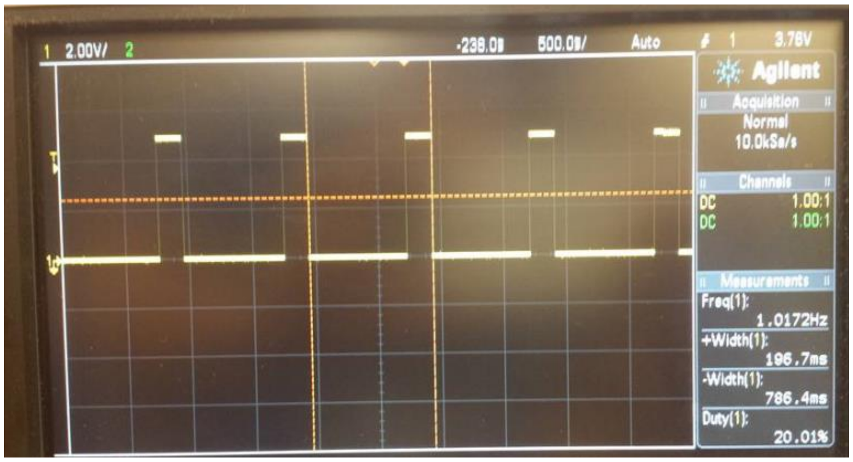


ECC 422 Midterm

Square wave

Abstract

This is a report that covers the design and implementation of the square wave signal generator. Two 16 bit timers of PIC18f4455 microcontroller are used to change the state of the wave and also used for signal capture measurement. Also talks about the GUI application that control the entire signal generator.

Khadka, Ashish

ECE 422

Contents

1 Introduction:

The main objective of the lab is to generate square wave from 1 Hz to 1 KHz with controllable duty cycle form a user input. Also, to grate Graphical User Interface (GUI) so user have more control over the application. To accomplish this project times are used to count the time.

1.1 Components Used

- PIC18F4455
- 20 MHz crystal oscillator
- RJ CCS Compiler
- Prototype Board
- 5 V Volt Power Supply
- ICD
- USB cord
- RJ12 tubular
- Serial to Prototype board cable
- Keypad
- 220 Ω
- Push Button
- 10k potentiometer
- Two 0.1uF, capacitors

1.2 Software Used

- CCS C Compiler (Firmware code)
- Tera Term Emulator
- Visual Studio (GUI Creation)

2 Theory

To create signal timers plays vital role. For this project timer_0 and timer_1 is used for generating square wave signal and capturing wave period respectively. Timer_0 and timer_1 are configured as 16 bits. With the help of the timer, counter is created which counts with every clock. The total counts we can form each timer is 65535 counts and then the counter over flows and goes back to 0.

16 bit timer count limit = $2^{16} = (65536)_{10}$

Since counter starts form 0,

Max count = $(65536-1)_{10} = (65535)_{10} = (0xFFFF)_{16}$

With the help of counter an internal interrupt is created which is used to overflows, triggering external interrupt. Thus, at every flow the interrupt routine is setup with starting value. With every interrupt caused by the timer the Pin A3 in PIC-18F4455 is set high and low depending upon the condition. The conditions are set based on the duty cycle which defines how low the signal should be turned on or off. The conditions are created based on the user input of the duty cycle. The entered duty cycle is calculated form the period of the signal that user inputs. The user inputs the Frequency and the period gets calculated in microseconds. Based on the period the duty cycle is calculated using equation I

$$Period(T) = \frac{1}{f} \qquad \text{--I}$$

$$on\_time\_duty = \frac{Duty\ Cycle}{100} * Period \qquad \text{---II}$$

$$off_{\text{time}_{\text{duty}}} = Period - on\_time\_duty \qquad \text{--III}$$

Then the ticks are calculated for on time and off time for a signal to control the square wave. Based on the fact the clock is that is running the micro controller is 48 MHz the ticks of the counter are calculated with respect to clock. Since, the PIC chip uses 4 clock cycle per instruction the actual clock of the microcontroller is 12 MHz.

$$Microcontrollar\ clock = \frac{External\ Clock}{Instructon\ per\ clock} = \frac{48}{4} = 12 \text{ MHz}$$

After the clock is known the ticks of the clock is calculated. With every ticks of the clock a counter is raised. We know that the clock is running at 12 MHz Thus the period of the clock is

$T = \frac{1}{12*10^6}$ = 83.33 ns

This means the counter can count 1 in 83.33 ns with every tick of the clock. For 16 bit timer it would take 5.46 ms. The math is shown below

$$Time\ for\ overflow = Time\ for\ one\ tick * number\ of\ \max counts$$

$$Time\ for\ overflow = 83.33\, ns * 65535 = 5.46\, ms$$

Based on this information an interrupt routine is set up which counts the number of interrupts. Thus, the output pins on the microcontroller is toggled if the conditions match.

This interrupt routine is set in such way that it would track the time it needs to set the timers for before it goes to interrupt. The track of time is kept using this equation below

$$ON_{time_{tick}} = Max\ count - count\ to\ begin\ for\ positive\ duty\ cycle\ * Clock\ Speed --- IV$$

$$OFF_{time_{tick}} = Max\ count - \ count\ to\ begin\ for\ negative\ duty\ cycle * Clock\ Speed --- V$$

Count to begin is calculated using equation below

$$count\ to\ begin\ for\ positive\ duty\ cycle\ = \frac{on\_time\_duty}{time\ for\ one\ tick} --- VI$$

$$count\ to\ begin\ for\ negative\ duty\ cycle\ = \frac{off\_time\_duty}{time\ for\ one\ tick} --- VII$$

So, there are two count to begin due to the fact it's not DC and the counter needs to count for how long to turn the signal on for and how long to turn in off. After the time is known equation the interrupt is set which controls the on and off state of the wave.

This is only valid only if the period of the signal is less than 5.46 ms due to the fact the counter would count form 0 to 65535 before it goes to over flow in 5.56 ms. Thus, the max positive or negative width is 5.46 ms creating a period of lowest signal 16 bit counter would produce with a clock speed of 12 MHz is

T=5.46 ms *2 = 10.92 ms ---VIII

$$F = \frac{1}{T} = \frac{1}{10.92\ ms} = 91.57\ Hz \qquad ---IX$$

This is the limitation of the counter which limits the capability of generating signals less than 91.5 Hz. To overcome it number of over flow is counted and compensated.

3 <u>Design :</u>

To overcome the task the problem is divided in small modules. These modules are the state machine which control the flow of the system. The state machine designed for the project is shown below in figure 1.

3.1 State Machine

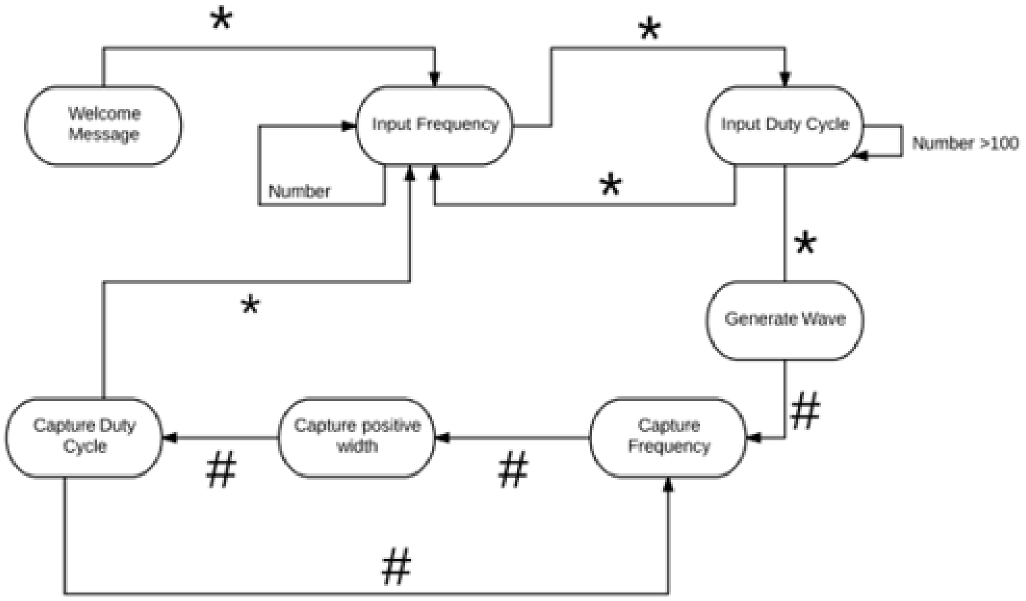


Figure 1 : System Flow

The system flow included the combination of Mealy type and Moore type state machine as shown in *figure 1*. As it can be seen in the *figure 1* that different state perform different action. The states of the machine does not change unless user inputs '*' or '#' symbol from GUI or from the key pad. Here, in the system '*' key is used to get information from the user whereas '#' symbol is used to measure the signal integrity. Once, the microcontroller gets the input from user each state perform a particular task. The first 4 state are used to display welcome message, get input frequency, get duty cycle and generate wave respectively whereas last three states are used to capture the wave information. The last three state are capture frequency, capture positive width and duty cycle respectively.

Each state machine performs individual task. Basically the project is divided into two modes. i.e. wave generation and wave capture mode.

3.2 Wave Generation

Figure 2 shows the logic behind for the square wave generation with controllable duty cycle. As can be seen in *figure 2* that for duty cycle the on time and the off time are converted to microsecond and then converted to ticks that counter counts with one period. Now, the ticks are known thus the interrupt routine is set. With the interrupt the high and low state are controlled, depending upon the factor whether the ticks are overflowing or not. If there is wave needs to be generated which has resolution less than 5.46 ms then there is no need of counter because the resolution of the wave is within the limit of overflow time. However, if there is overflow then the timer is set in interrupt routine with the residue of overflow that is in decimal. Once the routine interrupts then another interrupt is set with integer that is left which is tick. Now when interrupt occurs the output of the pin is toggled depending upon state. Similarly, another state is achieved and vice versa.

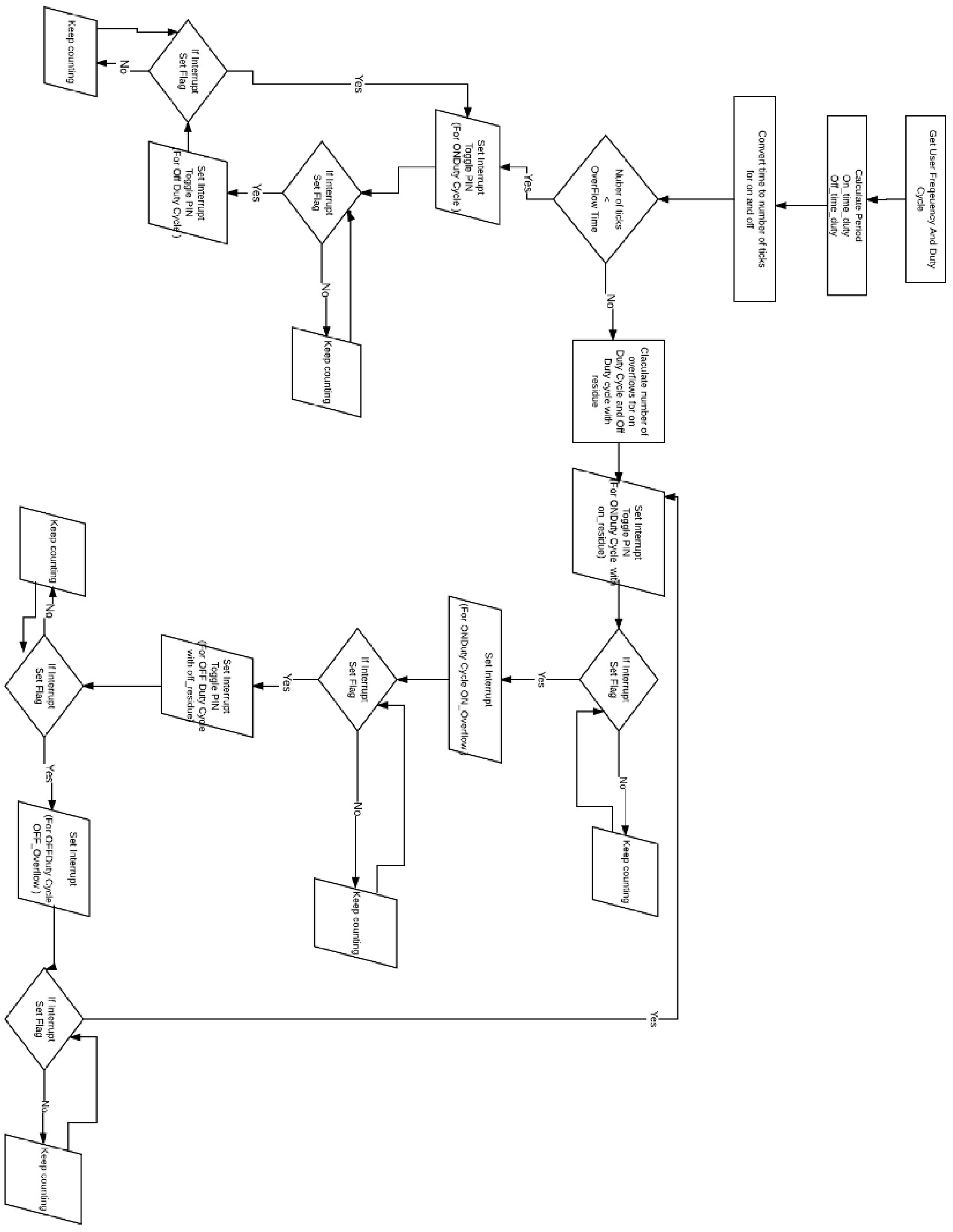

Get User Frequency And Duty Cycle
Calculate Period On_time_duty Off_time_duty
Convert time to number of ticks for on and off
Number of ticks < OverFlow Time
Yes
No
Set Interrupt Toggle PIN (For ONDuty Cycle)
If Interrupt Set Flag
Yes
No
Keep counting
Set Interrupt Toggle PIN (For Off Duty Cycle)
If Interrupt Set Flag
Yes
No
Keep counting
Claculate number of overflows for on Duty Cycle and Off Duty cycle with residue
Set Interrupt Toggle PIN (For ONDuty Cycle with on_residue)
If Interrupt Set Flag
Yes
No
Keep counting
Set Interrupt (For ONDuty Cycle ON_Overflow)
If Interrupt Set Flag
Yes
No
Keep counting
Set Interrupt Toggle PIN (For OFF Duty Cycle with off_residue)
If Interrupt Set Flag
Yes
No
Keep counting
Set Interrupt (For OFFDuty Cycle OFF_Overflow)
If Interrupt Set Flag
Yes
No
Keep counting

