Lab Manual for BME 361
Biomeasurement Laboratory

The Department of Electrical, Computer, and Biomedical Engineering
University of Rhode Island
Kingston, Rhode Island, 02881
Course website: `http://www.ele.uri.edu/courses/bme360/`

Spring 2016

Preface

Biomeasurement is fundamental to our understanding of biology, physiology, and medicine. While biomeasurement may refer to any observation (pulse, respiration rate, etc.) increasingly, measurements rely on the advances of technology and engineering to provide more accurate estimation of biological signals (electrical and mechanical) and help interpret the underlying physiological mechanisms responsible for those signals. Perhaps the best known and widely used example of biomeasurement is the ECG (electrocardiogram, or EKG elektrokardiogramm (GER)). The history of the ECG can be traced to 1872 when an electrical engineering PhD student connected wires to the wrist of a patient in St. Bartholomew's Hospital in London. Since then, it has become the gold standard for vital sign assessment. Though the underlying methodology of obtaining the ECG is little changed in the past 100 years, history, and a fair amount of signal processing techniques, have allowed us to interpret the ECG with surprising diagnostic acumen. To be sure, without the ECG and the countless other forms of biomeasurement, medicine and health care would have a remarkably different landscape.

In this laboratory, you will be introduced to some of the critical design considerations when attempting to acquire a biological signal. Issues such as noise, small signal-to-noise ratios, electrode interface and biocompatibility, as well as signal processing all combine to make the acquisition of a biological signal a non-trivial exercise. Further considerations are given to analog to digital (A/D) conversion, microprocessor-based circuitry, and LCD screen display.

Contents

List of Figures

Lab 1: Introduction to PIC18F4525 and MPLAB: Binary Counter

If you have never used MPLab please refer to Introduction to MPLab - Learning Exercise in the appendix.

PURPOSE: Simple introduction to the PIC processor as well C++ programming using MPLAB. This lab will give you a basic idea of how to program the PIC processor as well as implement it on a breadboard.

GOAL: Today you will be creating a system with a binary counter that will count from 0-7 (using four LED's). To complete this task you must first create the circuit shown in Figure 1 on your breadboard, as well as download the code for the binary counter from the course website `http://www.ele.uri.edu/courses/bme360/` listed as Sample Program: C code. This is a stock program, meaning it provides basic functionality but does not perform the exact function you will need to build. In its current form, the program simply sets up the analog to digital conversion (A/D) function of the chip. Inside the main loop, you can build the binary counter.

You will need to build the circuit and follow the procedure portion of the lab to get the initial counter working. Once you have the binary counter working correctly, you may move onto the Tasks portion of the lab.

IMPORTANT: The stock code performs A/D. The program initializes a pin on the chip to use for this purpose. The problem here is that if you leave the pin floating (not connected to anything), the onboard A/D has a difficult time getting a value and so it will affect the performance of your program. There are two ways to deal with this: hardware or software. For the hardware solution, you can simply tie that pin to ground or 5V with a 10 kΩ resistor (this is called a pull-down or pull-up resistor, respectively). Alternatively, you can comment out the section of the code that performs the A/D function. In either case, you will need to determine where in the code this is performed and which pin is assigned to the A/D function.

MATERIALS:

- 9 Volt Battery & Clip
- Breadboard
- 4 LED's
- (4) 470 Ω Resistor
- 10 kΩ Resistor
- Microcontroller PIC18F4525
- 4 MHz ceramic resonator
- LM7805 5V voltage regulator
- 6 pin ICSP header

SCHEMATIC:

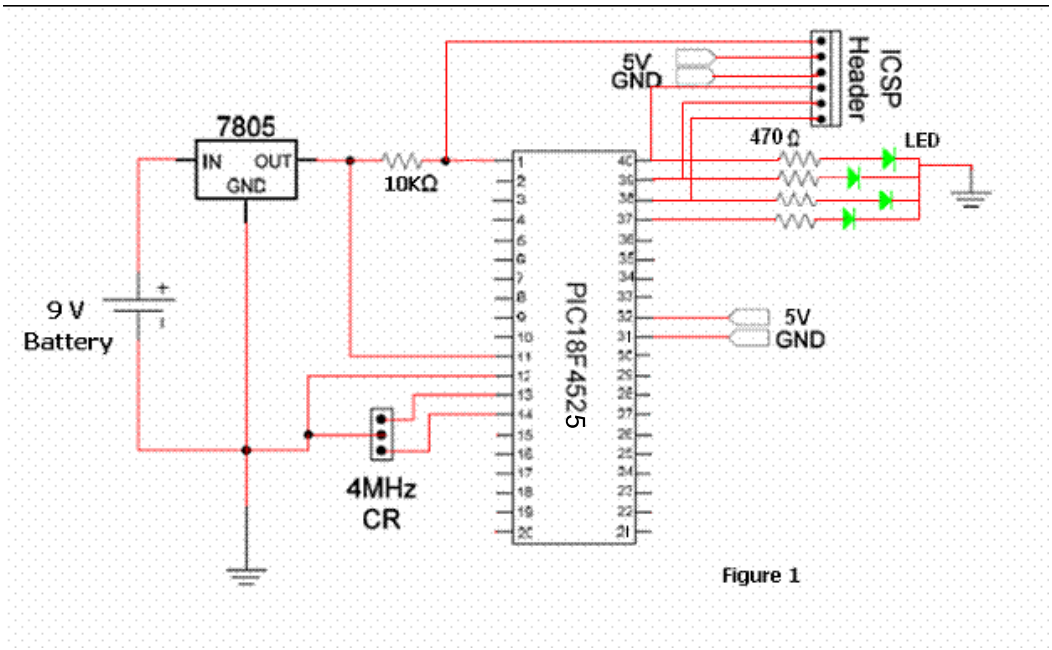
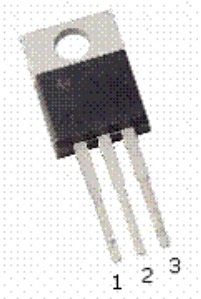
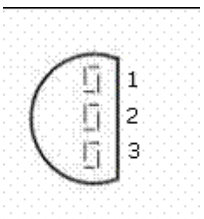


