Designing the Twenty Dollar Oscilloscope

Stephen Sieb
University of Utah
50 S. Central Campus Dr.
Salt Lake City, UT 84112

***Abstract*-In an effort to make teaching engineering concepts easier to students K-12 an inexpensive oscilloscope which has simplified design and functionality is being designed. The oscilloscope replaces current methods of teaching circuit analysis in the classroom. The oscilloscope in design fits in the palm of the hand, is self powered, powers a circuit, makes simple measurements, and performs various calculations. The oscilloscopes functionality features the ability to calculate amplitude, frequency, and DC operating point. The oscilloscope also makes 20 measurements at either 10 ms and 1 ms intervals then displays these values on an LCD display. The end result of the design is to allow teachers to teach math and engineering principles in the classroom without breaking their budgets.**

I. Introduction

This report gives an overview of building a prototype oscilloscope which can be mass produced for under $20 for student use in the classroom.

Currently for students to learn the principals of electrical engineering in the classroom costs teachers several thousand dollars in oscilloscopes and probes. These oscilloscopes come loaded with features that are only used in high-end applications by advanced users. They can make measurements in the range of microvolts, and picoamps. Students do not need all the features of current oscilloscopes, but due to a lack of options, teachers still are required to pay these premium prices if they desire to have an oscilloscope. Oscilloscopes on the market today range from $350 to $8000, and with 35 kids in a class even sharing oscilloscopes would greatly exceed any teacher's budget.

The design of the oscilloscope in this project reduces costs by eliminating the unnecessary functions of the oscilloscope thereby simplifying the interface. The Oscilloscope features only 3 buttons. The first button of the oscilloscope runs the circuit and makes the measurements. The second button steps through a list of these measured voltages. The third button outputs the frequency and the amplitude of the measured wave. Each press of these buttons advances through the associated list of measurements tied to that button.

The oscilloscope also acts as a signal generator and a power source. The oscilloscope runs off of one nine-volt battery that powers the microcontroller, buttons, and display. The battery is also used as the rail voltages (+4 and -5 V) for the circuits students design. Metal posts on the oscilloscope allow students to hook their circuits to these rail voltages allowing students to power their circuit directly from the oscilloscope. A third post provides an AC sin wave operating at 10 Hz to students for use with their circuit.

The interface has been kept as simple and as uncluttered as possible so that students using the oscilloscope as early as kindergarten will still be able to benefit from the oscilloscope when they are

seniors in high school. The designed oscilloscope fits into its own niche in the world of electronic devices, and has a very strong potential market. The oscilloscope is not bound for use solely in the classroom, but could also have applications in factories, laboratories, and other places where only simple measurements are needed and cost is a factor.

II. Methods

This section details the steps in designing the oscilloscope, time required, parts used, circuit schematic, and code used to program the microcontroller.

The following is a breakdown of the steps involved in building the prototype for the oscilloscope:

1) Research and Proposal: (inches)*:* The first step in designing the inexpensive oscilloscope is to see what devices there are currently on the market, check their functionality, and check their cost. Upon completion of this research a proposal must be sent to the engineering department, the acceptance of which marks the beginning of the oscilloscope design.

2) Design and Parts: To meet project specifications the oscilloscope includes all the aforementioned features, but the parts available for construction limit the functionality of the oscilloscope. Reducing the price of the oscilloscope also limits the parts available further narrowing the functions the oscilloscope can perform. A list of parts is included after this section.

3) Test Parts and Boot Up Programmable Integrated Circuit (PIC): To begin the construction of the oscilloscope all parts are ordered from Digikey as well as Microchip and then tested. The main testing of parts involves checking to see if the PIC can provide enough current to supply power to drive the liquid crystal display (LCD), checking whether the Step up converter provides enough current to drive the PIC, and checking the input resistances of the PIC.