Figure 2: Wave Generation Flow Chart

3.3 Wave Capture

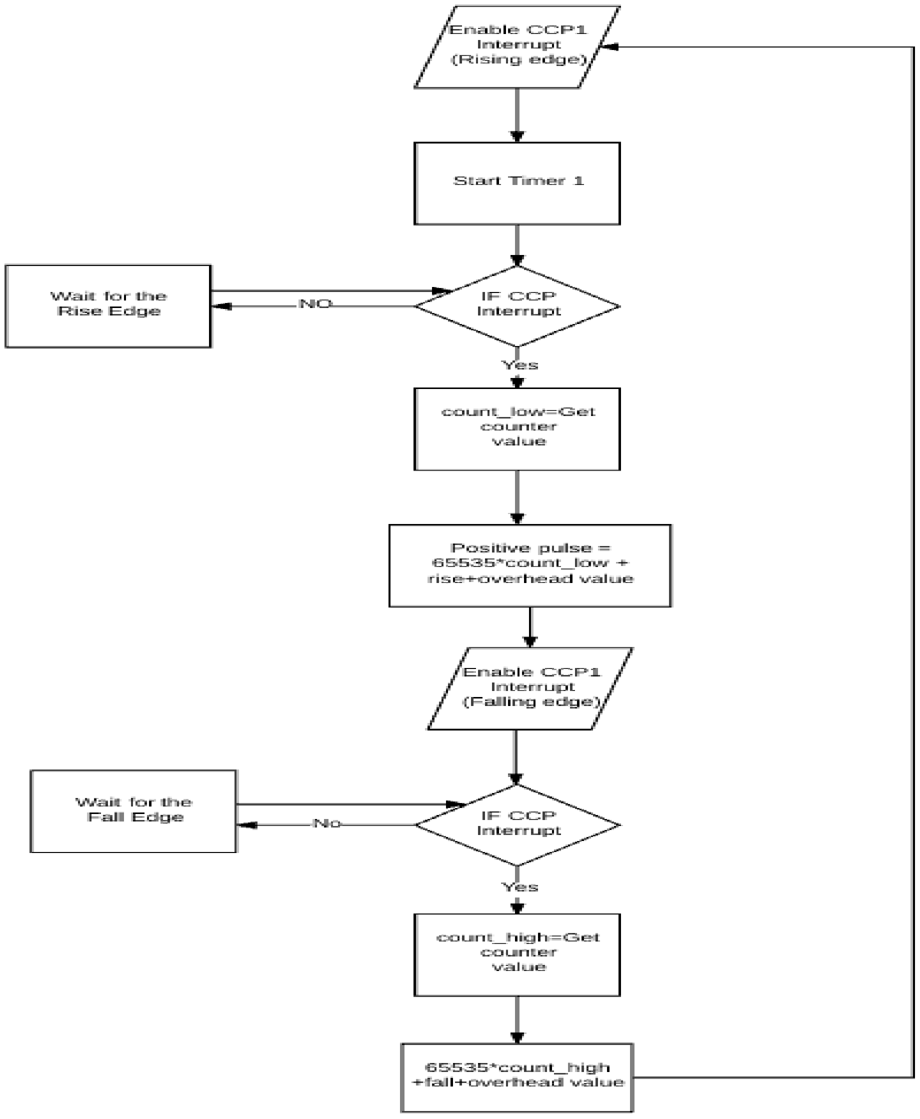

Enable CCP1 Interrupt (Rising edge)
Start Timer 1
Wait for the Rise Edge
NO
IF CCP Interrupt
Yes
count_low=Get counter value
Positive pulse = 65535*count_low + rise+overhead value
Enable CCP1 Interrupt (Falling edge)
Wait for the Fall Edge
No
IF CCP Interrupt
Yes
count_high=Get counter value
65535*count_high +fall+overhead value

Figure 3: Wave Capture flow Chart

Figure 3: shows the design that is used to capture the wave information. When the user wants to capture the signal then the timer 1 is enabled and capture mode of the CCP pin on the microcontroller is enabled. Nothing happens until the rising edge of wave is capture by the CCP pin . Once the Rising edge is capture then the timer value is passed

to a variable. Simultaneously, timer1 is also running counting number of overflows. If there is not over flow then Positive pulse value would be given by the following equation.

$$Positive\ Edg = 65535 * counteroverflow + measure_{count_{from_{CCP}}} + overhead - - - X$$

There is no overflow for pulse which have width less than 5.46 ms. So, the counter overflow value would be 0. Thus only measure count from CCP would be the actual count. But if there is overflow due the fact that there is overflow then the counter would have integer value which can be plugged in equation X to get the number of ticks. After tick is calculated then the CCP pin is setup for Falling Edge. And this runs in circle counting ticks for both edges. After having ticks for both falling and rising edge frequency of signal can be calculated using the flowing equations.

$$F = \frac{12000000}{PW1 + PW2} - - - XI$$

Where PW1 and PW2 are positive width and negative edges respectively and 12 MHz is the clock speed at which the timer1 is running. Plugging the variables would give the captured frequency.

And measured duty cycle is given by

$$measured\_duty \ = \ (100 * P\_W1)/(P\_W1 + P\_W2) \quad XII$$

3.4 Hardware Schematics:

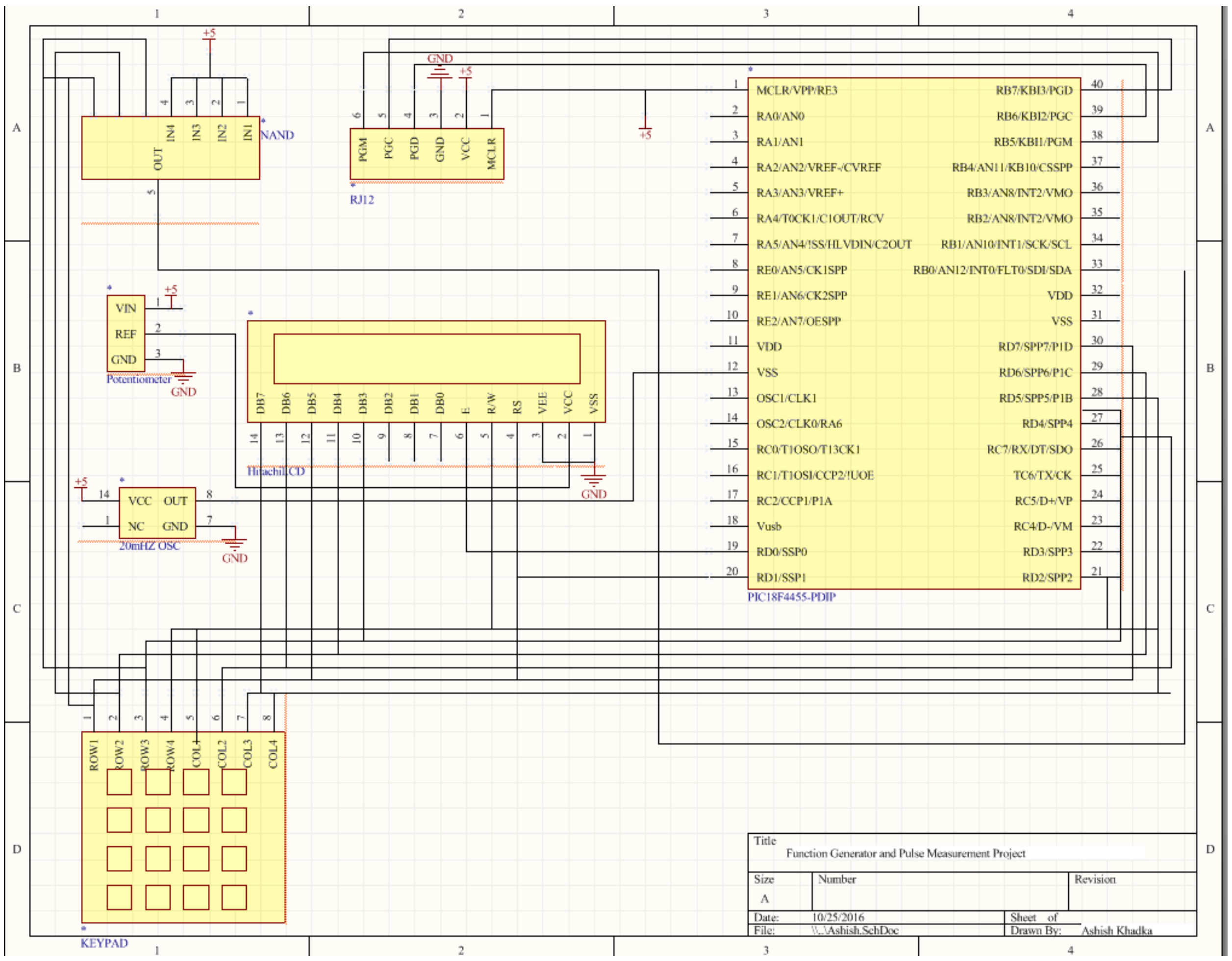


Figure 3: Altium Schematics

Figure 3 shows the breadboard design in Altium which be later placed for PCB Implementation if needed in the future.

3.5 Hardware Setup

For the project the hardware is setup for external interrupts. There are several components that completes the circuits. The main components for the project are explained down below with the connections.

I. Clock :

An external clock is used to drive the microcontroller. The Crystal oscillator of 20 MHz is used to for clocking the PIC18f4455 microcontroller. But in software PLL fuse is used to is used to crank up the clock to 48 Mhz. However, the PIC18F4455 take 4 clock cycles to get one instruction done thus clock running is now running at 12 MHz which makes the period of 83.33 ns.

II. Bluetooth :

RN42I/RM Bluetooth module is used for serial communication (RS232) between GUI and the microcontroller. The RX of Bluetooth is connected to TX pin 25 of the microcontroller and TX of Bluetooth is connected to RX of the microcontroller which is pin 26 on the microcontroller.

III. Keypad (96BB2) :

This keypad is integrated with Matrix Circuitry which works with active low logic. The first 7 pins of the keypad are connected to RD1 to RD7 of the PIC micron roller where as D7-D4 of the Hitachi LCD to send the signal. And the first 3 pins are connected to Enable, Write Select and Read Write pin of the LCD respectively.

Also, the output form the keypad is used to setup external interrupt. All the line from D7 to D4 are connected to NAND gate. And the output of the NAND gate is fetched to the External Interrupt (INT0) pin of the PIC18F4455 which is pin 33 also called as External Interrupt 0 for the microcontroller.

IV. LCD (Hitachi HD44780U)

The HD44780U is a dot-matrix liquid crystal display controller and driver LSI displays alphanumeric, Japanese kana characters, and symbols**.** This LCD can display up to one

8-character line or two 8-character lines. The pins of the LCD are connected to RD0 to RD7 of the microcontroller.

V. RJ12 Connector :

A Registered Jack (RJ) is used to transfer bits to a microcontroller while programming the PIC18F4455 device.

VI. USB Connector with PCB

This device is used to power up the circuit with 5 volts form USB.

VII. ICD-U64

ICD is used for in-circuit serial programming (ICSP). Also ICD-U64 is used for Circuit programming and Circuit Debugging developed by Microchip's. ICD-U64 debug support covers all targets that have debug mode when used in conjunction with the C-Aware IDE Compilers.

VIII. PIC18F4455

This is the brain of the hardware which controls every peripheral l attached to it. The Pic microcontroller interacts with every device and functions as a team. This is where the code is implemented and executed with every clock.

4 Implementation

4.1 GUI Operation

A simple GUI is created which controls the whole system shown in figure 4. User has to click on '*' to input the data whereas click on '#' to capture the data.

After the system starts a welcome message is prompted on *Work Screen*. Nothing happens until user clicks '*'. As soon as the user clicks on '*' the Work Screen would display a message “Enter Frequency”. After user inputs the frequency in Hz *Entered Frequency* is filled with the input frequency then then user has to input the duty cycle of the square wave by clicking ‘*’ on the GUI. The input Duty Cycle is printed in *Entered Duty Cycle Box*. The user has to click ‘*’ to generate the wave. Then the *Work Screen* would display message “Generating Wave”. Now the generating part is done. But to capture user has to press ‘#’. As soon as ‘#’ is pressed the *Measured Frequency* box is filled with the measured frequency value calculated with the help of CCP pin in the microcontroller. Now to measure the positive pulse width the user has to press ‘#’ and the *Measure Positive Pulse* is filled with the captured value. And at last to measure Duty Cycle user has to press ‘#’ to capture the Duty cycle of the wave and the *Measured Duty Cycle* box is filled with the captured value.