Figure 1.1: Schematic for test circuit.



(a) LM7805 - TO-220 packaging

(b) Pin assignments



(c) LM7805 - Transistor packaging

Figure 1.2: 5V regulators and pin assignments

PROCEDURE:
Hardware

1. Using your breadboard, implement the circuit seen above in Figure 1.

2. Be sure you have all the necessary components, and that they are oriented correctly. (Note: if you do not know/remember the pin assignment for a certain component, look up the datasheet and use it as a reference.)

*IMPORTANT: The ceramic resonator shown in Figure 1 must be orientated correctly with the label side facing away from the PIC.

Software

1. Open MPLAB on either a desktop computer in the lab, or your own laptop; if you would like to download MPLAB onto your personal laptop, you can follow the installation manual.
2. Download the stock program.

TASKS:

1. You must add code to execute a binary counter in the stock code. You shouldn't have to change the original code, just write your own new code in the 'main' section.

2. Once you have modified the code you will need to download the program to your PIC:

- Connect the MPLAB ICD3 programmer to the ICSP Header and to the PC, and download the program onto the PIC (figure 1.3).
- YOU MAY NEED TO REMOVE THE LEDs from pins 40 and 39 as the diode creates a capacitive load on the programming pin. This capacitance interferes with the fast changing signals being

sent to your microcontroller by the programmer and computer. Hmmm, seems you may have heard something about capacitance and high frequency signals somewhere along your academic prerequisites.

- You do not need to disconnect the MPLAB ICD3 programmer from the ICSP Header. At this point you should see the LED's lighting up as a binary counter.

Figure 1.3: Microchip MPLAB ICD3 Programmer

TROUBLESHOOTING:

There are some common errors related to improperly building the circuit shown in figure 1.1. Pay close attention to the power connections associated with pin 1 as well as the 5V connections. Often, the 5V connections appear after the 10 kΩ resistor that connects to pin one. That is, it's on the wrong side of the resistor. Remember, if you have a voltage supply on one side of a resistor, a certain amount of the voltage will drop across that resistor depending on what other resistance is in that leg of the circuit. This means that you WILL NOT be supplying 5V to the power connections that require 5V.

Check the value of all resistors. Remember ROYGBIV - Red is 100 multiplier, Orange is 1000 multiplier, Yellow is 10000 multiplier, etc. For instance, a brown stripe followed by a black stripe followed by a yellow stripe is a 100 kΩ resistor - 10 (brown is 1, black is 0) times 10000 = 100000.

Make sure your configuration bits are correctly set. Refer to appendix A, **Introduction to MPLab - Learning Exercise** if you are unsure.

Lab 2: ECG Simulation

You must have a fully functional PIC circuit from LAB 1 to continue on with this lab.

PURPOSE: Further explore the functional relationship between the PIC Microprocessor and C++ Programming using MPLAB. This lab will include the digital to analog converter and the voltage converter; these chips are essential for the implementation of the ECG simulation. By understanding how these two chips work you will have a greater appreciation for the PIC microprocessor and MPLAB and realize their many possibilities.

GOAL: In today's lab you will be creating an ECG Simulation by building upon the circuit you constructed in Lab 1. You will do this by adding new components (highlighted in Figure 1), both the DACØ8ØØ (Digital to Analog Converter) and LMC7660 (Voltage Converter) chips, as well as adding a new program.

MATERIALS:

- DAC 0800 chip
- LMC7600
- (2) 0.1uF Capacitor
- 0.01uF Capacitor
- (2) 10uF Capacitor

SCHEMATIC:

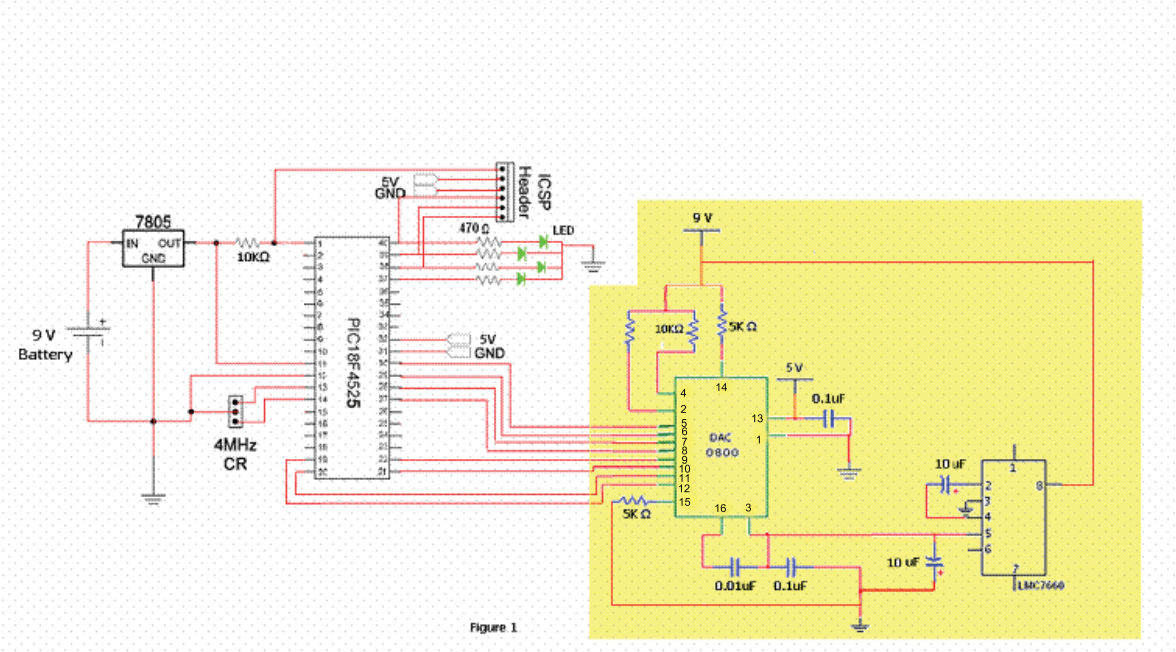


Figure 2.1: Schematic of circuit with DAC for ECG simulation

PROCEDURE:

Hardware

1. Using your breadboard and circuit from Lab 1, add the highlighted section seen above in Figure 1 to your previously existing circuit.

2. Be sure you have all the necessary components, and that they are oriented correctly. (Note: if you do not know/remember the pin assignment for a certain component, look up the datasheet and use it as a reference.)

Software

1. Open MPLAB and download the ECG Simulation program. You will need to add this code to the program from Lab 1 - this will be the beginning of your "master program".

2. Connect theMPLAB ICD3 programmer to the ICSP Header and to the PC, and download your new program onto the PIC.

3. Disconnect theMPLAB ICD3 programmer from the ICSP Header.

TASKS:

1. Once you have programmed the PIC you need to use an oscilloscope to test your circuit.

2. Connect the oscilloscope between pins 2 and 4 on the DAC 0800chip.

NOTE: This means that the "GND" wire of the oscilloscope will be at pin 4 (not connected to GND on your breadboard), while the normal lead will be at pin 2.

3. The oscilloscope should now display an ECG signal automatically. If at first the signal is too small, you may need to adjust the settings on the oscilloscope to see the signal more clearly.

2.1 Lab 2 - ECG Simulation Supplement

This is a real-time program. Instead of storing the entire ECG waveform in the memory and playing it back, compute what to send to the D/A converter on a point-by-point basis. You can implement the code in the interrupt service routine. Set the timer to generate periodical interrupts at an appropriate interval - suggesting 1 ms. So you need to figure out the hexadecimal numbers for loading the TMR0H and TMR0L. Study the diagram in figure 2.2 carefully for the design of the ECG waveform. Learn to use the "switch" instruction to implement the various modes for generating the individual segments of the ECG waveform. See sample code on the following page.