4) Programming the PIC: The bulk of time and effort in engineering the oscilloscpe is programming the PIC. The PIC is programmed using the MPLab software which came included with the PIC Debug2 kit. This software allows code written in C to be uploaded to the PIC device. Code uploaded to the PIC needs to recognize button presses from the user and perform the desired operation. The code needs to output a 10 Hz square wave, read data from the students circuit into memory and save those values to be output to the LCD display later. The PIC acts as the brain of the oscilloscope.

5) Construction of the Circuit: The circuit needs first to power up the circuit using the chosen slider switch. Closing the slider switch connects the battery to the 5V step down converter. The 5V step down converters output is used as the ground for the circuit, while the terminals of the battery are used as the rail voltages of the circuit. The circuitry also provides protection to the PIC by using blocking diodes and op-amps with negative feedback loops. The circuitry inside the oscilloscope protects the PIC and reduces current by placing resistances to the inputs of the ACD and buttons. Finally, the circuitry takes the output square wave from the PIC runs it through a low pass filter to produce the desired sin wave. Wire

wrapping is the means of construction of the prototype circuit, but ultimately the oscilloscope places circuit components on a printed circuit board (PCB).

6) Testing: To insure the oscilloscope is functioning properly all possible button combinations must be pressed with various circuits hooked to both the outputs and inputs of the oscilloscope. Comparisons must be made to actual oscilloscope measurements and theoretical measurements compared to design measurements to insure accuracy of the design. The circuit must also be tested by novice users to ensure simplistic function and design.

7) Presentation: Before being released to the public, a plastic casing must be designed to enclose the circuit. The oscilloscope must also meet approval of those who have requested the design to ensure it has met all required specifications.

8) Future Work: While not currently in the design specifications, many uses for the oscilloscope could still be added. The easiest addition would be to add fourth button to the oscilloscope which when pressed would cause the circuit to act as a digital voltmeter. The most beneficial tool to students and teachers would be to allow the oscilloscope to transmit data to a graphing calculator or laptop for visual display of recorded values, and to make calculations on recorded data easier. It was also proposed that a phase delay button might be added to the circuit with a step function.

A grant has been given in the amount of $500 for the research and design of the oscilloscope. The total cost of the project is much less than this amount. This includes design, development, and testing. The goal of the project is to keep the actual oscilloscope under $20.

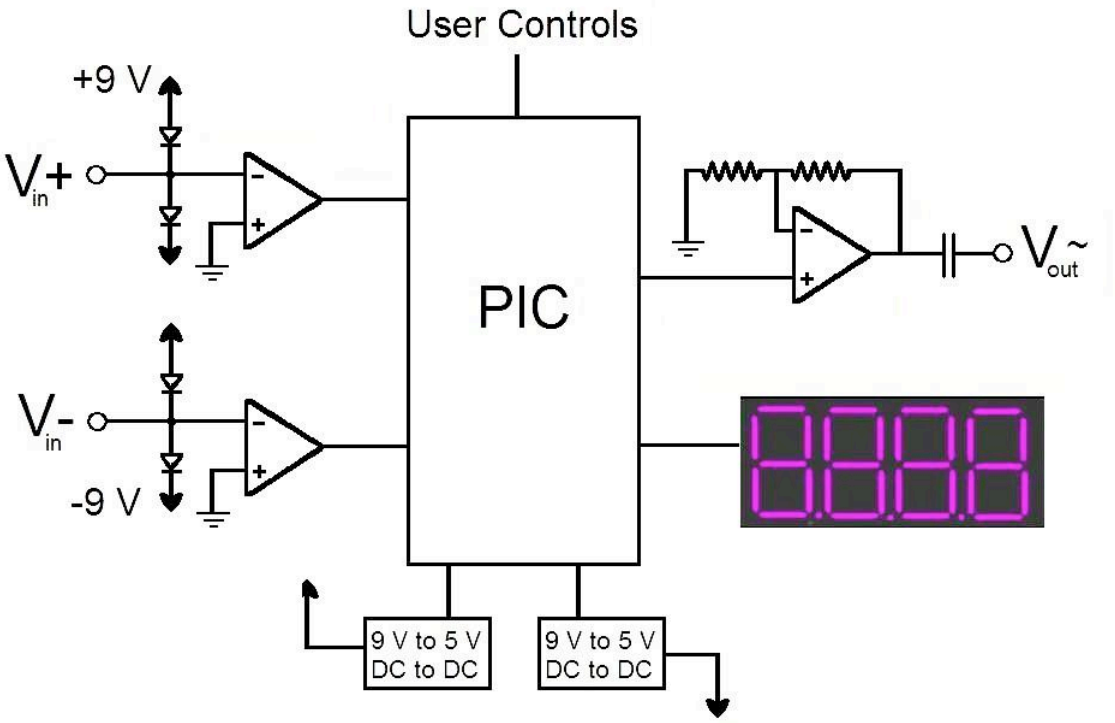
The cost of parts in the oscilloscope as well as other project costs is given in the table below:

TABLE I
COST OF PARTS FOR OSCILLOSCOPE

Quantity	Part Number	Description	Price
1	MC16F877	PIC microcontroller	1.84
1	LCD-S301C31TF	3 Digit LCD display	1.66
3	450-1649-ND	Pushbuttons	.76
3	10 KW	Resistor for PB	.09
1	LM-353	Dual input op-amp	.75
4	1N-4001	Blocking diodes	.20
4	10 MW	Blocking Resistor	.12
1	LM2574N-5	5 V step-down converter	1.84
1	22 uF	Converter capacitor	.14
1	220 uF	Converter capacitor	.22
1	330 uH	Converter inductor	.60
1	1N-4001	Converter Diode	.05
1	EG1916-ND	On/off switch	.73
1	9 V battery clip	9 V battery clip	.35
1	9 V battery	9 V battery	2.30
		Subtotal	**11.65**

1	PICKit Debug Express 2	Programming board	59.95
1	Wire wrapping tool	Wire wrapping tool	6.95
2	Wire wrapping wires	Wire wrapping wires	2.98
1	Project board	Project board	2.95
		Total	**84.48**

The following is a high level design of the circuit. For a complete schematic see the appendix.



The following is an overview for the code which runs the PIC. For the complete code see the appendix.

Initialize the program and variables.

If pushbutton 3 is pressed, run the test for 3.14 seconds. This test stores 20 values at .1 second intervals the first time the button is pressed. It stores 20 values at .01 second intervals the second time the button is pressed. During the test it measures the frequency, amplitude and average of the input signal.

If pushbutton 1 is pressed, it displays the average value of the recorded wave. Each successive press of button 1 cycles through the 20 stored values, and displays end after all values have been stepped through.

If pushbutton 2 is pressed, it displays the amplitude of the measured wave. Upon the second button press it displays the frequency. Every successive press cycles between these two values.

The code also has functions that convert ADC values to do be properly displayed , and control the LCD display.

III. Results

This section details the steps in designing the oscilloscope, time required, parts used, circuit schematic, and code used to program the microcontroller.

The oscilloscope has an operating range of -4.5 to +4.0 V. It can capture frequencies from 1-150 radians per second. It's run time for capturing input is 3.14 seconds. It outputs a sin wave at 10 Hz. while performing calculations. The oscilloscope has circuit protection so a user can not overload or short out the microcrontroller in any way. The oscillocope measures input data with a .1 V resolution. The final cost of building another oscilloscope with a printed circuit board is $17.50, which is under the $20 target. Additional features include displaying on when the device has finished powering up, displaying run while the circuit is running, and showing - - - when it has finished making capturing it's data. The oscilloscope displays 'end' when the user has finished stepping through the 20 recorded data points.