4.2 GUI Design

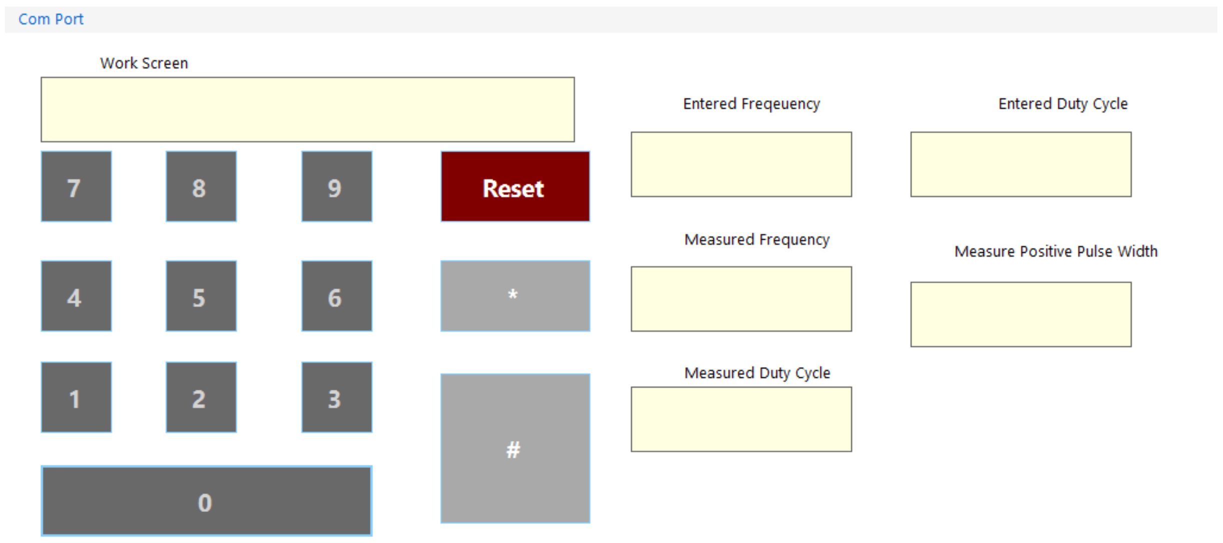
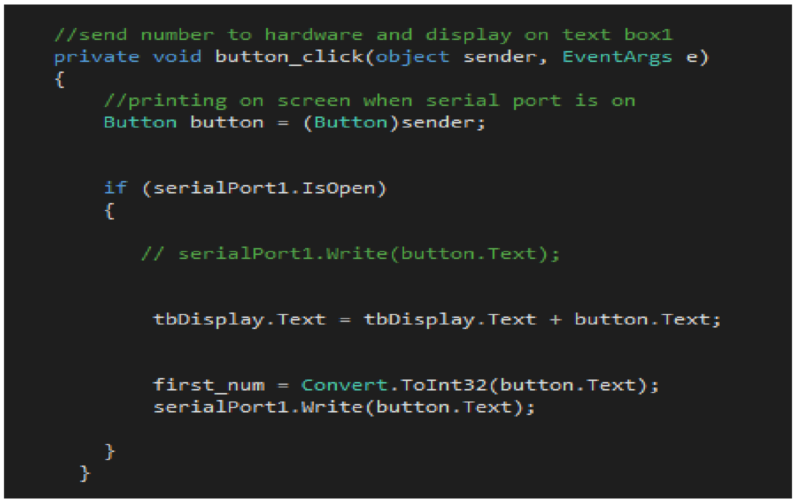


Figure 4: GUI for Square Wave Generator

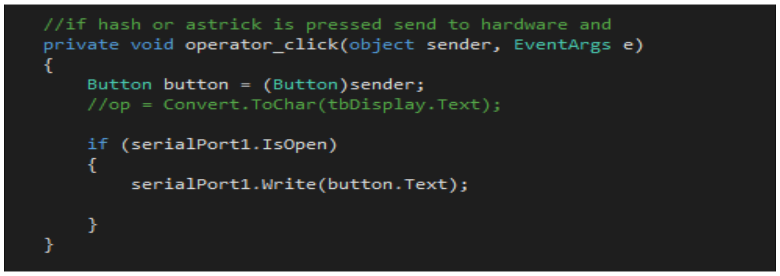
4.3 GUI code (Creation)



```
//send number to hardware and display on text box1
private void button_click(object sender, EventArgs e)
{
    //printing on screen when serial port is on
    Button button = (Button)sender;


    if (serialPort1.IsOpen)
    {

       // serialPort1.Write(button.Text);


        tbDisplay.Text = tbDisplay.Text + button.Text;


        first_num = Convert.ToInt32(button.Text);
        serialPort1.Write(button.Text);

    }
  }
```

Figure 5: C# Event Handler for Numbers

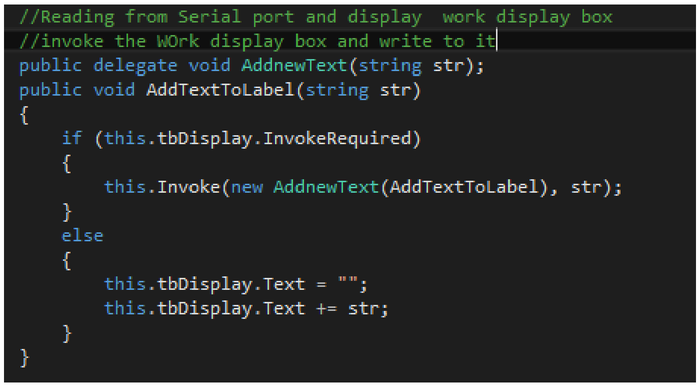
To send numbers to the microcontroller though serial port a Button instance is created form button class which has a character embedded to it. Here, all the numeric buttons are grouped into single event handler called button_click() which converts the character to Int32 before the GUI sends the number to microcontroller triggering interrupt.



```
//if hash or astrick is pressed send to hardware and
private void operator_click(object sender, EventArgs e)
{
    Button button = (Button)sender;
    //op = Convert.ToChar(tbDisplay.Text);

    if (serialPort1.IsOpen)
    {
        serialPort1.Write(button.Text);

    }
}
```

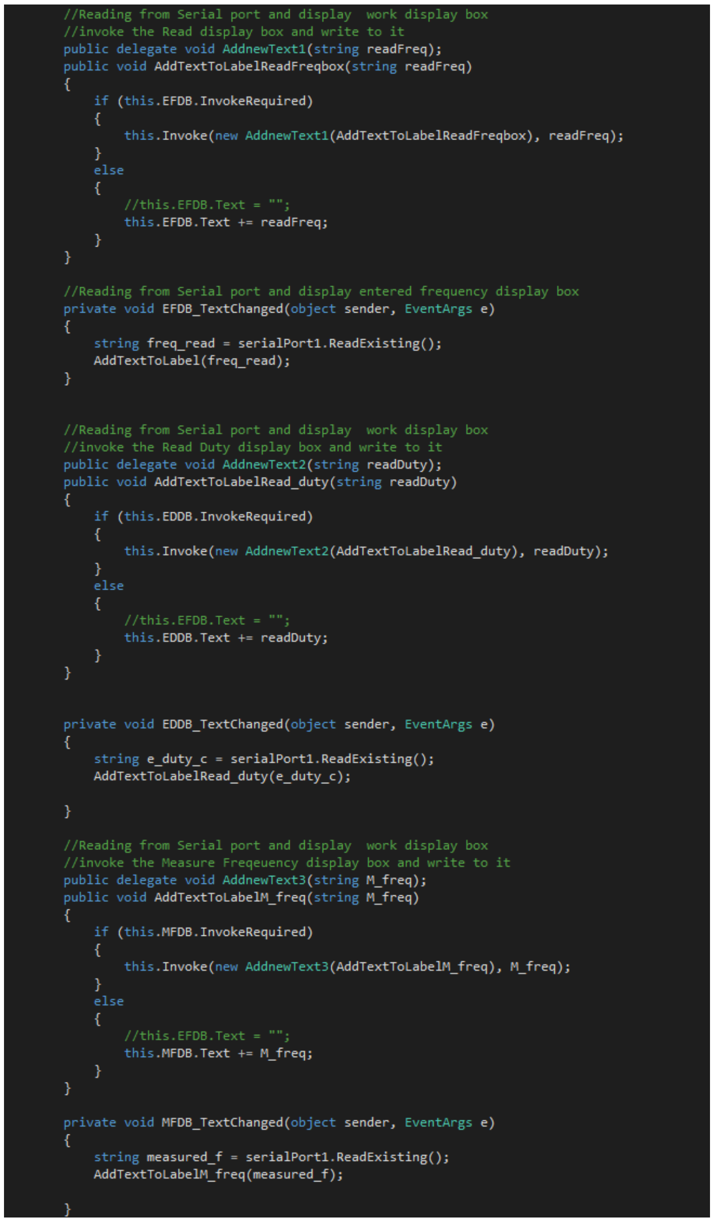
Figure 6 : C# Event Handler for Operator

Similarly in *figure 6*, it can be seen that Button instance is created from Button class to send the character which are operators '#' and '*' triggering interrupt.



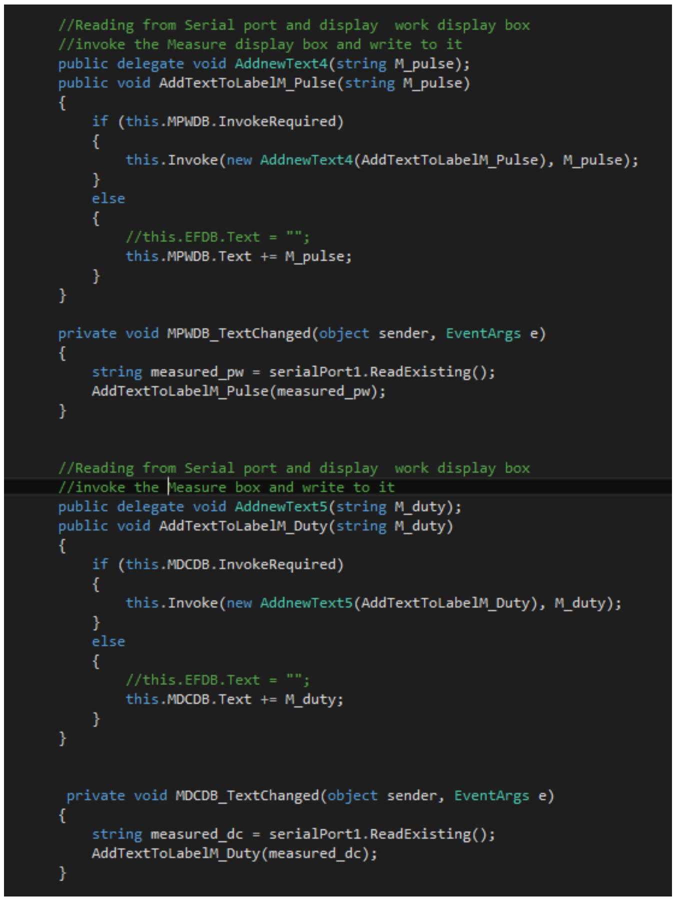
```
//Reading from Serial port and display  work display box
//invoke the WOrk display box and write to it
public delegate void AddnewText(string str);
public void AddTextToLabel(string str)
{
    if (this.tbDisplay.InvokeRequired)
    {
        this.Invoke(new AddnewText(AddTextToLabel), str);
    }
    else
    {
        this.tbDisplay.Text = "";
        this.tbDisplay.Text += str;
    }
}
```

Figure 7 : Invoking the Work Space Displace Box



```
//Reading from Serial port and display  work display box
//invoke the Read display box and write to it
public delegate void AddnewText1(string readFreq);
public void AddTextToLabelReadFreqbox(string readFreq)
{
    if (this.EFDB.InvokeRequired)
    {
        this.Invoke(new AddnewText1(AddTextToLabelReadFreqbox), readFreq);
    }
    else
    {
        //this.EFDB.Text = "";
        this.EFDB.Text += readFreq;
    }
}

//Reading from Serial port and display entered frequency display box
private void EFDB_TextChanged(object sender, EventArgs e)
{
    string freq_read = serialPort1.ReadExisting();
    AddTextToLabel(freq_read);
}


//Reading from Serial port and display  work display box
//invoke the Read Duty display box and write to it
public delegate void AddnewText2(string readDuty);
public void AddTextToLabelRead_duty(string readDuty)
{
    if (this.EDDB.InvokeRequired)
    {
        this.Invoke(new AddnewText2(AddTextToLabelRead_duty), readDuty);
    }
    else
    {
        //this.EFDB.Text = "";
        this.EDDB.Text += readDuty;
    }
}


private void EDDB_TextChanged(object sender, EventArgs e)
{
    string e_duty_c = serialPort1.ReadExisting();
    AddTextToLabelRead_duty(e_duty_c);

}

//Reading from Serial port and display  work display box
//invoke the Measure Freqeuency display box and write to it
public delegate void AddnewText3(string M_freq);
public void AddTextToLabelM_freq(string M_freq)
{
    if (this.MFDB.InvokeRequired)
    {
        this.Invoke(new AddnewText3(AddTextToLabelM_freq), M_freq);
    }
    else
    {
        //this.EFDB.Text = "";
        this.MFDB.Text += M_freq;
    }
}

private void MFDB_TextChanged(object sender, EventArgs e)
{
    string measured_f = serialPort1.ReadExisting();
    AddTextToLabelM_freq(measured_f);

}
```

Figure 8: Invoking the Display Boxes



```
//Reading from Serial port and display  work display box
//invoke the Measure display box and write to it
public delegate void AddnewText4(string M_pulse);
public void AddTextToLabelM_Pulse(string M_pulse)
{
    if (this.MPWDB.InvokeRequired)
    {
        this.Invoke(new AddnewText4(AddTextToLabelM_Pulse), M_pulse);
    }
    else
    {
        //this.EFDB.Text = "";
        this.MPWDB.Text += M_pulse;
    }
}

private void MPWDB_TextChanged(object sender, EventArgs e)
{
    string measured_pw = serialPort1.ReadExisting();
    AddTextToLabelM_Pulse(measured_pw);
}


//Reading from Serial port and display  work display box
//invoke the Measure box and write to it
public delegate void AddnewText5(string M_duty);
public void AddTextToLabelM_Duty(string M_duty)
{
    if (this.MDCDB.InvokeRequired)
    {
        this.Invoke(new AddnewText5(AddTextToLabelM_Duty), M_duty);
    }
    else
    {
        //this.EFDB.Text = "";
        this.MDCDB.Text += M_duty;
    }
}


 private void MDCDB_TextChanged(object sender, EventArgs e)
{
    string measured_dc = serialPort1.ReadExisting();
    AddTextToLabelM_Duty(measured_dc);
}
```

