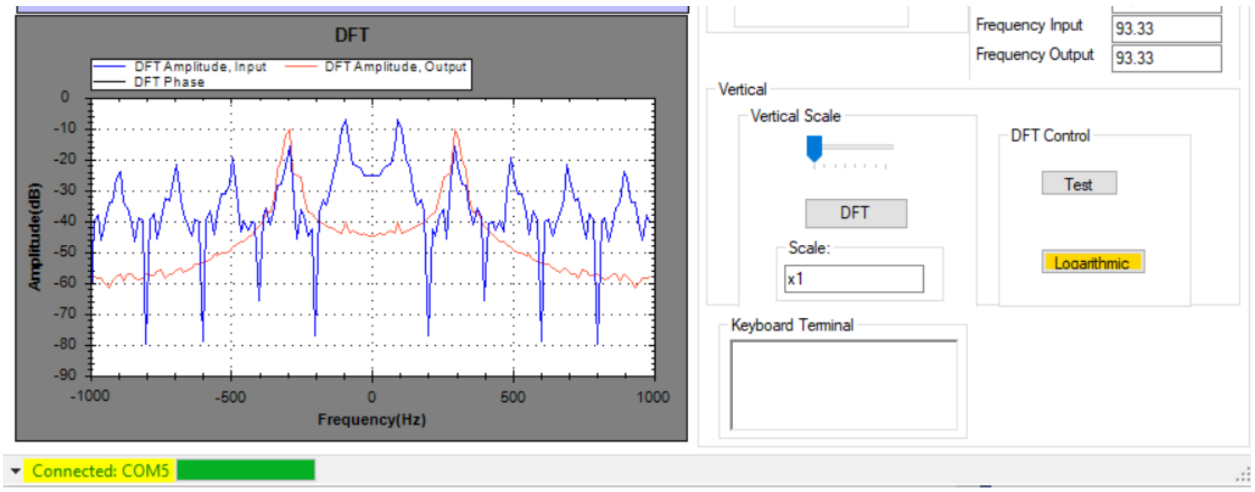


Frequency Input	93.33
Frequency Output	293.33

Scale: x1

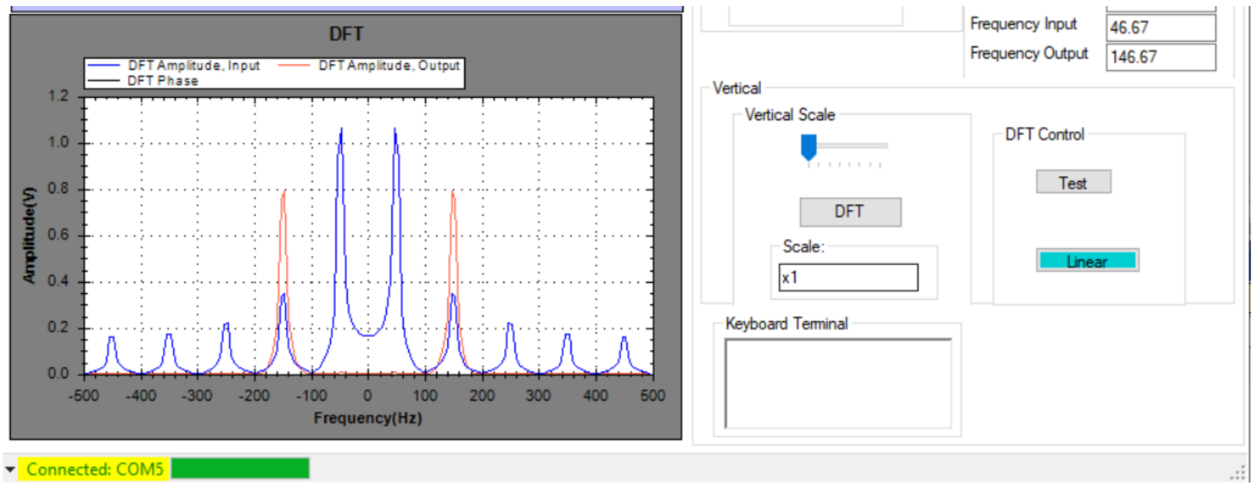
Connected: COM5

Figure (27) DFT Amplitude V for 100 Hz.