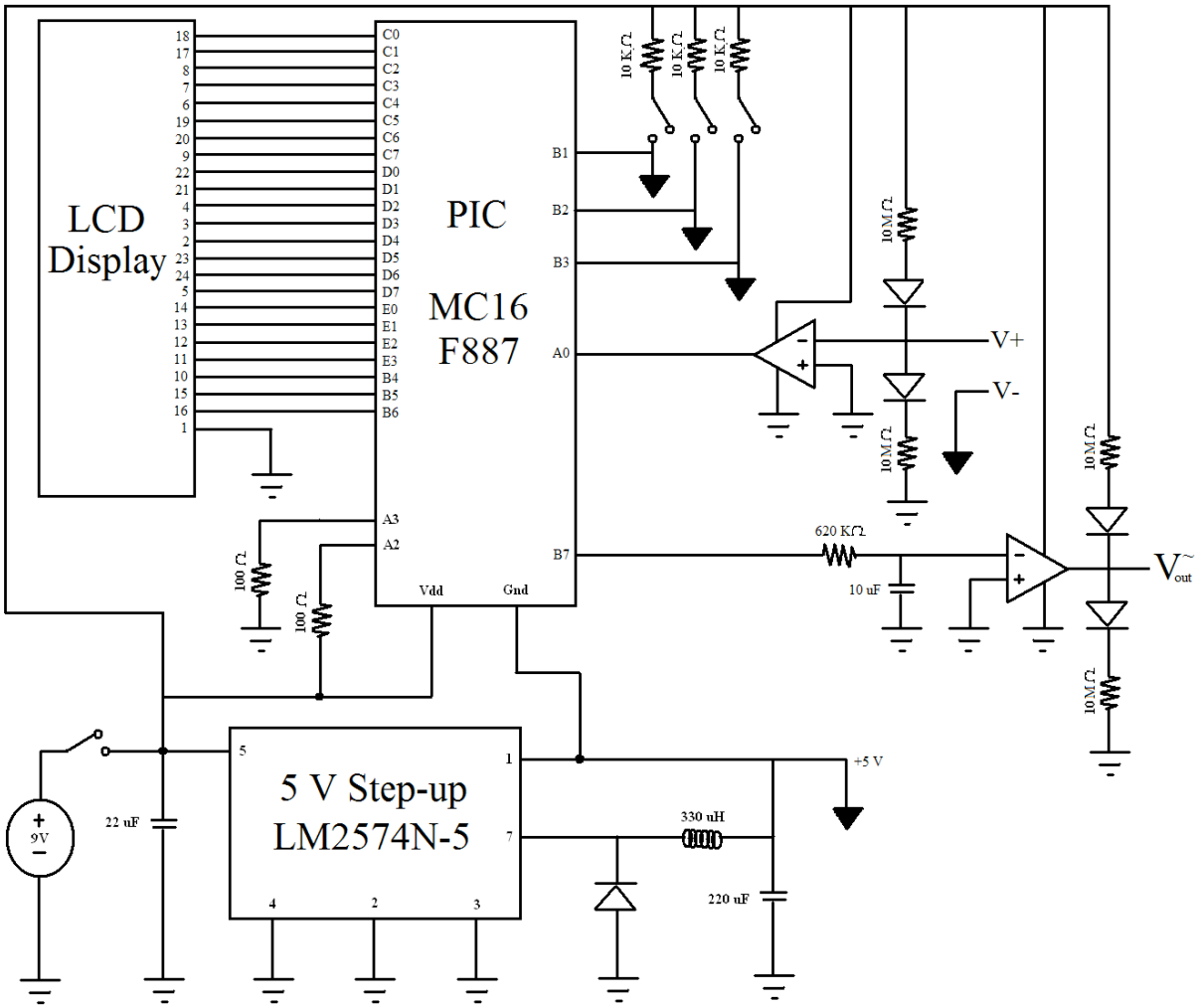
IV. Conclusion

The final goal of this project is to spark interest in younger students to draw them into the field of electrical engineering. Basic circuit design can be used to help teach math and physics principles to students at all levels. The basic circuits have already been designed, and are currently being used in schools to teach these principles. The oscilloscope packaged with these basic circuits complete the kits to be sold to teachers making it possible for students to design, build, and test circuits at their own desks.

The main difficulty of this project was to write the code for the PIC processor and to get parts that were compatible with each other for the lowest cost possible. The oscilloscope meets design specifications, and may be easily modified to meet the changing needs of teachers. The project as a whole is under budget, and the oscilloscope can be produced for less than $20 as intended.