Figure 9: Invoking the Display Boxes contd

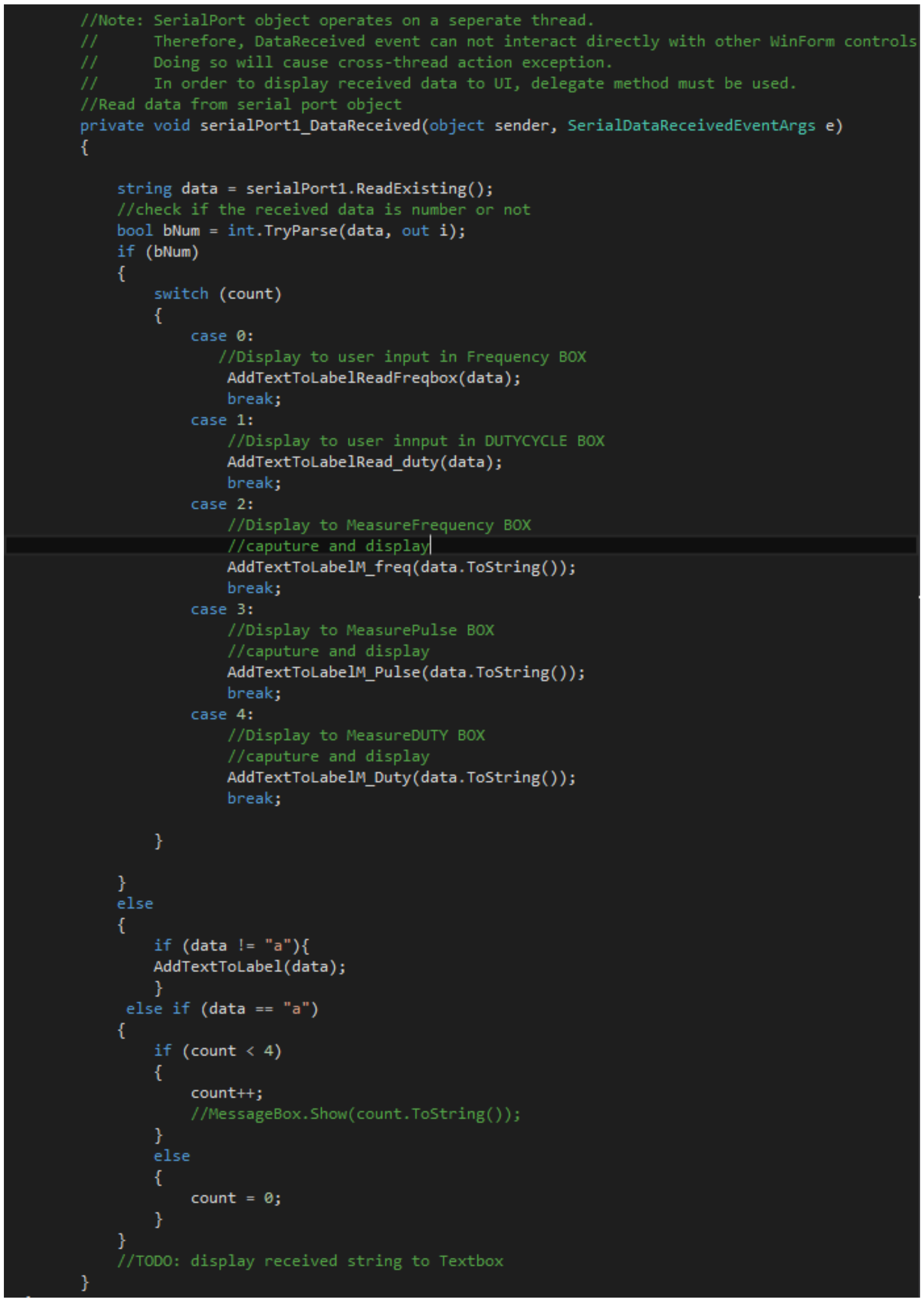
Figure 5,6 shows the functions created for each display box to filled with the specific values after user gives an input. In figure 4 GUI it can be seen that there are 6 Display boxes. Their functionality are shown down below

- Entered Frequency Box: Fills in entered frequency by user
- Entered Duty Cycle: Fills entered duty cycle from the user is filled
- Measured Frequency: Displays measured frequency by PIC18F4455
- Measure Positive Width: Displays measured positive pulse width by PIC18F4455
- Measured Duty Cycle: Displays Measured duty cycle by PIC18F4455
- Work Screen: Displays the user prompt messages

As, shown in figure 7,8,9 separate functions are created for each display box to print the data received form serial port . The function in figure are described below

- **void AddTextToLabel(string str) :** Reads the messages from serial port
- **public void AddTextToLabelReadFreqbox(string readFreq) :** Reads the entered Frequency from serial port
- **public void AddTextToLabelRead_duty(string readDuty)** : Reads the Duty Cycle from serial port
- **public void AddTextToLabelM_freq(string M_freq)** : Reads the measured Frequency from serial port
- **public void AddTextToLabelM_Pulse(string M_pulse) :** Reads the measured width of Pulse
- **public void AddTextToLabelM_Duty(string M_duty) :** Reads the measured Duty Cycle of the wave

- **private void EFDB_TextChanged(object sender, EventArgs e) :** Writes to Entered Frequency Entered Frequency Box display box
- **private void EDDB_TextChanged(object sender, EventArgs e)** : Writes to Entered Duty cycle in Entered Duty Cycle display box
- **private void MFDB_TextChanged(object sender, EventArgs e) :** Writes to Entered Frequency in Measured Frequency display box
- **private void MPWDB_TextChanged(object sender, EventArgs e) :** Writes to Entered Frequency in Measured Positive Width display box
- **private void MDCDB_TextChanged(object sender, EventArgs e) :** Writes to Entered Frequency Measured Duty Cycle display box



```
        //Note: SerialPort object operates on a seperate thread.
        //      Therefore, DataReceived event can not interact directly with other WinForm controls
        //      Doing so will cause cross-thread action exception.
        //      In order to display received data to UI, delegate method must be used.
        //Read data from serial port object
        private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {

            string data = serialPort1.ReadExisting();
            //check if the received data is number or not
            bool bNum = int.TryParse(data, out i);
            if (bNum)
            {
                switch (count)
                {
                    case 0:
                       //Display to user input in Frequency BOX
                        AddTextToLabelReadFreqbox(data);
                        break;
                    case 1:
                        //Display to user innput in DUTYCYCLE BOX
                        AddTextToLabelRead_duty(data);
                        break;
                    case 2:
                        //Display to MeasureFrequency BOX
                        //caputure and display
                        AddTextToLabelM_freq(data.ToString());
                        break;
                    case 3:
                        //Display to MeasurePulse BOX
                        //caputure and display
                        AddTextToLabelM_Pulse(data.ToString());
                        break;
                    case 4:
                        //Display to MeasureDUTY BOX
                        //caputure and display
                        AddTextToLabelM_Duty(data.ToString());
                        break;

                }

            }
            else
            {
                if (data != "a"){
                AddTextToLabel(data);
                }
             else if (data == "a")
            {
                if (count < 4)
                {
                    count++;
                    //MessageBox.Show(count.ToString());
                }
                else
                {
                    count = 0;
                }
            }
            //TODO: display received string to Textbox
        }
```

Figure 10: C# code to select box

In order, to select which box to print the information the count variable is used to check which box the application is currently writing. As soon as the function receives data from the serial port the function would check if the received data is number or string. If it's number then the box then the status of the box is not changed. As soon as the serial port receives special character ‘a’ then the count is increased thus moved to next display box. However, by default the function would write data in Work Space Display Box. In this way the display box is enabled and disabled.

4.4 UART/Bluetooth Connection

To connect the GUI application on PC serial communication is used. A serial interface which RS-232 is implemented to connect with a PC serial port. UART 1 from microcontroller is used for communication to serial port. RS232 with BUD rate of 115200 Parity bits of N, sends a byte per transaction with a stop bit of 1 and time out of 500 as shown in *figure 11* below.

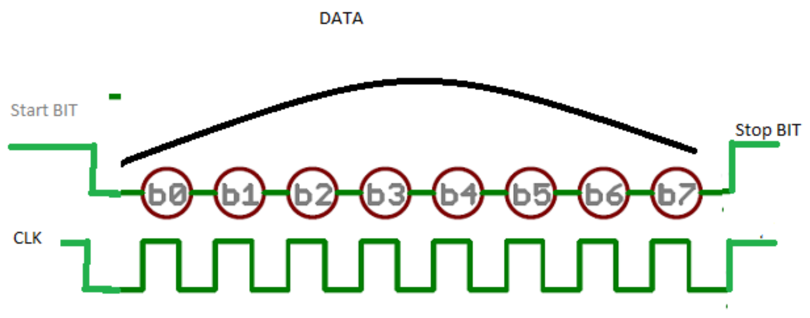
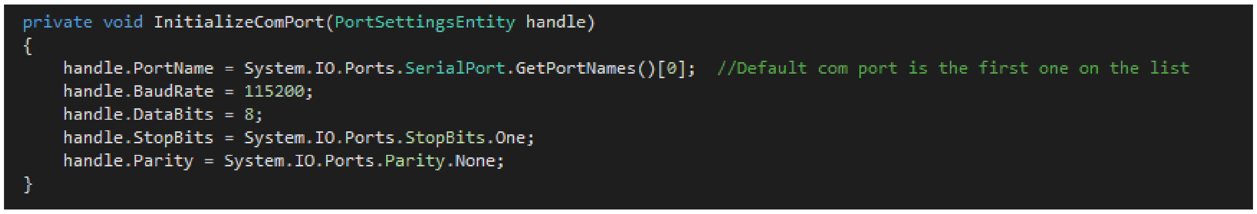


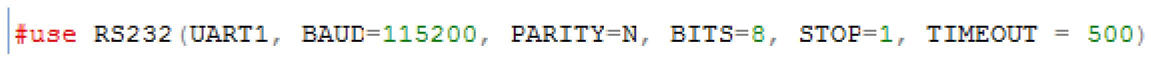
Figure 11: UART Communication

Thus a byte is sent with high logic bit for start and end bit for the UART communication with the transfer rate of 11520 byte/s.



```
private void InitializeComPort(PortSettingsEntity handle)
{
    handle.PortName = System.IO.Ports.SerialPort.GetPortNames()[0];  //Default com port is the first one on the list
    handle.BaudRate = 115200;
    handle.DataBits = 8;
    handle.StopBits = System.IO.Ports.StopBits.One;
    handle.Parity = System.IO.Ports.Parity.None;
}
```

Figure 12: C# code configuring the serial communication (GUI)



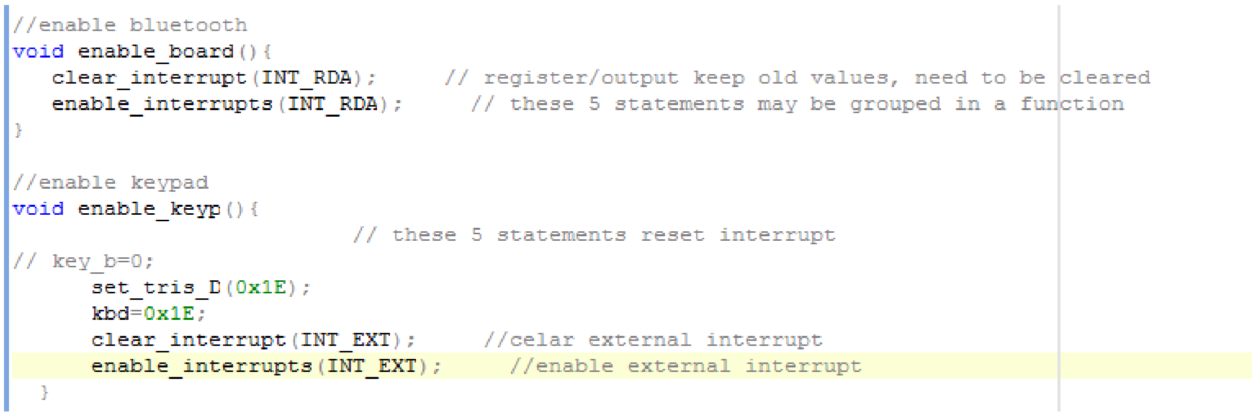
```
#use RS232(UART1, BAUD=115200, PARITY=N, BITS=8, STOP=1, TIMEOUT = 500)
```

Figure 13 : C code configuring the serial communication (Hardware)

4.5 PIC code

The code that is burnt in hard ware is written in C. The pic code follows the state machine as shown in **figure state diagram**. In this code there are altogether 2 external interrupts and 3 internal that makes the system complete. The interrupts with their function are described down below.