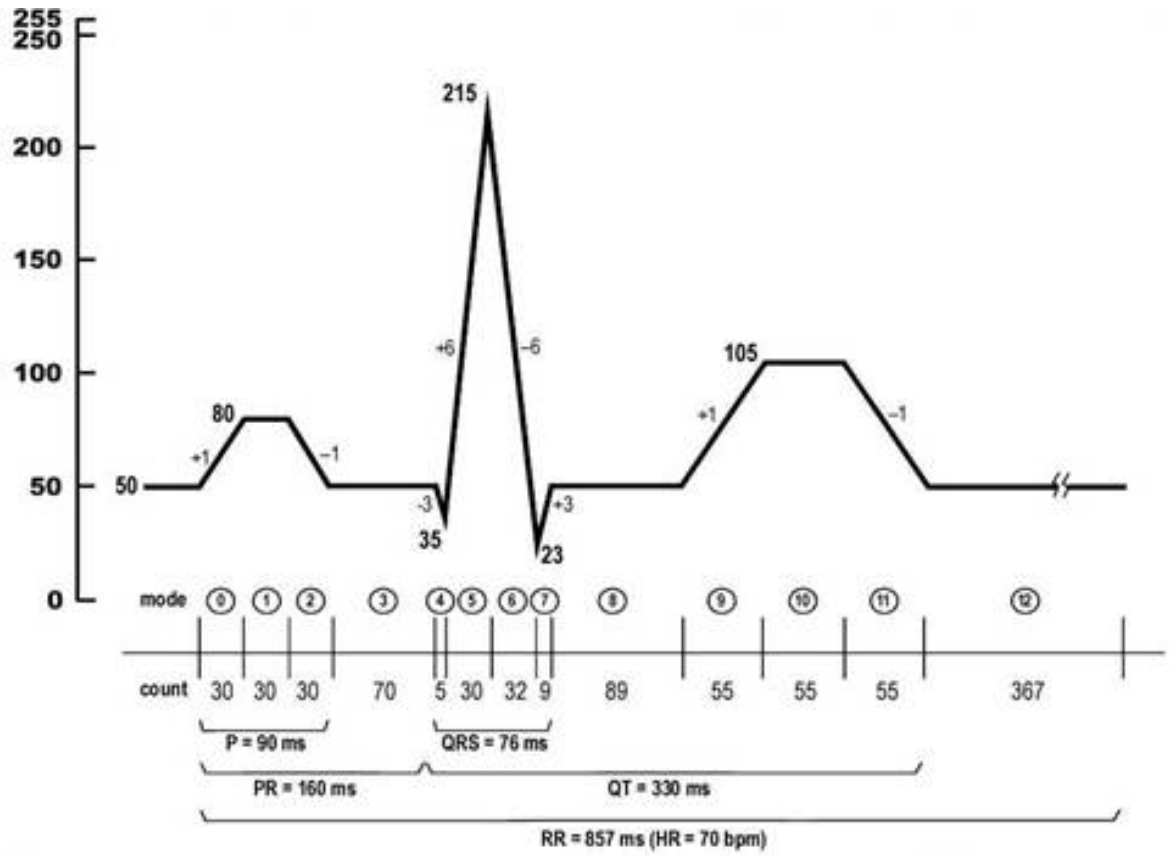


Figure 2.2: Timing diagram with waveform intervals for ECG simulation.

```
1  Declare global variables:
2
3  unsigned char da_output, mode;
4  unsigned int count;
5
6  void _highPriorityInt(void){
7  checkflags:
8    if(INTCONbits.TMR0IF == 1) {  // When there is a timer0 overflow, this loop ↩
       runs
9      INTCONbits.TMR0IE = 0;      // Disable interrupt
10     INTCONbits.TMR0IF = 0;     // Reset timer 0 interrupt flag to 0
11     switch (function) {
12     case 0:              // LAB 1 - BINARY COUNTER
13       TMR0H  =0x??;        // Reset timer count: high-order and low-order bytes
14       TMR0L = 0x??;          // 0xFFFF - 0x???? = 0x03E8 = 1000 (decimal) ¬ 1ms
15       break;
16     case 1:              // LAB 2 - ECG SIMULATION
17       //ad_input = ReadADC();      // Bonus Section - Variable Heartrate
18       TMR0H = 0x??;    // Load upper 8 bits to TMR0H
19       TMR0L = 0x??;     // Load lower 8 bits to TMR0L
20       switch(mode){
21       case 0:      // P wave up
22         count++;
23         da_output++;
24         if (count == 30) mode++;
25         break;
26       case 1:      // P wave flat
27         count--;
28         if (count == 0) mode++;
29         break;
30       case 2:      // P wave down
31         count++;
32         da_output--;
33         if (count == 30) mode ++;
34         break;
35       }
36     }
37     PORTD = da_output;
38     INTCONbits.TMR0IE = 1;     // Re-arm by enabling the Timer0 interrupt
39       goto checkflags;          // Check again in case there is a timer interrupt
40   }
41 }
```

Lab 3: Echo and Derivative Programs

PURPOSE: Introduce students to writing and implementing various programs using C++ and MPLab. Students will modify the provided Echo program to display the derivative of a signal. This lab will enhance students' programming skills in the C++ language.

GOAL: In today's lab you will be exploring an echo program and later implementing a derivative program - quite literally taking the derivative (discrete time) of your input signal. You will do this by studying the provided code, and then adapting it to fit the specifications presented in the Tasks portion of this lab.

PROCEDURE:

(You will be using your breadboard and circuit from Lab 2.)

Software

1. Open MPLAB and add the following code. Before you can add this code you must make some minor modification to the ECG simulation code. Essentially, you will be placing all the `switch − case` statements inside a larger `switch − case`. Follow the layout of lab 2 when the ECG code was made `case` 1: . (Recall, we also made the Binary Counter `case` 0:). Now `case` 2: will be the ECHO and `case` 3: will be the DERIVATIVE. It will take you a little while to get comfortable with this but it will save time later.

```
1  // LAB 3 - ECHO
2  case 2:
3      TMR0H  =0x??; // Reset timer count for 240 Hz
4      TMR0L = 0x??; // 0xFFFF-0x???? = 4167 => 240 Hz
5      da_output = ReadADC();  // Read A/D and send it to output
6      break;
7  // LAB 3 - DERIVATIVE
8  case 3:
9      TMR0H  =0x??; // Reset timer count for 240 Hz
10     TMR0L = 0x??; // 0xFFFF-0x???? = 4167 => 240 Hz
```

```
11      data1 = data0;  // Save the previous sample in data1
12      data0 = ReadADC(); // Read current ADC and save in data0
13      dumb = data0;
14      dumb -= data1;  // Backward difference: data0 - data1
15      dumb += 128;  // Shift baseline up
16      if (dumb > 255) dumb = 255;
17      if (dumb < 0) dumb = 0;
18      da_output = dumb;
19      break;
```

2. Connect theMPLAB ICD3 programmer to the ICSP Header and to the PC, and upload the program onto the PIC.

3. Disconnect theMPLAB ICD3 programmer from the ICSP Header.

Hardware

4. You can now test the Echo program using a signal generator and an oscilloscope. You will do this by connecting the positive lead of the signal generator to pin 2 of the PIC (the negative goes to ground) and the oscilloscope probe to pin 2, with the probe ground to pin 4 of the DAC 0800, just like lab 2.
The echo program will take an analog input signal (from the signal generator) and convert that analog signal to a digital signal so it can be used as digital input to the PIC, where it will be reproduced exactly - or at least to within quantization error. The output signal will then be converted back to analog on a digital to analog converter where it can be viewed on an oscilloscope.