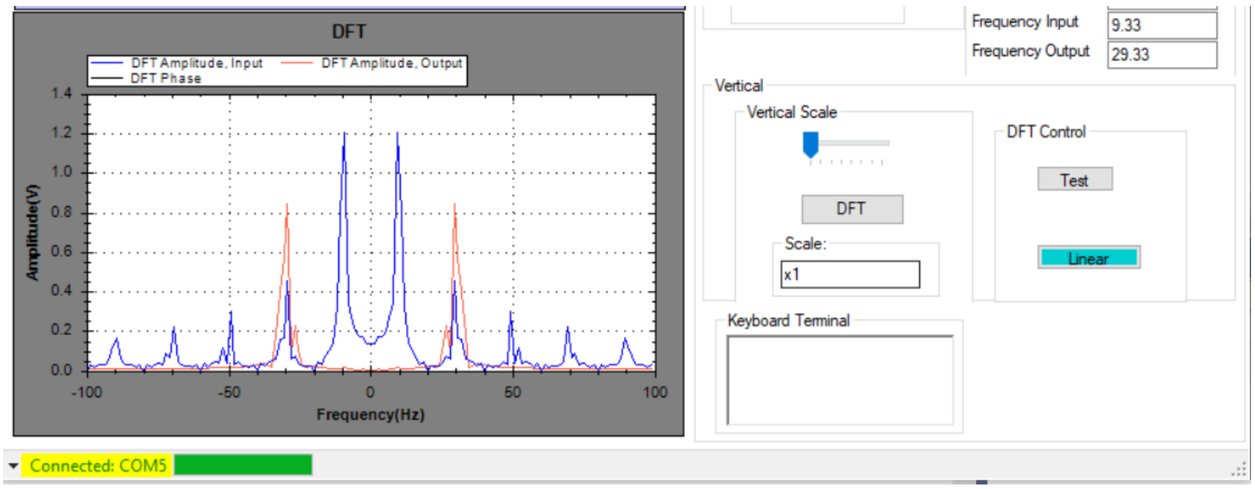
Frequency Input 93.33
Frequency Output 93.33
Scale: x1
Logarithmic
Connected: COM5

Figure (28) DFT Amplitude dB for 100 Hz.



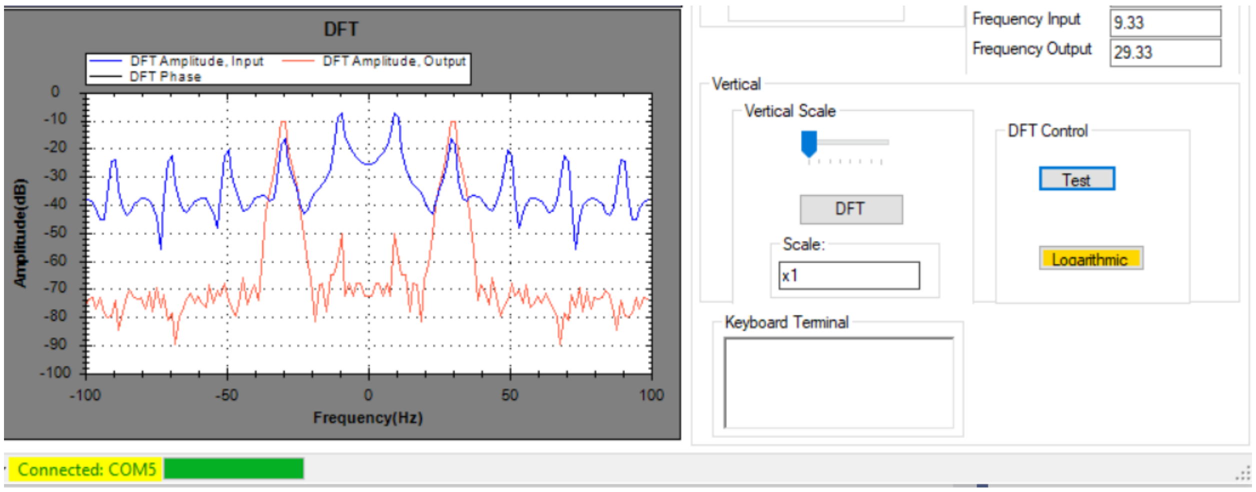
Frequency Input 46.67
Frequency Output 146.67
Scale: x1
Linear
Connected: COM5

Figure (29) DFT Amplitude V for 50 Hz.



Frequency Input 9.33
Frequency Output 29.33
Scale: x1
Linear
Connected: COM5

Figure (30) DFT Amplitude V for 10 Hz.



Frequency Input: 9.33
Frequency Output: 29.33
Scale: x1
Connected: COM5

Figure (31) DFT Amplitude dB for 10 Hz.

The plots above show the input and output frequency at the top left of the figure and plot the DFT wave in logarithmic and phase. As seen in figure (27) for DFT Amplitude V for 100 Hz; it shows an input of 93 Hz and output of 293 Hz which is supposed to be close to 100 Hz and 300 Hz. The other plots show the wave of the DFT in dB which starts at -40 dB.