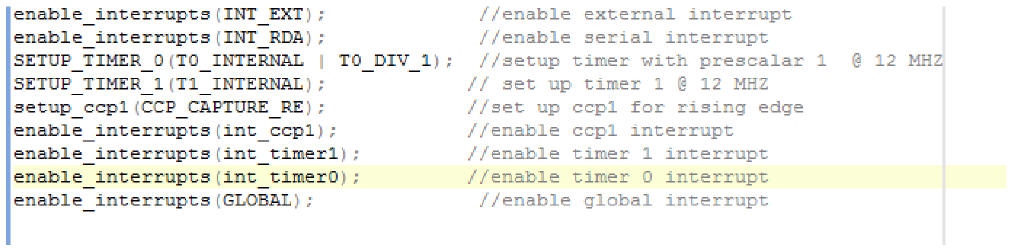
a. Interrupt setup



```
//enable bluetooth
void enable_board(){
   clear_interrupt(INT_RDA);      // register/output keep old values, need to be cleared
   enable_interrupts(INT_RDA);      // these 5 statements may be grouped in a function
}

//enable keypad
void enable_keyp(){
                              // these 5 statements reset interrupt
// key_b=0;
      set_tris_D(0x1E);
      kbd=0x1E;
      clear_interrupt(INT_EXT);      //celar external interrupt
      enable_interrupts(INT_EXT);      //enable external interrupt
  }
```

Figure 14 : C code in PIC to configure interrupt

These are the functions that enable the Bluetooth and keypad shown in *figure 14*. Also Interrupts are enabled in the main() of the C code. The *figure 15* down below shows the code the that was used to setup interrupts in the main() .



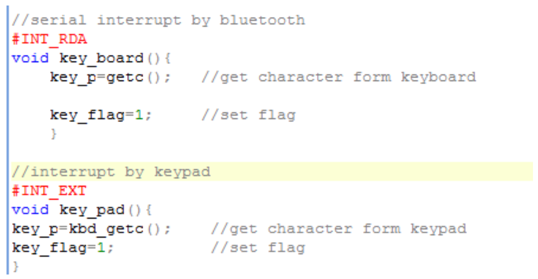
```
enable_interrupts(INT_EXT);              //enable external interrupt
enable_interrupts(INT_RDA);              //enable serial interrupt
SETUP_TIMER_0(T0_INTERNAL | T0_DIV_1);   //setup timer with prescalar 1  @ 12 MHZ
SETUP_TIMER_1(T1_INTERNAL);             // set up timer 1 @ 12 MHZ
setup_ccp1(CCP_CAPTURE_RE);             //set up ccp1 for rising edge
enable_interrupts(int_ccp1);            //enable ccp1 interrupt
enable_interrupts(int_timer1);          //enable timer 1 interrupt
enable_interrupts(int_timer0);          //enable timer 0 interrupt
enable_interrupts(GLOBAL);               //enable global interrupt
```

Figure 15 : Enabling interrupts in main

The *figure 15* is self-explanatory. The *figure 15* contradicts with the implement code which is provided at appendix on enabling timer 0. Timer 0 is only enabled when the user provides frequency with duty cycle, otherwise timer 0 is disabled until user enables to generate wave. Other than that everything needs to be enabled when the program runs. But in globe scope the function of the interrupts are defined

b. Interrupt functions

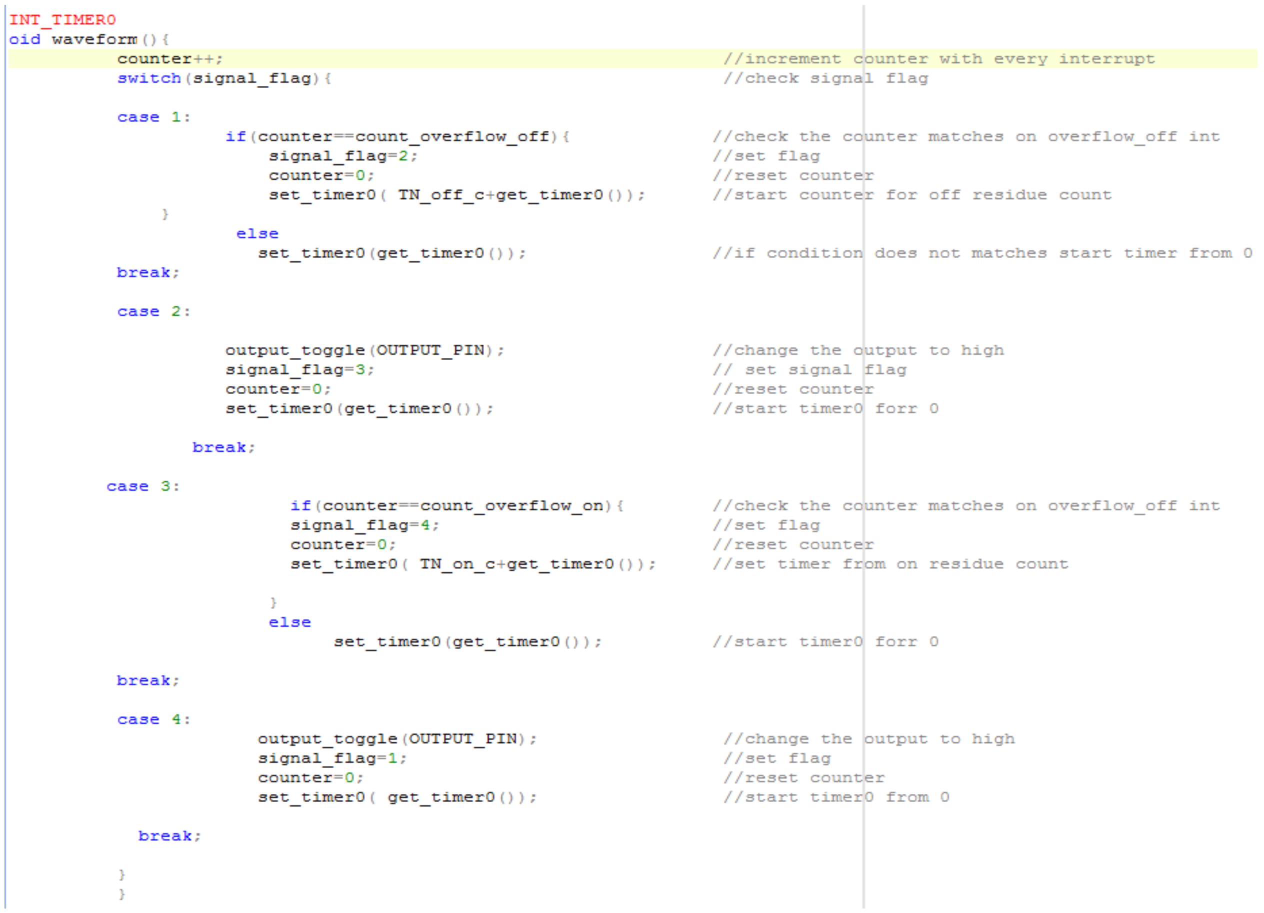
There are total of 5 functin defined for each inturrupts, which takes cares of the interrupt when the program runs in interrupt mode putting every thing on stack and executing the interrupt routine



```
//serial interrupt by bluetooth
#INT_RDA
void key_board(){
    key_p=getc();   //get character form keyboard

    key_flag=1;     //set flag
    }

//interrupt by keypad
#INT_EXT
void key_pad(){
key_p=kbd_getc();    //get character form keypad
key_flag=1;          //set flag
}
```

Figure 16: External Interrupt for user input

In *figure 16*, it can be seen that external interrupt is implemented to get serial input and keypad input. #INT_RDA and #INT_EXT are the serial and external interrupt respectively. As the program goes to these routine the value entered though the keypad or keyboard from GUI is key_p get the value from the user and set the key_flag as 1.



```
INT_TIMER0
oid waveform(){
          counter++;                                                  //increment counter with every interrupt
          switch(signal_flag){                                        //check signal flag

          case 1:
                    if(counter==count_overflow_off){                 //check the counter matches on overflow_off int
                        signal_flag=2;                               //set flag
                        counter=0;                                   //reset counter
                        set_timer0( TN_off_c+get_timer0());          //start counter for off residue count
              }
                     else
                       set_timer0(get_timer0());                     //if condition does not matches start timer from 0
          break;

          case 2:

                    output_toggle(OUTPUT_PIN);                       //change the output to high
                    signal_flag=3;                                   // set signal flag
                    counter=0;                                       //reset counter
                    set_timer0(get_timer0());                        //start timer0 forr 0

                           break;

         case 3:
                          if(counter==count_overflow_on){            //check the counter matches on overflow_off int
                          signal_flag=4;                             //set flag
                          counter=0;                                 //reset counter
                          set_timer0( TN_on_c+get_timer0());         //set timer from on residue count

                        }
                        else
                              set_timer0(get_timer0());              //start timer0 forr 0

          break;

          case 4:
                       output_toggle(OUTPUT_PIN);                     //change the output to high
                       signal_flag=1;                                 //set flag
                       counter=0;                                     //reset counter
                       set_timer0( get_timer0());                     //start timer0 from 0

            break;

          }
          }
```

Figure 17: Timer Interrupt for wave generation

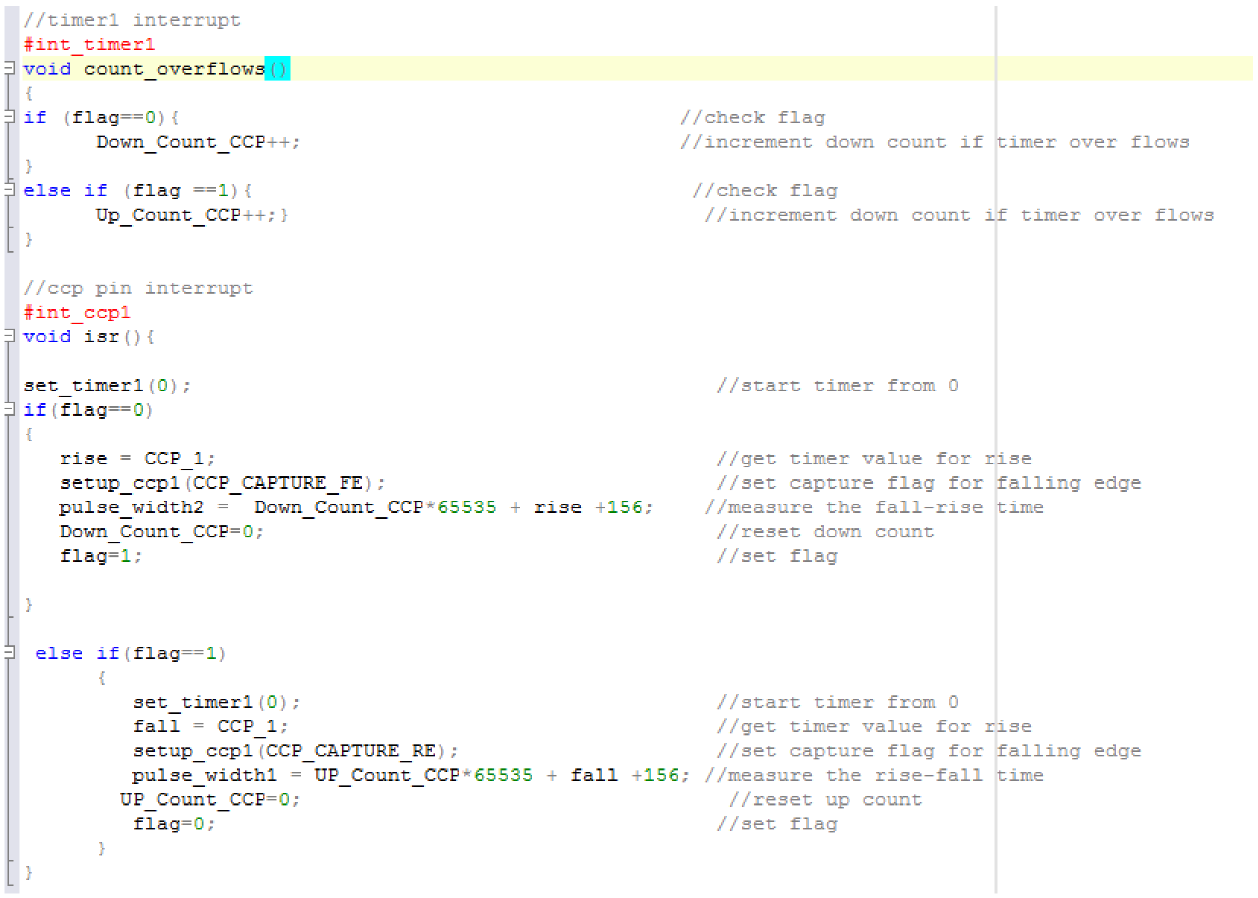
In *figure 17*, the timer interrupt #INT_TIMER0 is used to generate square wave which is internal interrupt. With every overflow in the timer0 the program enters in the waveform routine. With every interrupt in timer 0 the counter in waverform() routine increaments the counter by one. There are 4 states in the waveform module in which sate 1 and 2 controls the off signal and state 3 and 4 controls on sate of the signal. The functions of these sate are as follow.

- Sate1: This state controls how long the signal need to be off for. Here, if statement doesn't match the counter then the timer is set to start from 0 to 65535 and the counter is increased by 1 with every interrupt of internal timer

#int_Timer0 . And once it reaches the overflow number of the timer 0 then the interrupt is triggered and counter is increased. In this way the integer part of the overflow is compensated. Once the counter matches the if condition then the signal flag is set and counter is reset, finally moved to sate 2 with the start count of the residue remaining for off state.

- State 2: This state is reached when the counter in timer 0 overflows which was set form the residue time. Then the pin in the microcontroller is toggled to high, signal flag is set and the counter is reset again because counter was increment by one.
- State 3 : This state controls how long the signal needs to be turned on for. This state works exactly same as state 1 except the on residue and the counter value that needs to be compared is different value if the signal is not 50% duty cycle. Once the condition is matched the flag and the counter is set. Then the timer starts from the residue of the on part.
- State 4 : This state is reached after the timer goes in overflow mode form the residue part of the on part. Then the signal is to high and the counter is reset. Also the state is changed to 1 which will execute after interrupt occurs.