Appendix

Circuit Schematic:


LCD Display
PIC MC16 F887
C0
C1
C2
C3
C4
C5
C6
C7
D0
D1
D2
D3
D4
D5
D6
D7
E0
E1
E2
E3
B4
B5
B6
B1
B2
B3
A0
A3
A2
B7
Vdd
Gnd
10 KΩ
10 KΩ
10 KΩ
10MΩ
10MΩ
10MΩ
10MΩ
V+
V-
100 Ω
100 Ω
620 KΩ
10 uF
V~out
9V
22 uF
5 V Step-up
LM2574N-5
330 uH
220 uF
+5 V

PIC code:

```
//Startup Definitions
#include <16F887.h>

#use fast_io(A)
#use fast_io(B)

#FUSES INTRC,NOWDT,NOPUT,NOMCLR,NOPROTECT,NOCPD,NOBROWNOUT,NOIESO,NOFCMEN,NOLVP


// global variable to be passed between functions
struct my_times
    {
    int value;
    } my_timer;

// function to set half the 3rd LCD digit
int set_u3display(int digit)
{
    int lights = 0;
    if (digit == 0)
    {
      lights = 0x70;
    }
    if (digit == 1)
    {
      lights = 0x00;
    }
    if (digit == 2)
    {
      lights = 0xB0;
    }
    if (digit == 3)
    {
      lights = 0x90;
    }
    if (digit == 4)
    {
      lights = 0xC0;
    }
    if (digit == 5)
    {
      lights = 0xD0;
    }
    if (digit == 6)
    {
      lights = 0xF0;
    }
    if (digit == 7)
    {
      lights = 0x00;
    }
    if (digit == 8)
    {
      lights = 0xF0;
    }
    if (digit == 9)
    {
      lights = 0x0C0;
```

```
    }
    return lights;
}

/function to set the other half of the 3rd LCD digit
int set_l3display(int digit)
{
    int lights = 0;
    if (digit == 0)
    {
      lights = 0x07;
    }
    if (digit == 1)
    {
      lights = 0x06;
    }
    if (digit == 2)
    {
      lights = 0x03;
    }
    if (digit == 3)
    {
      lights = 0x07;
    }
    if (digit == 4)
    {
      lights = 0x06;
    }
    if (digit == 5)
    {
      lights = 0x05;
    }
    if (digit == 6)
    {
      lights = 0x04;
    }
    if (digit == 7)
    {
      lights = 0x07;
    }
    if (digit == 8)
    {
      lights = 0x07;
    }
    if (digit == 9)
    {
      lights = 0x07;
    }
    return lights;
}

//this function can be used to set the other 2 digits of the LCD
int set_display(int digit)
{
    int lights = 0;
    if (digit == 0)
    {
      lights = 0x3F;
    }
    if (digit == 1)
```

```
    {
      lights = 0x06;
    }
    if (digit == 2)
    {
      lights = 0x5B;
    }
    if (digit == 3)
    {
      lights = 0x4F;
    }
    if (digit == 4)
    {
      lights = 0x66;
    }
    if (digit == 5)
    {
      lights = 0x6D;
    }
    if (digit == 6)
    {
      lights = 0x7D;
    }
    if (digit == 7)
    {
      lights = 0x07;
    }
    if (digit == 8)
    {
      lights = 0x7F;
    }
    if (digit == 9)
    {
      lights = 0x67;
    }
    return lights;
}

// this function takes the 8-bit ADC value and converts it
// into an analog value to be output to the LCD.
int code_to_DC(int digit)
{
    int temp1 = 0;
    int temp2 = 4;
    int temp0 = 0;
    int cool = 0;

    for (temp0 = 0; temp0 < 255; temp0++)
    {
      if (temp0%5 == 0)
      {
        temp2 = temp2 + 1;
        temp1 = temp1 + 1;
      }
      if (temp1 == 5)
      {
        temp2 = temp2 - 1;
        temp1 = 0;
      }
      if (temp0%30 == 0)
```

```
      {
        temp2 = temp2 + 1;
      }
      if (temp0 == digit)
      {
        cool = temp2;
      }
    }
    return cool;
}

// This function accepts an 8 bit value, and outputs it to the LCD
void LCD_Display(int digit)
{
    int lights = 0;
    int lights2 = 0;
    int lightsu3 = 0;
    int lightsl3 = 0;

    lights = set_display((digit-digit%100)/100);
    lights2 = set_display(((digit-digit%10)/10)-(digit-digit%100)/10);
    lightsu3 = set_u3display(digit%10);
    lightsl3 = set_l3display(digit%10);

    OUTPUT_B(lightsu3);
    OUTPUT_C(lights2);
    OUTPUT_D(lights);
    OUTPUT_E(lightsl3);
}

//These next functions display run,---, on, and end on the LCD
void Display_run()
{
    OUTPUT_B(0xA0);
    OUTPUT_C(0x1C);
    OUTPUT_D(0x50);
    OUTPUT_E(0x04);
}

void Display_on()
{
    OUTPUT_B(0xA0);
    OUTPUT_C(0x5C);
    OUTPUT_D(0x00);