The parameter of DFT was calculated in GUI using the formula provided in the class and this part was plotted correctly. See the calculation for the DFT and Function to extract DFT magnitude samples in the appendix (C).

Implementing Bootloader:

A bootloader is a program running in a microcontroller to be programmed. It receives the code externally via serial communication means Bluetooth RS232 and writes that information to the program memory of the processor. The code of the bootloader is given. See appendix (D) for boot loader code.

The code is usually downloaded into the microcontroller via a hardware device called a programmer linker. This external hardware controls the programming pins of the microcontroller to write the information into the program memory. After the program is compiled on a computer, the HEX/ICD file describing the program memory content is created. Therefore, the programmer needs to be connected to the computer and to the programming pins of the microcontroller.

First, download the bootloader code via the external programmer into the microcontroller. After the download is done, the programmer can be removed, and the system will be receiving code via Bluetooth without the programmer. The interaction will be between the computer and the microcontroller. See figure (32) for the system block diagram for the boot loader.

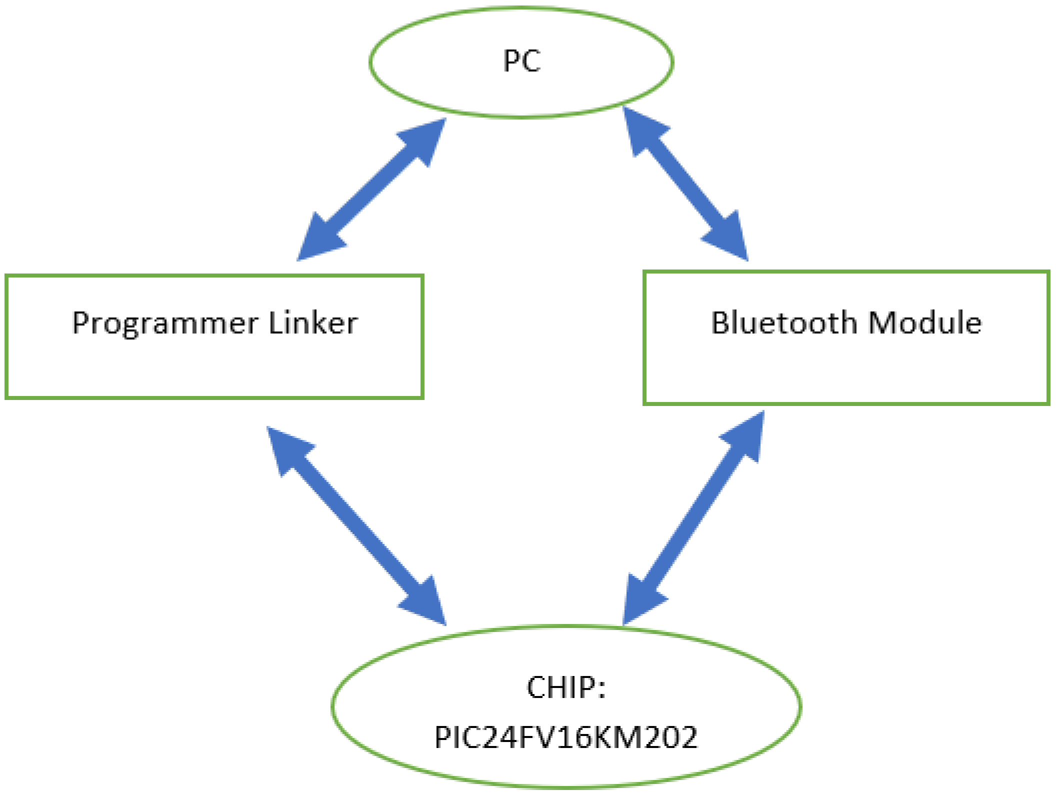


Figure (32) Boot loader system block diagram.

From the CCS Serial Monitor tool to make a connection to Bluetooth and browse the files to compile and download the application program HEX/ICD file which interacted with the bootloader already in the microcontroller and sends the new binary to the bootloader; the bootloader writes the application program's binary to the microcontroller's program memory and then causes the application program to be executed. See figure (33) for bootloader program flowchart and appendix (D) for bootloader code.