5. You may chose to input any type of wave from the signal generator (square, sign, or triangle) and the oscilloscope should display the exact same wave. Be sure to use both channel 1 and 2 on the oscilloscope. Do this by connecting channel 1 to the signal generator and channel 2 to the output (pin 2) of the DAC 0800. By doing this you will be able to determine if your output is the exact "echo" of your input.

TASKS:

1. Now that you have seen what the echo program does, you will need to study the code in order to modify it to produce a derivative program.

The derivative program will take an input from the signal generator just like the echo program, but this time your program in the PIC will produce the derivative of the original input. For example if your input signal is a triangle wave, than your output signal would be a square wave.

HINTS:

1. You will need to declare three unsigned characters (one for the current data point, one for the previous data point, and one for the output), and at least one integer as a "dummy variable".

2. In order to obtain the derivative you will need to subtract each current data point from the previous data point.

3. You will then need to store that value as your dummy variable (at this point this value may be positive or negative depending on the two points you have used). Since you will be running this value through the PIC and ADC, you must find the absolute value of this variable as you cannot use negative values.

4. After you have converted the output back to an unsigned character, you must assign its value to PORTD of the PIC.

Lab 4: Implementation of Various Modes and LCD Display

PURPOSE: Introduce students to writing and implementing numerous modes using C++ and MPLab, as well as introduce a push button and LCD Display to your project.

GOAL: In today's lab you will be exploring modes and later using them to creating your own program. You will do this by studying code you have been given, and modifying it to fit the specifications presented in the Tasks portion of this lab. You will also have the opportunity to create your own code which will implement a push button to drive the various modes in your program. Finally, you will be give code for an LCD Display which must be slightly modified to accommodate your project.

MATERIALS:

- LCD Display Screen
- Push Button
- 10 kΩResistor

SCHEMATIC:

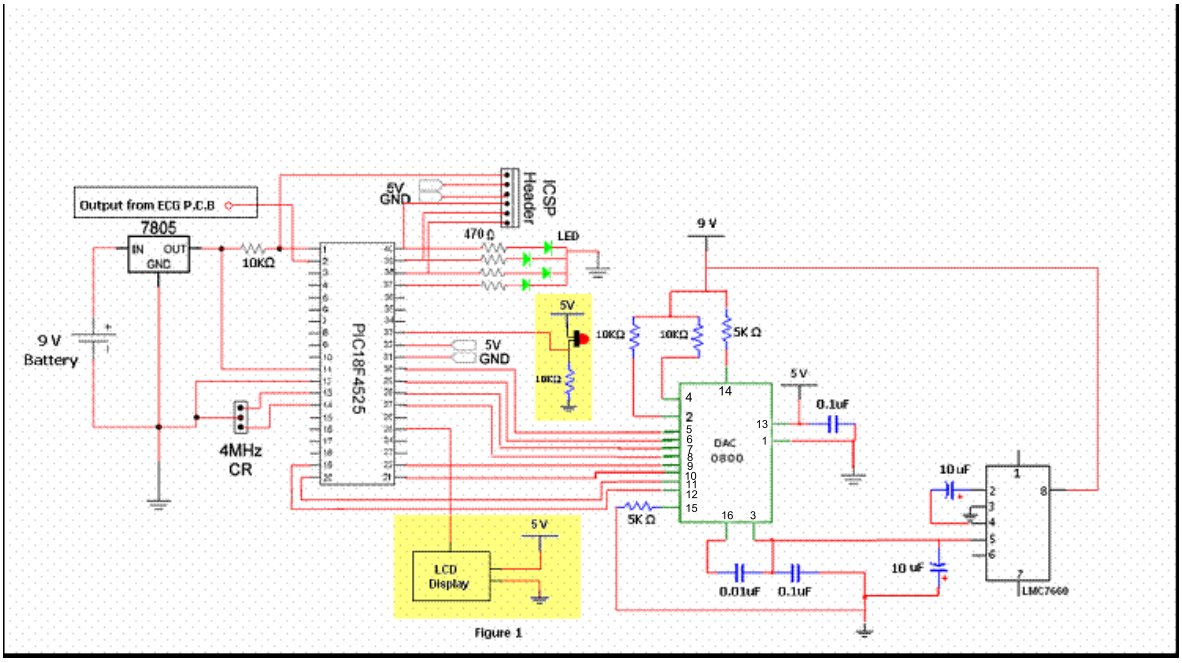


Figure 4.1: Schematic of circuit for LCD screen and multi-mode display

PROCEDURE:

Hardware

1. Using your breadboard and circuit from Lab 3, add the highlighted section seen above in Figure 1 to your previously existing circuit.

Software

1. Open MPLAB and add both the LCD Display and Modes programs to your already existing master program.

TASKS:

1. Since you have been given code for the LCD Display you will only need to create your own function to output variables to the LCD. The program will already be set up to display information on the screen - now you just need to tell it what to display.

2. You have been given sample code for basic mode and LCD programs - now you will need to modify them to include multiple modes to display all the programs you have used in the previous labs.

So far you have worked on an ECG Simulation, as well as Echo and Derivative programs. You will

need to include at least three modes to accommodate these programs.

3. Reference PortB on your datasheet for the PIC to determine how to program the pushbutton you have just added at pin 33. This will require you to use a high priority interrupt routine.