    OUTPUT_E(0x04);
}

void Display_No()
{
    OUTPUT_B(0xB0);
    OUTPUT_C(0x54);
    OUTPUT_D(0x00);
    OUTPUT_E(0x0C);
}

void Display_dash()
{
    OUTPUT_B(0x80);
    OUTPUT_C(0x40);
    OUTPUT_D(0x40);
```

```
    OUTPUT_E(0x00);
}

void Display_End()
{
    OUTPUT_B(0xB0);
    OUTPUT_C(0x54);
    OUTPUT_D(0x79);
    OUTPUT_E(0x0E);
}

// This function sets the first bit to be blank, and turns on the decimal.
void Output_Voltage(int digit)
{
    int temp = 0;
    temp = code_to_DC(digit);
    LCD_Display(temp);
    OUTPUT_D(0x00);
    OUTPUT_HIGH(PIN_C7);
}

//Initialize the IO pins:
void init_io()
{
    SET_TRIS_A(0x0D);
    SET_TRIS_B(0x0F);
    SET_TRIS_C(0x00);
    SET_TRIS_D(0x00);
    SET_TRIS_E(0x00);
}

//Initializes the ADC Converter
void init_adc()
{
    SETUP_ADC_PORTS(0x0D | VREF_VREF);
    SETUP_ADC(ADC_CLOCK_DIV_8);
    SET_ADC_CHANNEL(0);
    READ_ADC(ADC_START_ONLY);
}

//This function is used for the accurate timing.
#INT_RTCC
void timer0_interrupt_service()
{
    my_timer.value = my_timer.value + 1;
}

//Beginning of Program
void main()
{
    //Declare Variables
    int switcha = 0;
    int switchb = 0;
    int switchc = 0;
    int switchd = 0;
    int temp1 = 0;
    int temp2 = 0;
    int temp3 = 0;
    int temp4 = 0;
    int temp5 = 1;
```

```
    int  time_length = 1;
    int write_count = 0;
    int read_count = 0;
    int data_points[22] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
    int data_points2[22] =
{0,100,150,175,200,200,150,100,50,25,0,25,50,100,150,200,225,200,150,100,50,25};
    int recent_values[3] = {0,0,0};
    int freq = 0;
    int freq_check = 0;
    int freq_flag = 0;
    int fa_display = 1;
    int max = 0;
    int min = 255;
    int amplitude = 0;
    int average = 0;
    int average_helper = 0;
    int dv = 0;
    int max_dv = 0;

    display_on();

    // Set up MCU
    init_io();
    init_adc();
    SETUP_TIMER_0(RTCC_INTERNAL | RTCC_DIV_4);
    ENABLE_INTERRUPTS(GLOBAL);
    ENABLE_INTERRUPTS(INT_RTCC);

    //This Section of the program is constantly looping and checking for button presses
    while(1)
    {

      //This section of the code runs when button 3 is pressed
      if (INPUT_STATE(PIN_B3) == 0)
      {
        //this if statement acts as a software debounce for the switch
        if (switchd <= 12)
        {
          switchd = switchd + 1;
        }
        if (switchd == 12)
        {
          //this if statement cycles between .1 second and .01 second measurements
          if (time_length == 1)
          {
            time_length = 9;
          }
          if (time_length == 10)
          {
               time_length = 0;
          }
          time_length = time_length+1;

          //resetting all requrred values
          write_count = 0;
          read_count = 0;
          freq = 0;
          freq_check = 0;
          freq_flag = 0;
          max = 0;
```

```
            min = 254;
            fa_display = 1;
            average = 0;
            average_helper = 0;

            Display_run();

            // this section performs all the nesecarry measurements for amplitude,
            // frequency, the first 20 stored values, and average.  It runs for 3.14 s
               for (temp1 = 0; temp1 < 3; temp1++)
            {
               for (temp2 = 0; temp2 < 10; temp2++)
               {
                  //This part of the code creates a 10 Hz square wave.
                  if (temp2 == 4)
                  {
                     OUTPUT_D(0xD0);
                  }
                  if (temp2 == 9)
                  {
                     OUTPUT_D(0x50);
                  }

                  //this section of the code completes the last .14 sec of calculations
                  // it performs the same functions as listed in the code below.
                  if (temp1 == 2)
                  {
                     if(temp2 == 1)