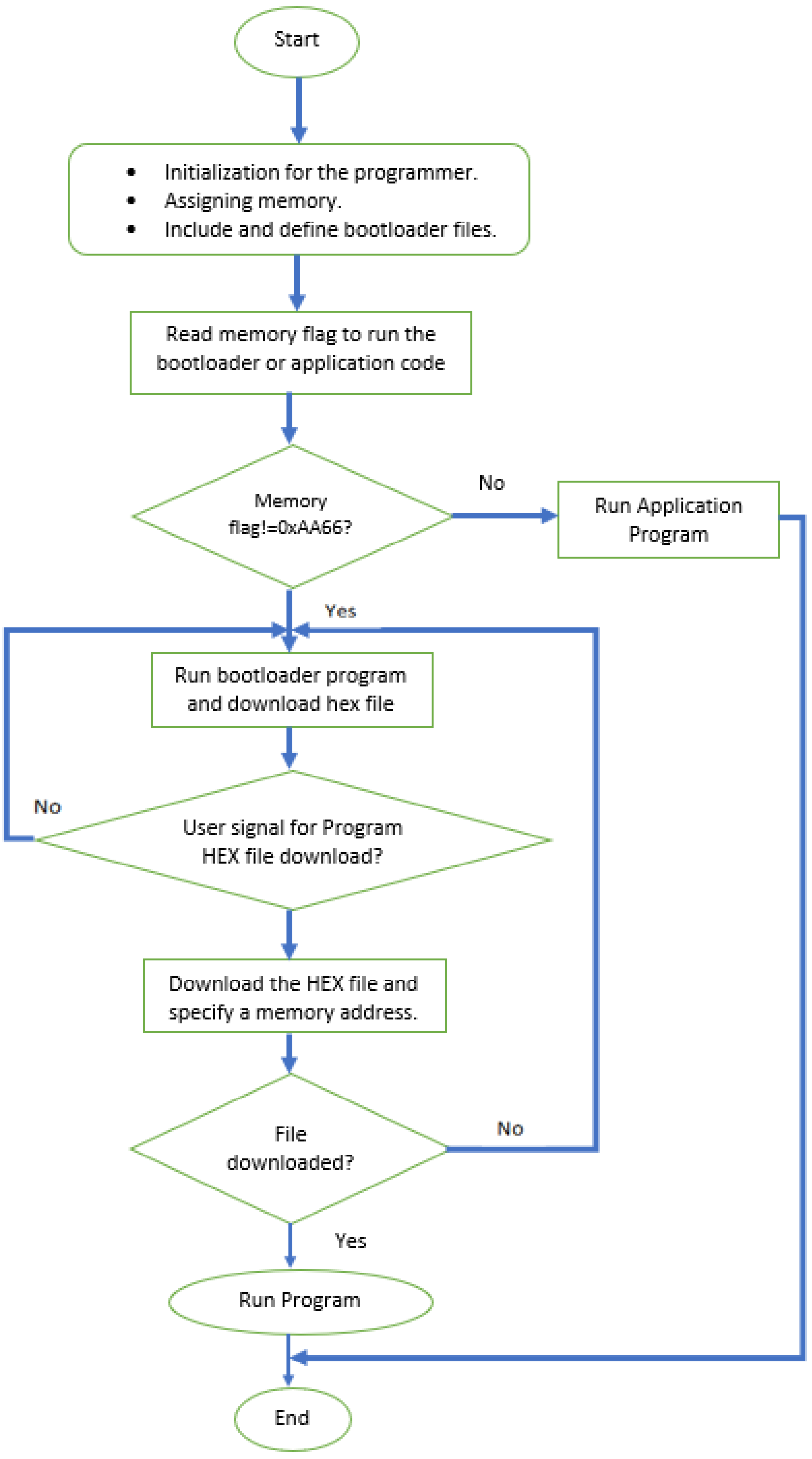
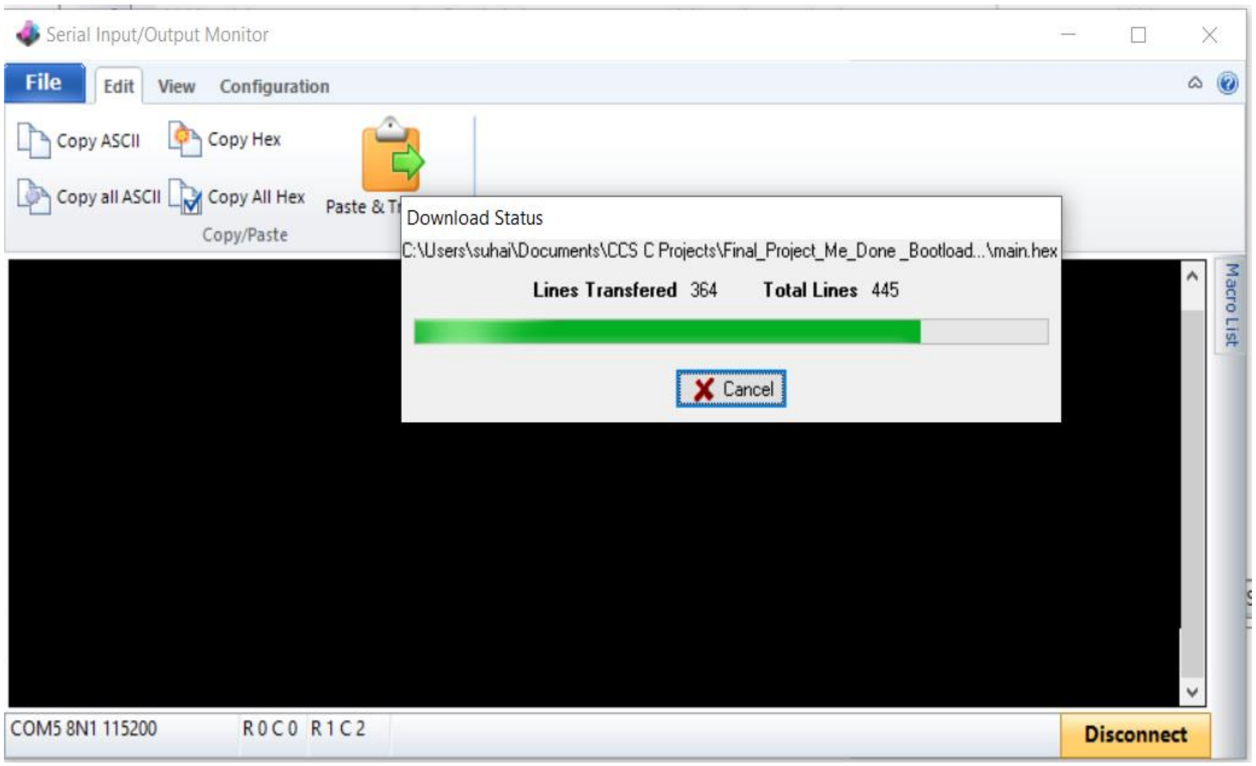