The purpose of an interrupt routine is to put the normal program on hold, execute a subroutine, and then continue on with the normal program. The subroutine can only be executed after a certain event (interrupt) has occurred. In this case you may want to implement a flag which will activate the high priority interrupt.

4.0.1 LAB 4 Supplemental

```
1  //BME361/BME463 Lab #4 Implementation of Various Modes and LCD Display - Supplement
2
3  /****************************** Global variables *********************************/
4  unsigned char output, counter, mode, function, data0, data1;
5  unsigned char LCD_update, LED_count, LED_count1, buttondelay;
6  int dummy;
7  ......
8
9  /*********Prototype functions***********/
10 void backlight(unsigned char state);
11 void SetPosition(unsigned char position);
12 void PrintLine(rom unsigned char *string, unsigned char numChars);
13 void PrintNum(unsigned char value, unsigned char position);
14 void SetupSerial();
15 void SetupADC(unsigned char channel);
16 void delay_ms(unsigned char x);
17 void ClearScreen();
18
19 void _highPriorityInt(void) /***** high priority interrupt service routine *********/
20 {
21 checkflags:
22   if(INTCONbits.TMR0IF == 1) {     // When there is a timer0 overflow, this loop runs
23     INTCONbits.TMR0IE = 0;       // Disable interrup
24     INTCONbits.TMR0IF = 0;       // Reset timer 0 interrupt flag to 0
25     if (buttondelay != 0) buttondelay--;  // Delay to debounce pushbutton
26     switch (function) {
27     case 0:        // ECG simulation
28       TMR0H = 0xFC; // Reload Timer 0 for 1 ms count
29       TMR0L = 0x17; // 0xFFFF-0xFC17 = 0x3E8 = 1000
30       switch (mode) { // ECG Simulation from Lab #2
31       case 0:      // P wave up
32         counter++;
33         output++;
34         if (counter == 30) mode++;
35         break;
36       case 1:      // P wave flat
37         .......
38       case 13:
```

```
39          .......
40          break;
41        }
42        break;
43      case 1:         // Echo
44        TMR0H  =0xEF; // Reset timer count: high-order and low-order bytes
45        TMR0L = 0xB8; // 0xFFFF-0xEFB8 = ↩
              1047 = 4167 => 4.167ms => 240Hzoutput = ReadADC(); //ReadA/DandsendittooutputGO = 1; //Restar
              1047 = 4167 => 4.167 ms => 240 Hz
46
47
48
49
50
51
52        data1 = data0;           // Save the previous sample in data1
53        data0 = ReadADC();         // Read ADC and save the present sample in data0
54        dumb = data0;
55        dumb -= data1;           // Backward difference: data0 - data1
56        dumb += 128;           // Shift baseline up
57        if (dumb > 255) dumb = 255;
58        if (dumb < 0) dumb = 0;
59        output = dumb;
60        break;
61      }
62      PORTD = output;    // Output to the D/A via the parallel port D
63      INTCONbits.TMR0IE = 1;       // Enable timer interrupt
64    }
65    if (INTCON3bits.INT1IF == 1) {     // When the button (pin 34) is pushed, this ↩
        interrupt is called
66      INTCON3bits.INT1IE = 0;      // Disable interrupt
67      INTCON3bits.INT1IF = 0;      // Reset interrupt flag
68      if (buttondelay == 0) { // If buttondelay is not 0, it's a switch bounce
69        function++;   // Increment the function mode
70        if (function == 3) {
71          function = 0;   // Back to ECG simulation mode
72          output = 50;
73          mode = counter = 0;
74        }
75        LCD_update = 1;   // Signal the main program to update LCD
76        buttondelay = 200;  // Delay by 100 timer periods to debounce switch
77      }
78      INTCON3bits.INT1IE = 1;      // Enable interrupt
79      goto checkflags;     // Check again in case there is a timer interrupt
80    }
```

```
81 }
82
83 void Transmit(unsigned char value){       /************ Send an ASCII Character↩
       to USART ***************/
84   while(!PIR1bits.TXIF) continue;      // Wait until USART is ready
85   TXREG = value;           // Send the data
86   while (!PIR1bits.TXIF) continue;       // Wait until USART is ready
87   delay_ms (2);          // Wait for 2 ms
88 }
89 void ClearScreen(){          /******************* Clear LCD Screen ↩
       ************************/
90   Transmit(254);           // See datasheets for Serial LCD and HD44780
91   Transmit(0x01);          // Available on our course webpage
92 }
93 void backlight(unsigned char state){     /*************** Turn LCD Backlight on/↩
       off ******************/