                     {
                        for (temp3 = 0; temp3 < 14; temp3++)
                        {
                           temp4 = READ_ADC(ADC_READ_ONLY);
                           READ_ADC(ADC_START_ONLY);
                           recent_values[0] = recent_values[1];
                           recent_values[1] = recent_values[2];
                           recent_values[2] = temp4;

                           if (write_count > 3)
                           {
                              dv = recent_values[1] - temp4;
                              if (dv > 250)
                              {
                                 dv = temp4 - recent_values[1];
                              }
                           }
                           if (dv > max_dv)
                           {
                              max_dv = dv;
                           }

                           if (((recent_values[0] <= recent_values[1])&(recent_values[2] <
recent_values[1]))|((recent_values[1] >= recent_values[2])&(recent_values[3] >
recent_values[2])))
                           {
                              freq = freq + 1;
                              freq_flag = 1;
                           }
                           if ((freq_flag == 1)&((freq_check - temp4 < 1)|(freq_check - temp4 >
254)))
                           {
```

```
                        freq_flag = 0;
                        freq = freq - 1;
                     }
                     else
                     {
                        freq_check = temp4;
                     }

                     for(my_timer.value = 0; my_timer.value <10;)
                     {
                     }
                     my_timer.value = 0;
                  }
               }
            }

            //This section of the code runs every .01 seconds checking ans storing values
            for (temp3 = 0; temp3 < 10; temp3++)
            {
               // Reads the ADC result, and makes a new measurement.
               temp4 = READ_ADC(ADC_READ_ONLY);
               READ_ADC(ADC_START_ONLY);
               // stores the last 3 values to be used to find frequency
               recent_values[0] = recent_values[1];
               recent_values[1] = recent_values[2];
               recent_values[2] = temp4;

               // This section of the code finds the frequency
               if (((recent_values[0] <= recent_values[1])&(recent_values[2] <
recent_values[1]))|((recent_values[1] >= recent_values[2])&(recent_values[3] >
recent_values[2])))
               {
                  freq = freq + 1;
                  freq_flag = 1;
               }
               if (freq_flag == 1)
               {
                  freq_check = temp4;
               }
               if ((freq_flag == 1)&((freq_check - temp4 < 4)|(freq_check - temp4 > 250)))
               {
                  freq_flag = 0;
                  freq = freq - 1;
               }

               // this section acts as error checking on the frequency
               if (write_count > 3)
               {
                  dv = recent_values[1] - temp4;
                  if (dv > 250)
                  {
                     dv = temp4 - recent_values[1];
                  }
               }
               if (dv > max_dv)
               {
                  max_dv = dv;
               }

               // This determines the amplitude
```

```
            if (write_count > 1)
            {
               if (temp4 > max)
               {
                  max = temp4;
               }
               if (temp4 < min)
               {
                  min = temp4;
               }
            }

            //Stores 20 values at .01 s resolution
            if (time_length == 1)
            {
               write_count = write_count+1;
               if (write_count < 22)