Start
Initialization for the programmer.
Assigning memory.
Include and define bootloader files.
Read memory flag to run the bootloader or application code
Memory flag!=0xAA66?
No
Run Application Program
Yes
Run bootloader program and download hex file
No
User signal for Program HEX file download?
Download the HEX file and specify a memory address.
File downloaded?
No
Yes
Run Program
End

Figure (33) Bootloader Program Flowchart.

The simple operation and function of the bootloader allow the user to download the code via Bluetooth without linking the hardware to the PC. The transferring via Bluetooth occurs without interruption; see figure (34) below for transferring the data via Bluetooth using a boot loader.



Download Status
C:\Users\suhai\Documents\CCS C Projects\Final_Project_Me_Done _Bootload...\main.hex
Lines Transfered 364 Total Lines 445

COM5 8N1 115200 R 0 C 0 R 1 C 2

Figure (34) Hex file Transfer via Bluetooth using Bootloader.

As seen in the figure above, the file is downloaded into the chip and shows the number of lines of the code needed for the project. After it is done downloading, the user can disconnect and reconnect from the C# GUI and run the program.

Analysis and Discussion

This final project to design and develop a real-time digital signal processing using FIR filter and wireless digital oscilloscope and spectral analyzer using PIC24FV16KM202, Bluetooth, and C# GUI was implemented and designed as required and it is met the specification with correct results, except some minor issue with the shape of the plotted wave that has a dent on the second amplitude, which was not perfect because of how was done in the timer and it supposed to be at the accurate timer. The objectives and project specifications as mentioned in the objective and specification sections are conducted. For example, the screenshot in appendix (A) is a picture that was taken during the testing if the PIC C is sending data to GUI using a tera terminal and it was a success. Also, figures (20-31) show the plot of LPF, BPF, DFT, and other parameter values like peak-peak and input-output frequencies. However, the results indicate that the system is working and verified through testing regardless of achieving the right trigger and some minor issues in the plots.

The major roadblock and problems were the planning of how to start the simple code and plotting the first level. The plan was straightforward of how to put the coefficients in C# and make an array to send the data and the timer to interrupt and trigger for the right values. In the beginning, the attempt of sending data using for loop was not successful; it takes 8 seconds to respond and send coefficients, however, using an array to send the data at one time was very fast. Also, another issue is with implementing the boot loader and the tutorials don't say anything about where to include and define the boot loader and other C files within the project code. After a lot of trials, the required lines to be added are successfully added in the right place and the boot loader worked perfectly.

The idea of improving the system is to follow the instruction given in the class about how to do normalization right and the timer in the right place and keep the main() just for printing data. This way the chip will not be very busy.

Overall, all these results and most of the required specifications were achieved, it can be determined that the system project was successfully designed and implemented. Most requirements given for this system were met as expected.