94   Transmit(124);
95   if (state) Transmit(0x9D);     // If state == 1, backlight on
96   else Transmit(0x81);       // otherwise, backlight off
97 }
98 void SetPosition(unsigned char position){   /***************** Set LCD Cursor ↩
       Position  ******************/
99   Transmit(254);
100  Transmit(128 + position);
101 }
102 void PrintLine(rom unsigned char *string, unsigned char numChars){ /******** ↩
       Print character string ******************/
103   unsigned char count;
104   for (count=0; count<numChars; count++) Transmit(string[count]);
105 }
106 void PrintNum(unsigned char value, unsigned char position){ /*********** Print ↩
       number at position ******************/
107 unsigned char units, tens, hundreds;
108   SetPosition(position);           // Set at the present position
109   hundreds = value / 100;          // Get the hundreds digit, convert to ASCII ↩
        and send
110   if(hundreds != 0)Transmit(hundreds + 48);
111   else Transmit(20);             // If hundreds = 0, display a space
112   tens = value - hundreds * 100;       // Get the tens digit
113   tens /= 10;
114   Transmit(tens + 48);           // Convert to ASCII and send
115   units = value - hundreds * 100;      // Get the units digit
116   units -= tens * 10;
117   Transmit(units + 48);          // Convert to ASCII and send
118 }
```

```
119 void SetupSerial(){          /****** Set up the USART Asynchronous Transmit (pin ↩
      25) *******/
120   TRISC = 0x80;             // Transmit and receive, 0xC0 if transmit only
121   SPBRG = 25;               // 9600 BAUD at 4MHz: 4,000,000/(16x9600) - 1 = ↩
        25.04
122   TXSTAbits.TXEN = 1;               // Transmit enable
123   TXSTAbits.SYNC = 0;               // Asynchronous mode
124   RCSTAbits.CREN = 1;               // Continuous receive (receiver enabled)
125   RCSTAbits.SPEN = 1;               // Serial Port Enable
126   TXSTAbits.BRGH = 1;               // High speed baud rate
127 }
128 void main() /************************ main program ↩
      ******************************/
129 {
130   mode = function = counter = buttondelay = LED_count = 0;  // Initialize
131   output = 50;
132   LCD_update = 1;
133   TRISD = 0b00000000; // Set all port D pins as outputs (connected to D/A)
134   TRISB = 0b00000010; // RB1 as input for pushbutton, others outputs
135   SetupADC(0);        // Call SetupADC() to set up channel 0, AN0 (pin 2)
136   SetupSerial();      // Set up USART Asynchronous Transmit for LCD display
137   backlight(1);     // turn backlight on
138   ClearScreen();      // Clear screen and set cursor to first position
139   SetPosition(0);     // Set cursor position to the beginning of line 1
140   PrintLine((rom unsigned char*)"Hello BME361 :)", 15);
141   SetPosition(67);    // Go to beginning of Line 2
142   PrintLine((rom unsigned char*)"Team ???", 8); // Put your trademark here
143   delay_ms(255);      // Take a deep breath ...
144   delay_ms(255);
145   delay_ms(255);
146   delay_ms(255);
147   delay_ms(255);
148   delay_ms(255);
149   T0CON = 0b10001000; // Setup the timer control register for interrupt
150   // bit 7 = GIE - global interrupt enable
151   // bit 5 = TMR0IE - Timer 0 overflow interrupt enable
152   // bit 2 = TMR0IF - Timer 0 interrupt flag
153   INTCON = 0b10100000;
154   INTCON2bits.INTEDG1 = 0;        // Enable pin 34 (RB1/INT1) for pushbutton ↩
        interrupt
155   INTCON3bits.INT1IE = 1;       // Enable INT1 interrupt
156   ADCON1 = 0b00000101;//Make PORTB bits digital
157   while (1) {
158     if (LCD_update == 1) {
159       LCD_update = 0;
```

```
160         ClearScreen();
161         SetPosition(1);
162         PrintLine("Function: ", 10);
163         switch (function) {
164         case 0:
165           PrintNum(function, 11);
166           SetPosition(64);
167           PrintLine((rom unsigned char*)"ECG Simulation", 14);
168           break;
169         case 1:
170           PrintNum(function, 11);
171           SetPosition(64);
172           PrintLine((rom unsigned char*)"Echo", 4);
173           break;
174         case 2:
175           PrintNum(function, 11);
176           SetPosition(64);
177           PrintLine((rom unsigned char*)"Derivative", 10);
178           break;
179         }
180       }
181       LED_count++;
182       LED_count1 = LED_count & 0xF0;
183       PORTB = LED_count1;
184       delay_ms(62);    // LED count to RB4-RB7 at 1 Hz
185     }
186   }
```