               {
                  data_points[write_count - 1] = temp4;
               }
               else
               {
                  write_count = write_count - 1;
               }
            }
            for(my_timer.value = 0; my_timer.value <10;)
            {
            }
            my_timer.value = 0;
         }

         // Stores 20 values at .1 s resolution
         if (time_length == 10)
         {
            write_count = write_count+1;
            if (write_count < 22)
            {
               data_points[write_count - 1] = temp4;
            }
            else
            {
               write_count = write_count - 1;
            }
         }
      }
   }
   Display_dash();
  }
 }
 else
 {
  switchd = 0;
 }

// This section of the code runs when button 1 is pressed.
if (INPUT_STATE(PIN_B1) == 0)
{
  // Debounce
  if (switcha <= 12)
  {
```

```
            switcha = switcha + 1;
          }
          if (switcha == 12)
          {
            // resets critical values
            fa_display = 1;
            data_points2[0] = 0;

            // Finds the average of the 20 values
            for (temp1 = 0; temp1 < 20; temp1++)
            {
              data_points2[0] = data_points2[0] + ((data_points2[temp1+1]-
data_points2[temp1+1]%20)/20);
              average_helper = average_helper + data_points2[temp1+1];
              if (average_helper >= 24)
              {
                data_points2[0] = data_points2[0] + 1;
                average_helper = average_helper - 20;
              }
            }

            // Steps through the 20 stored values, and displays them on the LCD
            if (read_count <= 20)
            {
              Output_Voltage(data_points2[read_count]);
            }
            else
            {
              Display_End();
            }
            read_count = read_count + 1;
          }
         }
         else
         {
          switcha = 0;
         }

       // Runs when button 2 is pressed.
       if (INPUT_STATE(PIN_B2) == 0)
       {
          // Debounce
          if (switchb <= 12)
          {
            switchb = switchb + 1;
          }
          if (switchb == 12)
          {
            // Reset critical values
            read_count = 0;

            // Cycles through frequency and amplitude
              if (fa_display == 1)
            {
              fa_display = 9;
            }
            if (fa_display == 10)
            {
                fa_display = 5;
            }
```

```
      if (fa_display == 6)
      {
         fa_display = 0;
      }
      fa_display = fa_display + 1;

      // Displays the frequency
      if (fa_display == 6)
      {
         freq = 64;
         LCD_Display(freq);

         // Error check on Freq
         if ((freq == 255)|(freq == 0))
         {
            Display_No();
         }
      }

      // Displays the amplitude
      if (fa_display == 10)
      {
         amplitude = max-min;
         Output_Voltage(225);
      }

      // Extra code for prototype, not used in actual code
      if (fa_display == 1)
      {
         LCD_Display(max_dv);
      }
   }
  }
  else
  {
   switchb = 0;
  }

// This section of the code was also used in the prototype, but is being
// left in so that a 4th button can be added which will cause the oscilloscope
// to act as a voltmeter.
if (INPUT_STATE(PIN_B0) == 0)
{
   if (switchc <= 12)
   {
      switchc = switchc + 1;
   }
   if (switchc == 12)
   {
      read_count = 0;
      if (temp5 == 1)
      {
         temp5 = 9;
      }
      if (time_length == 10)
      {
           temp5 = 0;
      }
      temp5 = temp5+1;
   }
```

```
        }
        else
        {
         switchc = 0;
        }
      if (temp5 == 10)
      {
         temp4 = READ_ADC(ADC_READ_ONLY);
         READ_ADC(ADC_START_ONLY);
         Output_Voltage(temp4);
      }
    }
}
```