Important things learned:

The most important thing was learned in this project was how the bootloader works. It is interesting to see a code that can interact with a PC via Bluetooth without a hardware external programmer.

Conclusion:

In conclusion, for the design and development of real-time digital signal processing using FIR filter and wireless digital oscilloscope and spectral analyzer using PIC24FV16KM202, Bluetooth, and C# GUI has been developed. The system can plot LPF, BPF, DFT, and show the peak to peak, maximum and minimum, and input-output frequency. The C# GUI has been used to display data. The required results of the system were obtained and plotted and displayed as expected.

Acknowledgment:

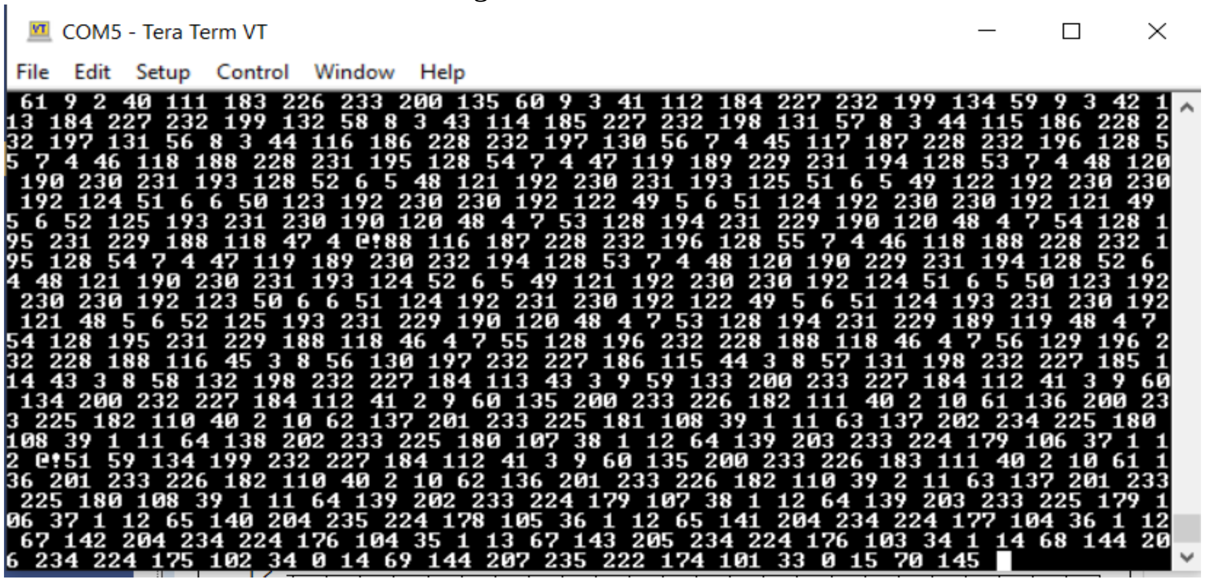
I would like to thank Dr. Yi Zheng for teaching us the skills of how to design the whole system with brilliant ideas that will help us in the future and providing us the sources of how to plot in a real-time using zed graph.

References

1- PIC24FV16KM202_YZ datasheet
2- Real-time plotting with zed graph tutorial.
3- Previous labs and lectures.
4- MATLAB FDA tool tutorial.
5- Bootloader tutorial.
6- Marc Kamsu Bootloader Project codes and tutorial.