In this way the switch statement controls the duty cycle of the wave creating a square wave with logic high and low.



```
//timer1 interrupt
#int_timer1
void count_overflows()
{
if (flag==0){                                                      //check flag
      Down_Count_CCP++;                                            //increment down count if timer over flows
}
else if (flag ==1){                                                 //check flag
      Up_Count_CCP++;}                                               //increment down count if timer over flows
}

//ccp pin interrupt
#int_ccp1
void isr(){

set_timer1(0);                                                        //start timer from 0
if(flag==0)
{
   rise = CCP_1;                                                      //get timer value for rise
   setup_ccp1(CCP_CAPTURE_FE);                                        //set capture flag for falling edge
   pulse_width2 =  Down_Count_CCP*65535 + rise +156;                 //measure the fall-rise time
   Down_Count_CCP=0;                                                  //reset down count
   flag=1;                                                            //set flag

}

 else if(flag==1)
      {
         set_timer1(0);                                               //start timer from 0
         fall = CCP_1;                                                //get timer value for rise
         setup_ccp1(CCP_CAPTURE_RE);                                  //set capture flag for falling edge
         pulse_width1 = UP_Count_CCP*65535 + fall +156;  //measure the rise-fall time
        UP_Count_CCP=0;                                                //reset up count
         flag=0;                                                      //set flag
      }
}
```

Figure 18 : Time1 and CCP1 interrupt

In *figure 18* it can be timer 1 when timer 1 interrupts the routine verifies the flag . If the flag condition is satisfied then the Up_Count_CPP or Down_Count_CPP is increased by 1 with every timer when every timer 1 goes to interrupt. However, if there is no interrupt occurs there won't be no count. Timer 1 is used instead of other timers due to the fact that CCP1 pin only uses timer 1 which is given in PIC18f4455 data sheet.

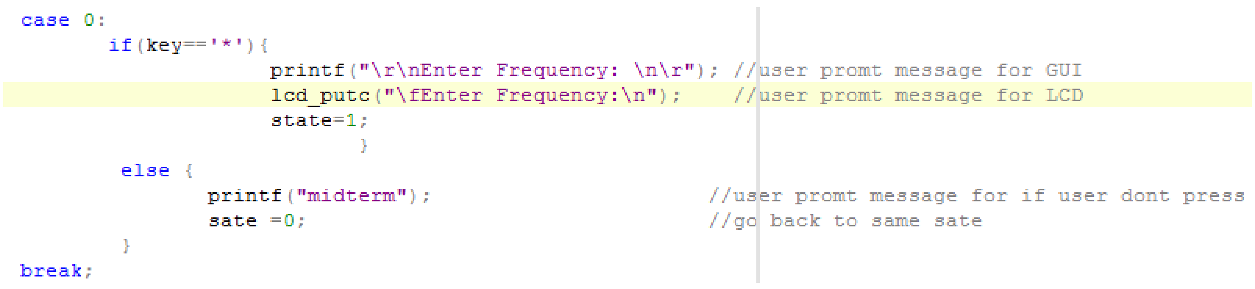
Under CCP1 interrupt the capturing of the pulse occurs under function named isr(). So, when every time the program enters to this routine timer 1 is enabled from 0. So, it will start counting from 0 until it reaches 65535 since, timer 1 is configured as 16 bits. Now the function check the flag condition. If the flag is equal to 0 then the variable named rise gets the value of the timer and sets up CCP for falling capture mode. Then the equation is used to calculate the number of counts before for falling edge. In the equation it can be seen that the following equation is used to capture the fall-rise time.

pulse_width2 = 65535* Down_Count_CPP + rise+156

This equation measures the total number of count the signal takes for before the signal goes low. But if there is no overflow in the counter of the timer1 then the value of Down_Count_CPP 0. So only riser+156 counts is calculated form the equation. However if there is over flow in the timer1 then the Down_Count_CPP hold the value which is the number of overflow before the signal goes low. Then the Down_Count_CPP is reset to 0 and flag is changed to 1. In this way fall-rise time is measured in terms of ticks which will be later converted to actual time in microseconds in the main() function. Similarly, rise-fall time is calculated.

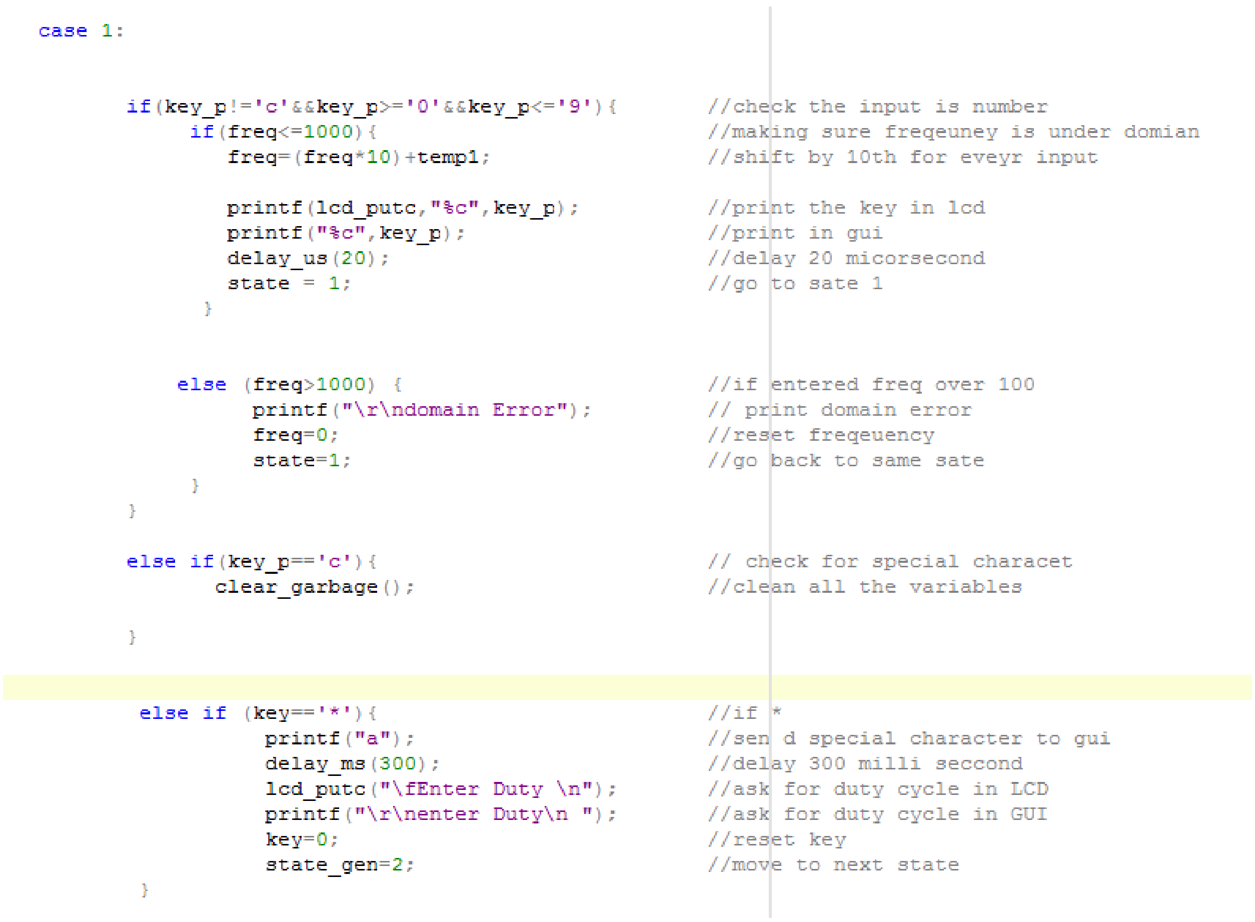
c. Main

The main initializes all the interrupts and all the data entered are processed to generate and capture the wave information. With every external interrupt the main() performs specific task with the states. State 1 takes the frequency, state 2 takes duty cycle input form the user sate 3 calculates the period, on duty time ,off duty time etc. State 4 captures the frequency of the wave, sate 5 measure the positive pulse of the wave and the last sate 6 measures the duty cycle of the measure wave. The explain of each sate with the code is shown down below.



```
case 0:
       if(key=='*'){
                    printf("\r\nEnter Frequency: \n\r"); //user promt message for GUI
                    lcd_putc("\fEnter Frequency:\n");    //user promt message for LCD
                    state=1;
                           }
        else {
                printf("midterm");                    //user promt message for if user dont press
                sate =0;                              //go back to same sate
        }
break;
```

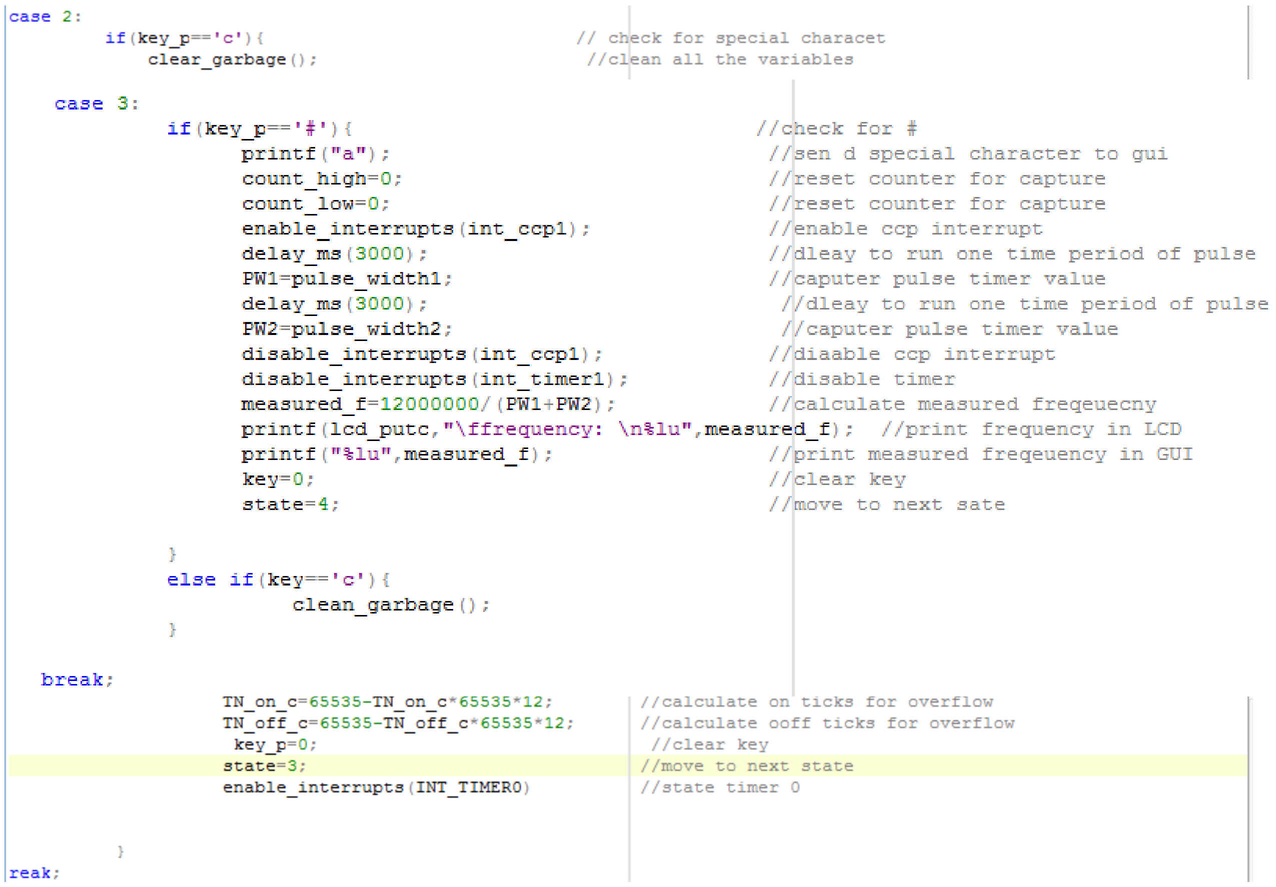
This state rpints the message to enter frequency. If the user press ‘*’ for key pad or GUI.



```
 case 1:


        if(key_p!='c'&&key_p>='0'&&key_p<='9'){        //check the input is number
             if(freq<=1000){                           //making sure freqeuney is under domian
                freq=(freq*10)+temp1;                  //shift by 10th for eveyr input

                printf(lcd_putc,"%c",key_p);           //print the key in lcd
                printf("%c",key_p);                    //print in gui
                delay_us(20);                          //delay 20 micorsecond
                state = 1;                             //go to sate 1
            }


            else  (freq>1000)  {                       //if entered freq over 100
                  printf("\r\ndomain Error");          // print domain error
                  freq=0;                              //reset freqeuency
                  state=1;                             //go back to same sate
             }
        }

        else if(key_p=='c'){                           // check for special charactet
                clear_garbage();                       //clean all the variables

        }


         else if  (key=='*'){                          //if *
                   printf("a");                        //sen d special character to gui
                   delay_ms(300);                      //delay 300 milli seccond
                   lcd_putc("\fEnter Duty \n");        //ask for duty cycle in LCD
                   printf("\r\nenter Duty\n ");        //ask for duty cycle in GUI
                   key=0;                              //reset key
                   state_gen=2;                        //move to next state
         }
```

Figure 19: Taking input form User (Frequency)