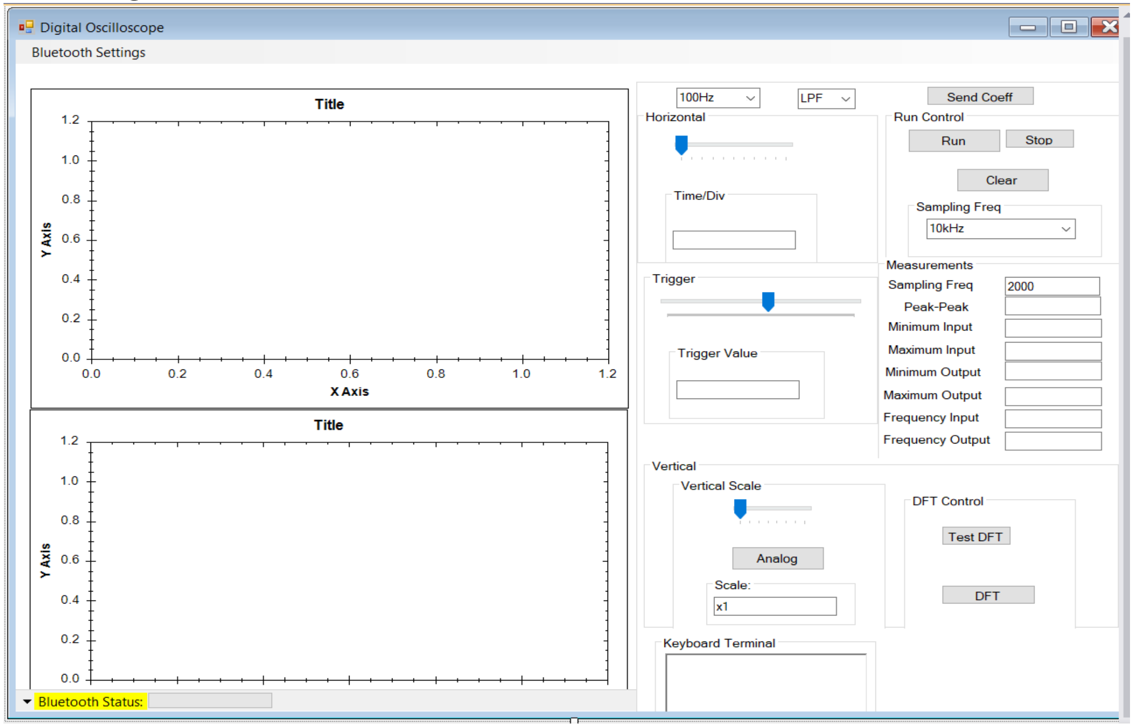
Appendices

A. Screenshot for Tera Terminal sending data:



```
 61 9 2 40 111 183 226 233 200 135 60 9 3 41 112 184 227 232 199 134 59 9 3 42 1
13 184 227 232 199 132 58 8 3 43 114 185 227 232 198 131 57 8 3 44 115 186 228 2
32 197 131 56 8 3 44 116 186 228 232 197 130 56 7 4 45 117 187 228 232 196 128 5
5 7 4 46 118 188 228 231 195 128 54 7 4 47 119 189 229 231 194 128 53 7 4 48 120
 190 230 231 193 128 52 6 5 48 121 192 230 231 193 125 51 6 5 49 122 192 230 230
 192 124 51 6 6 50 123 192 230 230 192 122 49 5 6 51 124 192 230 230 192 121 49
5 6 52 125 193 231 230 190 120 48 4 7 53 128 194 231 229 190 120 48 4 7 54 128 1
95 231 229 188 118 47 4 @!88 116 187 228 232 196 128 55 7 4 46 118 188 228 232 1
95 128 54 7 4 47 119 189 230 232 194 128 53 7 4 48 120 190 229 231 194 128 52 6
4 48 121 190 230 231 193 124 52 6 5 49 121 192 230 230 192 124 51 6 5 50 123 192
 230 230 192 123 50 6 6 51 124 192 231 230 192 122 49 5 6 51 124 193 231 230 192
 121 48 5 6 52 125 193 231 229 190 120 48 4 7 53 128 194 231 229 189 119 48 4 7
54 128 195 231 229 188 118 46 4 7 55 128 196 232 228 188 118 46 4 7 56 129 196 2
32 228 188 116 45 3 8 56 130 197 232 227 186 115 44 3 8 57 131 198 232 227 185 1
14 43 3 8 58 132 198 232 227 184 113 43 3 9 59 133 200 233 227 184 112 41 3 9 60
 134 200 232 227 184 112 41 2 9 60 135 200 233 226 182 111 40 2 10 61 136 200 23
3 225 182 110 40 2 10 62 137 201 233 225 181 108 39 1 11 63 137 202 234 225 180
108 39 1 11 64 138 202 233 225 180 107 38 1 12 64 139 203 233 224 179 106 37 1 1
2 @!51 59 134 199 232 227 184 112 41 3 9 60 135 200 233 226 183 111 40 2 10 61 1
36 201 233 226 182 110 40 2 10 62 136 201 233 226 182 110 39 2 11 63 137 201 233
 225 180 108 39 1 11 64 139 202 233 224 179 107 38 1 12 64 139 203 233 225 179 1
06 37 1 12 65 140 204 235 224 178 105 36 1 12 65 141 204 234 224 177 104 36 1 12
 67 142 204 234 224 176 104 35 1 13 67 143 205 234 224 176 103 34 1 14 68 144 20
6 234 224 175 102 34 0 14 69 144 207 235 222 174 101 33 0 15 70 145
```

B. GUI Design:



100Hz

LPF

Sampling Freq: 10kHz

Sampling Freq: 2000

Scale: x1

C. DFT calculation code in GUI:

```
        //calculates magnitude and phase of one signal and stores in the buffer
        private void calculate_dft_magnitude_phase(int index)
        {
            int rem = 0;
            double a;
            double real = 0,
                img = 0;

            for (int i = 0; i < BUFFER_SIZE; i++)
            {
                a = time_domain_signal_buffer[i] * 5.0 / (256.0 * BUFFER_SIZE);
                Math.DivRem(i, 2, out rem);
                if (rem != 0)
                    a *= -1;

                real += (double)(a * Math.Cos(2 * (Math.PI / BUFFER_SIZE) * i * index));
                img += (double)(-1 * a * Math.Sin(2 * (Math.PI / BUFFER_SIZE) * i *
index));
            }

            double mag = (double)(Math.Sqrt(Math.Pow(real, 2) + Math.Pow(img, 2)));
            //calculating DFT magnitude
            DFT_amplitude_signal_buffer[index] = mag;   //storing DFT magnitude

            double phase = (double)(Math.Atan2(img, real) * 180 / Math.PI);
            //calculating DFT phase
            DFT_phase_signal_buffer[index] = phase; //storing DFT phase
        }
```

Function to extracts DFT magnitude samples:

```
        private double[] teamB()    //extracts DFT magnitude samples
        {
            double[] a = new double[BUFFER_SIZE];
            double[] b = new double[BUFFER_SIZE / 2];

            for (int i = 0; i < BUFFER_SIZE; i++)
            {
                //store values from DFT amplitude signal buffer
                a[i] = DFT_amplitude_signal_buffer[i];
                if (i > BUFFER_SIZE / 2)
                {
                    b[i - (BUFFER_SIZE / 2)] = DFT_amplitude_signal_buffer[i];
                }
            }
            double maxValue = b.Max();  //finding max value
            int index = Array.IndexOf(b, maxValue); //finding index of max value

            float freq = (float)((max_sampling_frequency / (1.0 * BUFFER_SIZE)) * index);
            //converting index of max value into frequency of input signal
            //freq_box.Text = freq.ToString();    //displaying the frequency to text box
            freq_box.Text = string.Format("{0:N2}", input_freq);//input_freq.ToString();

            return a;
        }
```

D. Boot loader code:

Bootloader Code

```
//////////////////////////////////////////////////////////////////////////////
/////////////////
////                           BOOTLOADER.C
////
////
////
////  This program is an example standalone bootloader.
////
////
////
////  This program must be loaded into a target chip using a device
////
////  programmer.  Afterwards this program may be used to load new
////
////  versions of the application program.
////
////
////
////  Before loading this program into the target chip, please first
////
////  copy the "loader_pcd.c" file located at C:\Program Files
(x86)\PICC\Drivers         ////
////  into the same directory as this program. Then, comment out line 239
"reset_cpu();"  ////
////  in the "loader_pcd.c" file.
////
////
////
////  Use an RS232 link and the SIOW.EXE program to load a new HEX
////
////  file into the target chip.
////
////
////
//////////////////////////////////////////////////////////////////////////////
/////////////////
//////////////////////////////////////////////////////////////////////////////
/////////////////

#include <24FV16KM202.h>
#device icd=3
#fuses NOWDT
#use delay(internal=32mhz)
#use RS232(UART2, BAUD = 115200, PARITY = N, BITS = 8, STOP = 1)  // set up
bluetooth

#define _bootloader

#define LOADER_PAGES 50 //assign enough memory space for the bootloader
#include <pcd_bootloader.h>
#include "loader_pcd.c" //include local copy of loader_pcd.c

//One of the EEPROM address is used to store a flag variable for deciding
//whether to run the bootloader or the application.
//To run the bootloader again after the application was bootloaded, this flag
```

```
//has to be cleared in the application and the chip needs to be reset.
#define RUN_APPLICATION_FLAG        0xAA66
#define RUN_APPLICATION_FLAG_ADDR   0

#org APPLICATION_START
void application(void)
{
   while(TRUE);
}

void main(void)
{
   unsigned int16 eeFlag;

   // Wait for PLL
   delay_ms(140);

   eeFlag = read_eeprom(RUN_APPLICATION_FLAG_ADDR);

   if(eeFlag != RUN_APPLICATION_FLAG)
   {
      // Let the user know it is ready to accept a download
      printf("\r\nBootloader for PIC24FV16KM202 is running...\r\n");
      printf("\r\nWaiting for HEX file download...\r\n");

      // Load the program
      load_program();

      // EEPROM address is used to store a flag variable for deciding
      //whether to run the bootloader or the application.
      write_eeprom(RUN_APPLICATION_FLAG_ADDR, RUN_APPLICATION_FLAG);
   }

   // Bootloader has been loaded and the chip is ready for the application
code
   application();
}

#int_default
void isr(void)
{
   jump_to_isr(LOADER_END+5);
}
```