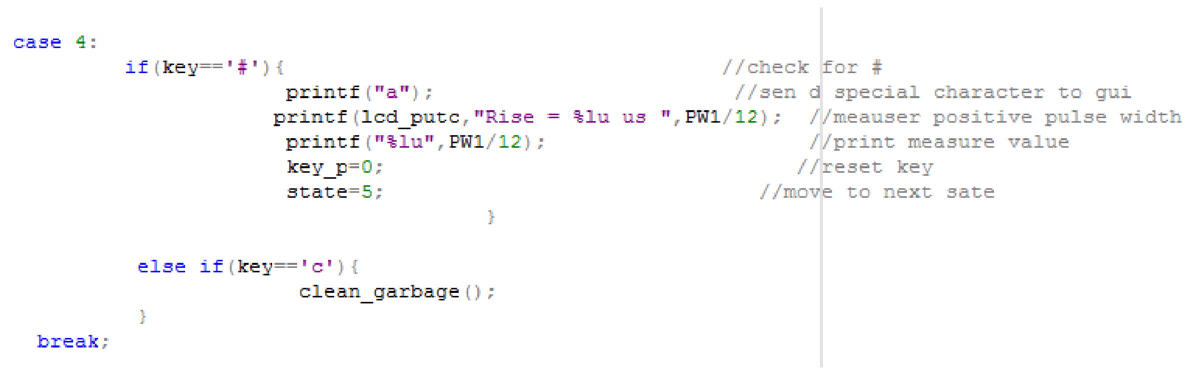
This sate takes the frequency input form the user. If the user presses ‘*’ then the state is change and display “enter duty “in LCD and GUI. If ’c’ then the variables are cleared and the sate remains the same.



```
case 2:
        if(key_p=='c'){                                    // check for special characet
            clear_garbage();                                //clean all the variables

    case 3:
            if(key_p=='#'){                                          //check for #
                  printf("a");                                        //sen d special character to gui
                  count_high=0;                                       //reset counter for capture
                  count_low=0;                                        //reset counter for capture
                  enable_interrupts(int_ccp1);                        //enable ccp interrupt
                  delay_ms(3000);                                     //dleay to run one time period of pulse
                  PW1=pulse_width1;                                   //caputer pulse timer value
                  delay_ms(3000);                                      //dleay to run one time period of pulse
                  PW2=pulse_width2;                                    //caputer pulse timer value
                  disable_interrupts(int_ccp1);                       //diaable ccp interrupt
                  disable_interrupts(int_timer1);                     //disable timer
                  measured_f=12000000/(PW1+PW2);                      //calculate measured freqeuecny
                  printf(lcd_putc,"\ffrequency: \n%lu",measured_f);  //print frequency in LCD
                  printf("%lu",measured_f);                           //print measured freqeuency in GUI
                  key=0;                                              //clear key
                  state=4;                                            //move to next sate

            }
            else if(key=='c'){
                      clean_garbage();
            }

   break;
                       TN_on_c=65535-TN_on_c*65535*12;       //calculate on ticks for overflow
                       TN_off_c=65535-TN_off_c*65535*12;     //calculate ooff ticks for overflow
                        key_p=0;                              //clear key
                       state=3;                              //move to next state
                       enable_interrupts(INT_TIMER0)         //state timer 0


          }
reak;
```

Figure 19: Taking input form User (Duty Cycle)

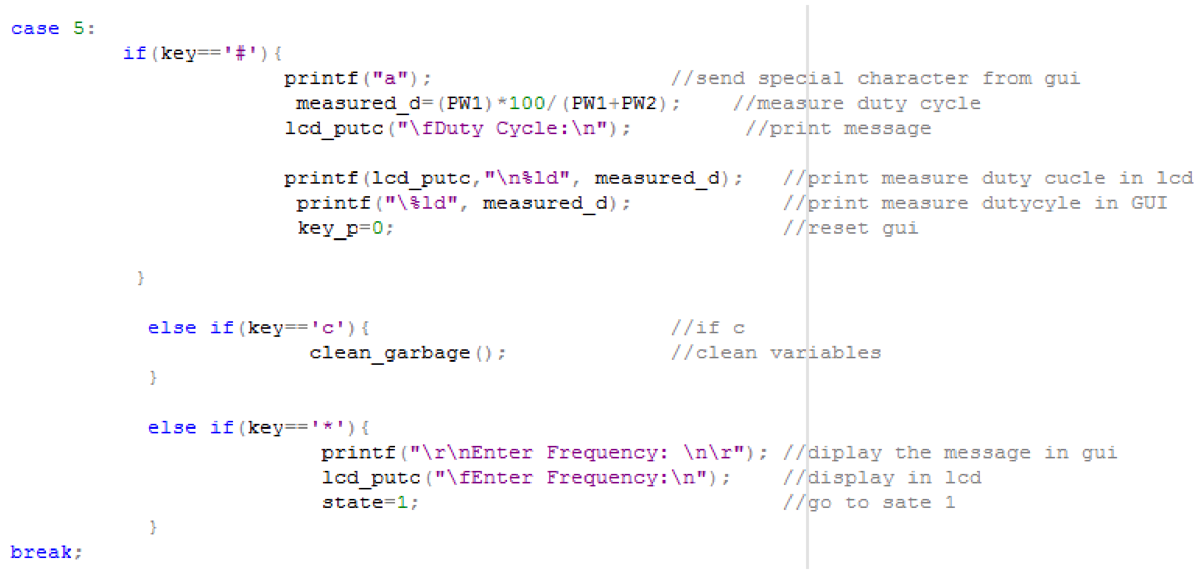
At this sate the duty cycle is entered taken form the user if the user presses number from 0 to 9 shown in *figure 19*. If the user press ‘c’ from GUI the variables are clear and if ‘*’ is pressed wave is generated by doing the calculation.This sate measure the captured frequency done by CCP pin and also display to LCD and GUI.



```
case 4:
         if(key=='#'){                                    //check for #
                     printf("a");                          //sen d special character to gui
                    printf(lcd_putc,"Rise = %lu us ",PW1/12);  //meauser positive pulse width
                     printf("%lu",PW1/12);                      //print measure value
                     key_p=0;                                  //reset key
                     state=5;                                //move to next sate
                                  }

          else if(key=='c'){
                     clean_garbage();
          }
  break;
```

Figure 20: Measuring positive Pulse

This sate calculated the captured Positive width of the pulse given to microcontroller.



```
case 5:
         if(key=='#'){
                     printf("a");                     //send special character from gui
                      measured_d=(PW1)*100/(PW1+PW2);     //measure duty cycle
                     lcd_putc("\fDuty Cycle:\n");          //print message

                     printf(lcd_putc,"\n%ld", measured_d);   //print measure duty cucle in lcd
                      printf("\%ld", measured_d);             //print measure dutycyle in GUI
                      key_p=0;                                //reset gui

          }

          else if(key=='c'){                    //if c
                      clean_garbage();              //clean variables
          }

          else if(key=='*'){
                       printf("\r\nEnter Frequency: \n\r"); //diplay the message in gui
                       lcd_putc("\fEnter Frequency:\n");    //display in lcd
                       state=1;                             //go to sate 1
          }
break;
```

Figure 21: Measuring captured Frequency

The last sate 5 measure the duty cycle of the given wave and displays it.

5 Results and Discussion

As specified the square wave needs to be generated form 1 Hz to 1 KHz with the duty cycle of 20% to 80 %. All the requirements were fulfilled except the pulse width measurement for the low frequency were not right due to the fact the counters were not cleared which caused the errors. The table below shows

Table 1:

Frequency Entered (Hz)	Frequency Observed(Hz)	Duty Cycle Entered %	Duty cycle observed	% Error for Duty Cycle	Measured Pulse width(ms)	Actual pulse width(ms	% Error Measure Width	% ERROR for Frequency
1	1.0238	80%	80.60%	0.75%	wrong	800	Wrong	0.0238
1	1.0501	50%	50%	0.30%	wrong	500	Wrong	0.0501
1	1.0172	20%	20.10%	0.50%	wrong	200	wrong	0.0172
100	100.26	33%	33.18%	0.55%	329	330	0.303030303	0.0026
1000	1005.41	70%	70.30%	0.43%	696	700	0.571428571	0.00541

Table 1 shows the Expected value and observed values. As can be seen that except low Frequency measurement the requirements are fulfilled. Also, it can be observed that if the duty cycle of the wave was increased than the error percentage went up for generation.

6 Conclusion

In conclusion this project clarified the relation between counters and timers. Using the relation between them specific task could be achieved. For this project square wave was created with logic high and logic low which was controlled by the timers and counters. Also, this project gave clear understanding between internal and external interrupts and the implementation of it on microcontroller. Moreover, serial communication between two modules was interesting and fun to learn for GUI application.

Appendix

```
/*Author: ashish Khadka
ECE 422 Midterm
SQuare wave generator
ver_1/0*/
#include <18f4455.h>
#FUSES NOWDT //No Watch Dog Timer
#FUSES NOPROTECT //Code not protected from reading
#FUSES NOBROWNOUT //Reset when brownout detected
#FUSES PUT //No Power Up Timer
#FUSES NODEBUG //No Debug mode for ICD
#FUSES NOWRT //Program memory not write protected
#FUSES NOLVP //No Low-Voltage ICSP Programming
#FUSES HSPLL // High-Speed Crystal with PLL enabled
#FUSES PLL5 //PLL prescaler set to 5
#FUSES CPUDIV1 //no postscaler
#use delay(clock = 48Mhz, crystal = 20Mhz)
#use RS232(UART1, BAUD = 115200, PARITY = N, BITS = 8, STOP = 1, TIMEOUT = 500)

#define OUTPUT_PIN PIN_A3
#include "kbd_yz.c"
#include <lcd.c>

//declare variables
char key_p; Int32  on_time_us;//=0;
int key_flag = 0;
float freq = 0; float period = 0;
Int16 TN, TN_O, TN_on_c, TN_off_c;  int signal_flag = 1; float on_c, off_c = 0;
int counter = 0;
int32 count_overflow_on, count_overflow_off = 0;

#INT_EXT
void key_pad(){
      key_p = kbd_getc();                                  //get character from key board
      key_flag = 1;                                        //set flag
}

#INT_RDA
void key_board(){
      key_p = getc();                                      //get character from key board
      key_flag = 1;                                        //set flag
}

#INT_TIMER0
void waveform(){
      counter++;                                           //increment counter with every interrupt
      switch (signal_flag){                                //check signal flag

```

```
	case 1:
		if (counter == count_overflow_off){		//check the counter matches on overflow_off int
			signal_flag = 2;					//set flag
			counter = 0;						//reset counter
			set_timer0(TN_off_c + get_timer0());	//start counter for off residue count
		}
		else
			set_timer0(get_timer0());			//if condition does not matches start timer from 0
		break;

	case 2:

		output_toggle(OUTPUT_PIN);				//change the output to high
		signal_flag = 3;						// set signal flag
		counter = 0;							//reset counter
		set_timer0(get_timer0());				//start timer0 forr 0

		break;

	case 3:
		if (counter == count_overflow_on){		//check the counter matches on overflow_off int
			signal_flag = 4;					//set flag
			counter = 0;						//reset counter
			set_timer0(TN_on_c + get_timer0());	//set timer from on residue count

		}
		else
			set_timer0(get_timer0());		//start timer0 forr 0

		break;

	case 4:
		output_toggle(OUTPUT_PIN);			//change the output to high
		signal_flag = 1;					//set flag
		counter = 0;						//reset counter
		set_timer0(get_timer0());			//start timer0 from 0

		break;

	}
}
}



//ccp pin interrupt
#int_ccp1
void isr(){

    set_timer1(0);                                          //start timer from 0
    if (flag == 0)
    {
                rise = CCP_1;                                       //get timer value for rise
                setup_ccp1(CCP_CAPTURE_FE);                         //set capture flag for falling edge
                pulse_width2 = Down_Count_CCP * 65535 + rise + 156; //measure the fall-rise time
                Down_Count_CCP = 0;                                 //reset down count
                flag = 1;                                            //set flag

        }

        else if (flag == 1)
        {
                set_timer1(0);                                  //start timer from 0
                fall = CCP_1;                                   //get timer value for rise
                setup_ccp1(CCP_CAPTURE_RE);                     //set capture flag for falling edge
                pulse_width1 = UP_Count_CCP * 65535 + fall + 156; //measure the rise-fall time
                UP_Count_CCP = 0;                                   //reset up count
                flag = 0;                                         //set flag
        }
    }


    void clear_garbage(){
        key_p = 0;                          //clear key
        key_flag = 0;                       // clear flag
        freq = 0;                           //clear frequency
        state = 0;                          // change sate to 0
        on_time_us = 0;                     //clear on duty
        output_low(OUTPUT_PIN);           //set output low
        printf("Reset performed\n");      //print message
        lcd_putc('\f');                   //clear lcd
        printf(lcd_putc, "Enter frequency \n"); //print in lcd
        delay_us(20);                        //delay 20 microsecond
        disable_interrupts(INT_TIMER0);      //disable timero interrupt
        signal_flag = 0;                       //clear flag

    }



    void enable_board();

    void enable_keyp();


```

```

void main(){
       Int16 temp = 0;
       Int16 temp1;//=0;
       output_low(OUTPUT_PIN);
       lcd_init();
       kbd_init();
       int duty_c = 0;
       int state = 0;
       enable_keyp();
       enable_board();
       lcd_putc("Midterm");

       enable_interrupts(INT_EXT);              //enable external interrupt
       enable_interrupts(INT_RDA);              //enable serial interrupt
       SETUP_TIMER_0(T0_INTERNAL | T0_DIV_1);  //setup timer with prescalar 1  @ 12 MHZ
       SETUP_TIMER_1(T1_INTERNAL);             // set up timer 1 @ 12 MHZ
       setup_ccp1(CCP_CAPTURE_RE);             //set up ccp1 for rising edge
       enable_interrupts(int_ccp1);            //enable ccp1 interrupt
       enable_interrupts(int_timer1);          //enable timer 1 interrupt
       enable_interrupts(int_timer0);          //enable timer 0 interrupt
       enable_interrupts(GLOBAL);               //enable global interrupt




       //  printf(lcd_putc,"\fMidterm \n");
       while (true){

              if (key_p != 0 && key_flag == 1){
                     disable_interrupts(INT_EXT);

                     temp1 = key_p - '0'; //convert char to int
                     key_flag = 0;      //disable flag


                     switch (state){

                     case 0:
                            if (key == '*'){
                                   printf("\r\nEnter Frequency: \n\r"); //user promt message for GUI
                                   lcd_putc("\fEnter Frequency:\n"); //user promt message for LCD
                                   state = 1;
                            }
                            else {
                                   printf("midterm"); //user promt message for if user dont press *
                                   sate = 0; //go back to same sate
                            }
                            break;

                     case 1:

```

```

    if (key_p != 'c'&&key_p >= '0'&&key_p <= '9'){    //check the input is number
        if (freq <= 1000){    //making sure freqeuney is under domian
            freq = (freq * 10) + temp1;    //shift by 10th for eveyr input

            printf(lcd_putc, "%c", key_p);    //print the key in lcd
            printf("%c", key_p);    //print in gui
            delay_us(20);    //delay 20 micorsecond
            state = 1;    //go to sate 1
        }


        else (freq>1000) {    //if entered freq over 100
            printf("\r\ndomain Error");    // print domain error
            freq = 0;    //reset freqeuency
            state = 1;    //go back to same sate
        }
    }

    else if (key_p == 'c'){    // check for special characet
        clear_garbage();    //clean all the variables

    }


    else if (key == '*'){    //if *
        printf("a");                   //send special character to gui
        delay_ms(300);    //delay 300 milli seccond
        lcd_putc("\fEnter Duty \n");       //ask for duty cycle in LCD
        printf("\r\nenter Duty\n ");       //ask for duty cycle in GUI
        key = 0;    //reset key
        state_gen = 2;    //move to next state
    }

    break;

case 2:
```

```
            if (key_p == 'c'){ // check for special characet
                clear_garbage(); //clean all the variables

            }

            else if (key_p != 'c'&&key_p >= '0'&&key_p <= '9'){ //check the input is number
                temp1 = key_p - '0'; //convert char to int
                if (duty_c<100){ //check if duty clcue is less than 100%
                    duty_c = (duty_c * 10) + temp1; //shift by 10th
                    printf(lcd_putc, "%c", key_p); //print entered key in lcd
                    delay_us(20); //delay to debounce
                    state = 2; //move to next sate
                }
            }
            else if (key_p == '*'){ //if *
                printf("\r\nGenerating pulse"); //print message
                lcd_putc("\fGenerating pulse"); //print message
                delay_us(20); //delay 20 microseocnd
                period = 1000000 / freq; //converting into microsecond
                on_time_us = (float)duty_c*(period) //100;//on time duty cycle in microsecond
                    on_c = duty_c*(period) / 100;; // get on duty time
                TN = 65535 - on_time_us * 12 + 12; //count ticks to keep the signal on
                TN_O = 65535 - (period - on_time_us) * 12 + 12; //off time ticks
                count_overflow_on = (on_time_us) / 5461.25; //count number of overflows in integeres if there is one integer
                count_overflow_off = (period - on_time_us) / 5461.25; //count number of over flow for off duty gives integer
                on_c = (on_c) / 5461.25; //calculate on residue
                off_c = (period - on_time_us) / 5461.25; //calculate off residue
                on_c = on_c - (long)on_c; //getting float for residue for on
                off_c = off_c - (long)off_c; //getting float for residue for off
                TN_on_c = on_c;
                TN_off_c = off_c;
                TN_on_c = 65535 - TN_on_c * 65535 * 12; //calculate on ticks for overflow
```

```
                    TN_off_c = 65535 - TN_off_c * 65535 * 12;    //calculate ooff ticks for overflow
                    key_p = 0;    //clear key
                    state = 3;    //move to next state
                    enable_interrupts(INT_TIMER0)    //state timer 0


                }
                break;


            case 3:
                if (key_p == '#'){    //check for #
                    printf("a");    //sen d special character to gui
                    count_high = 0;    //reset counter for capture
                    count_low = 0;    //reset counter for capture
                    enable_interrupts(int_ccp1);    //enable ccp interrupt
                    delay_ms(3000);    //dleay to run one time period of pulse
                    PW1 = pulse_width1;    //caputer pulse timer value
                    delay_ms(3000);    //dleay to run one time period of pulse
                    PW2 = pulse_width2;    //caputer pulse timer value
                    disable_interrupts(int_ccp1);    //diaable ccp interrupt
                    disable_interrupts(int_timer1);    //disable timer
                    measured_f = 12000000 / (PW1 + PW2);    //calculate measured freqeuecny
                    printf(lcd_putc, "\ffrequency: \n%lu", measured_f);  //print frequency in LCD
                    printf("%lu", measured_f);    //print measured freqeuency in GUI
                    key = 0;    //clear key
                    state = 4;    //move to next sate

                }
                else if (key == 'c'){
                    clean_garbage();
                }

                break;
            }


```

```

        case 4:
            if (key == '#'){                                //check for #
                printf("a");                        //sen d special character to gui
                printf(lcd_putc, "Rise = %lu us ", PW1 / 12);  //meauser positive pulse width
                printf("%lu", PW1 / 12);                    //print measure value
                key_p = 0;                                  //reset key
                state = 5;                                  //move to next sate
            }

            else if (key == 'c'){
                clean_garbage();
            }
            break;


        case 5:
            if (key == '#'){
                printf("a");                        //send special character from gui
                measured_d = (PW1)* 100 / (PW1 + PW2);     //measure duty cycle
                lcd_putc("\fDuty Cycle:\n");         //print message

                printf(lcd_putc, "\n%ld", measured_d);   //print measure duty cucle in lcd
                printf("\%ld", measured_d);            //print measure dutycyle in GUI
                key_p = 0;                                  //reset gui

            }

            else if (key == 'c'){                           //if c
                clean_garbage();               //clean variables
            }

            else if (key == '*'){
                printf("\r\nEnter Frequency: \n\r"); //diplay the message in gui
                lcd_putc("\fEnter Frequency:\n");    //display in lcd
                state = 1;                                 //go to sate 1
            }
            break;

            }
            enable_keyp();

        }
    }

    //enable bluetooth
    void enable_board(){
        clear_interrupt(INT_RDA);     // register/output keep old values, need to be cleared
        enable_interrupts(INT_RDA);     // these 5 statements may be grouped in a function
 }

    //enable keypad
    void enable_keyp(){
        // these 5 statements reset interrupt
        // key_b=0;
        set_tris_D(0x1E);
        kbd = 0x1E;
        clear_interrupt(INT_EXT);     //celar external interrupt
        enable_interrupts(INT_EXT);     //enable external interrupt
            }
```

```
/*c# CODE
 Ashish Khadka */
using System;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace Calculator_App
{
    public partial class FormMain : Form
    {
        int first_num = 0;

        int count = 0;

        int i=0;

        private FormPortSettings _formPortSettings;
        private PortSettingsEntity _defaultSettings, _currentSettings;

        public FormMain()
        {
            InitializeComponent();
            tbDisplay.ReadOnly = true;
        }

        private void InitializeComPort(PortSettingsEntity handle)
        {
            handle.PortName = System.IO.Ports.SerialPort.GetPortNames()[0];  //Default
com port is the first one on the list
            handle.BaudRate = 115200;
            handle.DataBits = 8;
            handle.StopBits = System.IO.Ports.StopBits.One;
            handle.Parity = System.IO.Ports.Parity.None;
        }



        private void connectToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //TODO: Implement serial port connect
            //error shoul dpoint here
            if (!serialPort1.IsOpen)
            {

serialPort1.Open();
                disconnectToolStripMenuItem.Enabled = true;
                connectToolStripMenuItem.Enabled = false;
            }
        }
        private void disconnectToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                //Disconnect serial port
                serialPort1.Close();
                connectToolStripMenuItem.Enabled = true;
                disconnectToolStripMenuItem.Enabled = false;
                tbDisplay.ReadOnly = true;
            }
        }

        private void FormMain_Load(object sender, EventArgs e)
        {
            _defaultSettings = new PortSettingsEntity();  //Object instance containing 
default settings
            _currentSettings = new PortSettingsEntity();  //Object instance storing 
current settings from user

            InitializeComPort(_defaultSettings);
            InitializeComPort(_currentSettings);

            //Initialize UI visual elements
            connectToolStripMenuItem.Enabled = true;
            disconnectToolStripMenuItem.Enabled = false;
        }

        private void FormSerialTx_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Close();
            }
        }



        //connect to seiral port
        private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (_formPortSettings == null)
            {
                _formPortSettings = new FormPortSettings();
            }

            _formPortSettings.Initialize(_defaultSettings, _currentSettings);
            if (_formPortSettings.ShowDialog() == DialogResult.OK)
            {
                //apply settings
                _currentSettings = _formPortSettings.NewSettings;
            }
        }

        //send reset message to hardware
        private void BtnClear_Click(object sender, EventArgs e)
        {
            tbDisplay.Clear();
            EFDB.Clear();
            EDDB.Clear(); MFDB.Clear(); MPWDB.Clear(); MDCDB.Clear(); count = 0;
            serialPort1.Write("c");

        }
        //send number to hardware and display on text box1
        private void button_click(object sender, EventArgs e)
        {
            //printing on screen when serial port is on
            Button button = (Button)sender;


            if (serialPort1.IsOpen)
            {

               // serialPort1.Write(button.Text);


                tbDisplay.Text = tbDisplay.Text + button.Text;


                first_num = Convert.ToInt32(button.Text);
                serialPort1.Write(button.Text);

            }
          }


        //if hash or astrick is pressed send to hardware and
        private void operator_click(object sender, EventArgs e)
        {
            Button button = (Button)sender;
            //op = Convert.ToChar(tbDisplay.Text);

            if (serialPort1.IsOpen)
            {
                serialPort1.Write(button.Text);

            }
        }

        //Reading from Serial port and display  work display box
        public delegate void AddnewText(string str);
        public void AddTextToLabel(string str)
        {
            if (this.tbDisplay.InvokeRequired)
            {
                this.Invoke(new AddnewText(AddTextToLabel), str);
            }
            else
            {
                this.tbDisplay.Text = "";
                this.tbDisplay.Text += str;
            }
        }

        private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs
e)
        {
            //Note: SerialPort object operates on a seperate thread.
            //      Therefore, DataReceived event can not interact directly with other
WinForm controls.
```

```
            //      Doing so will cause cross-thread action exception.
            //      In order to display received data to UI, delegate method must be
used.

            //Read data from serial port object
            string data = serialPort1.ReadExisting();
            //check if the received data is number or not
            bool bNum = int.TryParse(data, out i);
            if (bNum)
            {
                switch (count)
                {
                    case 0:
                       // count++;
                        AddTextToLabelReadFreqbox(data);
                        break;
                    case 1:
                       // count++;
                        AddTextToLabelRead_duty(data);
                        break;
                    case 2:
                      //  count++;
                        AddTextToLabelM_freq(data.ToString());
                        break;
                    case 3:
                        //count++;
                        AddTextToLabelM_Pulse(data.ToString());
                        break;
                    case 4:
                       // count++;
                        AddTextToLabelM_Duty(data.ToString());
                        break;
                    case 5:
                       // count++;
                        AddTextToLabelReadFreqbox(data.ToString());
                        break;
                }

            }
            else
            {
                if (data != "a"){
                AddTextToLabel(data);
                }
             else if (data == "a")
            {

                count++;
                MessageBox.Show(count.ToString());
            }
            //TODO: display received string to Textbox
        }
    }

        public delegate void AddnewText1(string readFreq);
        public void AddTextToLabelReadFreqbox(string readFreq)
        {
```

```
    if (this.EFDB.InvokeRequired)
    {
        this.Invoke(new AddnewText1(AddTextToLabelReadFreqbox), readFreq);
    }
    else
    {
        //this.EFDB.Text = "";
        this.EFDB.Text += readFreq;
    }
}



//Reading from Serial port and display entered frequency display box
private void EFDB_TextChanged(object sender, EventArgs e)
{
    string freq_read = serialPort1.ReadExisting();
    AddTextToLabel(freq_read);
}



public delegate void AddnewText2(string readDuty);
public void AddTextToLabelRead_duty(string readDuty)
{
    if (this.EDDB.InvokeRequired)
    {
        this.Invoke(new AddnewText2(AddTextToLabelRead_duty), readDuty);
    }
    else
    {
        //this.EFDB.Text = "";
        this.EDDB.Text += readDuty;
    }
}


private void EDDB_TextChanged(object sender, EventArgs e)
{
    string e_duty_c = serialPort1.ReadExisting();
    AddTextToLabelRead_duty(e_duty_c);

}






public delegate void AddnewText3(string M_freq);
public void AddTextToLabelM_freq(string M_freq)
{
    if (this.MFDB.InvokeRequired)
    {
        this.Invoke(new AddnewText3(AddTextToLabelM_freq), M_freq);
    }
    else
    {
        //this.EFDB.Text = "";
        this.MFDB.Text += M_freq;
    }
}



private void MFDB_TextChanged(object sender, EventArgs e)
{
    string measured_f = serialPort1.ReadExisting();
    AddTextToLabelM_freq(measured_f);

}




public delegate void AddnewText4(string M_pulse);
public void AddTextToLabelM_Pulse(string M_pulse)
{
    if (this.MPWDB.InvokeRequired)
    {
        this.Invoke(new AddnewText4(AddTextToLabelM_Pulse), M_pulse);
    }
    else
    {
        //this.EFDB.Text = "";
        this.MPWDB.Text += M_pulse;
    }
}

private void MPWDB_TextChanged(object sender, EventArgs e)
{
    string measured_pw = serialPort1.ReadExisting();
    AddTextToLabelM_Pulse(measured_pw);
}



public delegate void AddnewText5(string M_duty);
public void AddTextToLabelM_Duty(string M_duty)
{
    if (this.MDCDB.InvokeRequired)
    {
        this.Invoke(new AddnewText5(AddTextToLabelM_Duty), M_duty);
    }
    else
    {
        //this.EFDB.Text = "";
        this.MDCDB.Text += M_duty;
    }
}


 private void MDCDB_TextChanged(object sender, EventArgs e)
```

```
     {
         string measured_dc = serialPort1.ReadExisting();
         AddTextToLabelM_Duty(measured_dc);
     }



}

   }
```