Table of ASCII Codes:

ASCII Table

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

Figure 4.2: ASCII table for generating letters, numbers and symbols.

Lab 5: Introduction to Soldering: ECG Printed Circuit Board

If you have never used a soldering iron before please do a few practice rounds before starting this lab
PURPOSE: Introduce students to the use of a soldering iron.
GOAL: Today you will be creating an ECG amplifier on a previously designed printed circuit board. To complete this task you must create the circuit shown in the schematic on a printed circuit board. Each group will be given a set of leads which will be connected via electrodes to one member of your group. This will enable you to collect real data which you will be able to view on an oscilloscope.

You will need to build the circuit by following the procedure portion of the lab. Once you have the ECG PCB completely soldered, you may move onto the Tasks portion of the lab.

MATERIALS:

- Printed Circuit Board (P.C.B.)
- Soldering Iron Station
- All components listed in table 5.1
- ECG Leads (Right Arm, Left Arm, Left Leg)
- Battery and battery clip
- Auxiliary jack connector ($\frac{1}{4}$" audio jack)

COMPONENTS:

- (2) 8-Pin Sockets (U3 and U4)
- 14- Pin Socket (U1)
- Mono Audio Jack
- 2 and 4 pin Headers

R1	27K	Red Vio Org	C1 *	226 nF
R2	1K	Brw Blk Red	~~C2 *~~	
R3	27K	Red Vio Org	~~C3 *~~	
R4	1K	Brw Blk Red	~~C4 *~~	
R5	100K	Brw Blk Yel	C5	47 uF
R6	22K	Red Red Org	C6	100 uF
R7	330K	Org Org Yel	~~C7 *~~	
R8	10K	Potentiometer	C8	1 uF
R9	10K	Brw Blk Org	C9	1 nF
R10	10K	Brw Blk Org	C10	100 nF
R11	10K	Brw Blk Org	C11	100 nF
R12	10K	Brw Blk Org	C12 *	10 nF
R13	12K	Brw Red Org	U1	LM 324
R14	27K	Red Vio Org	~~U2 *~~	
R15	27K	Red Vio Org	~~U3~~	~~LM 7660~~
R16	5.1K	Grn Brw Red	~~U4~~	~~AD 620~~
R17	3.3K	Org Org Red	~~U5 *~~	

Figure 5.1: Table of parts. Note that some parts listed on the PCB are no longer used - they are crossed out in this table.

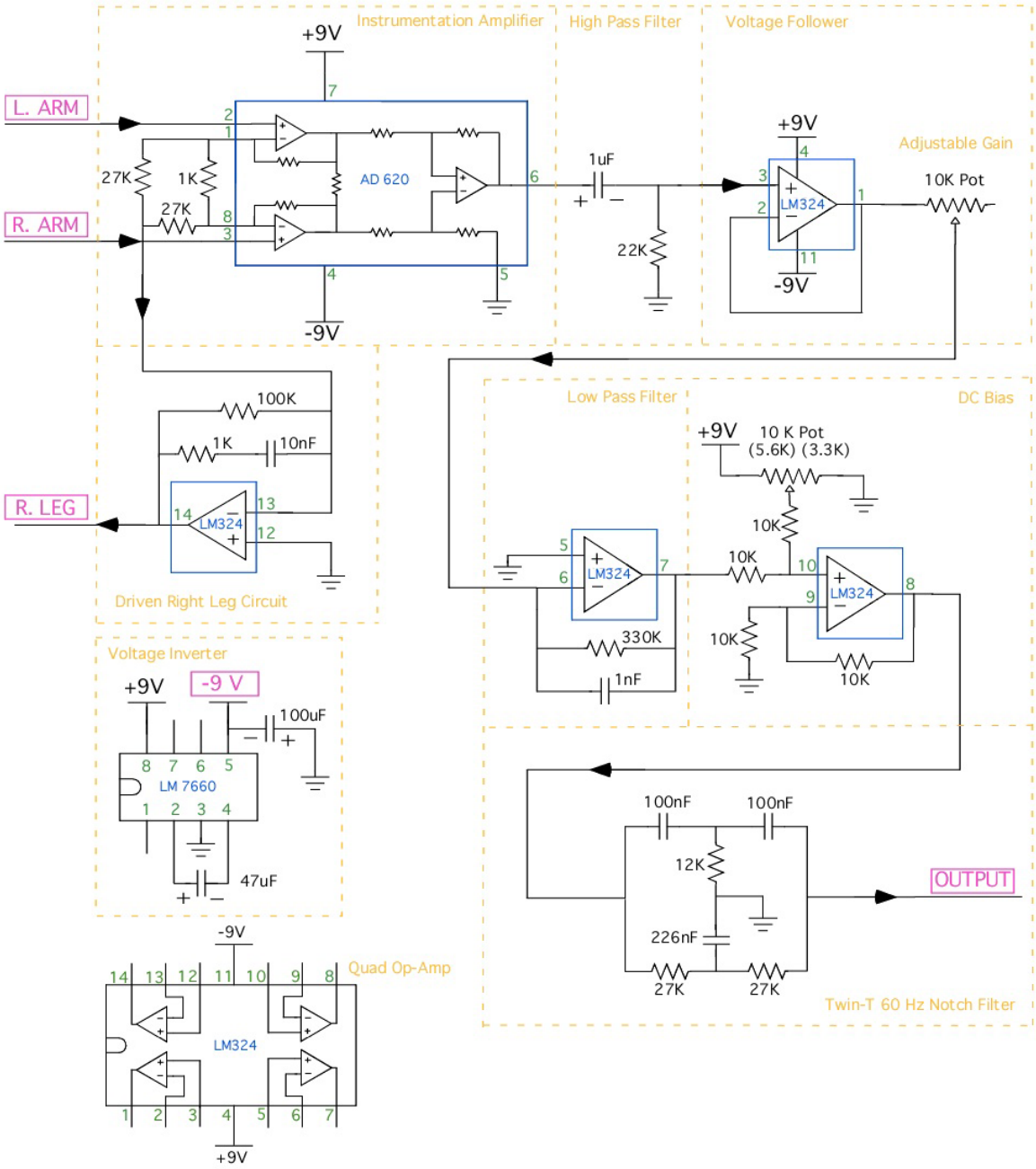


Figure 5.2: Schematic of PCB layout.

PROCEDURE:

1. Using the Printed Circuit Board you are given, create the circuit seen in the schematic. To complete this circuit all you need to do is find the component on the schematic, match it up to the corresponding label on the P.C.B., and solder the component to the board.

Note 1: You will NOT be soldering the larger chip components (324, 741, 7660, 620) directly to the PCB - instead you will be soldering 8 and 16-pin sockets to the board and then inserting the components into the sockets when you have completed soldering the board. *Note 2: The values of R1 and R3 (27 kΩ) must be exactly matched - be sure to check these resistors before soldering!

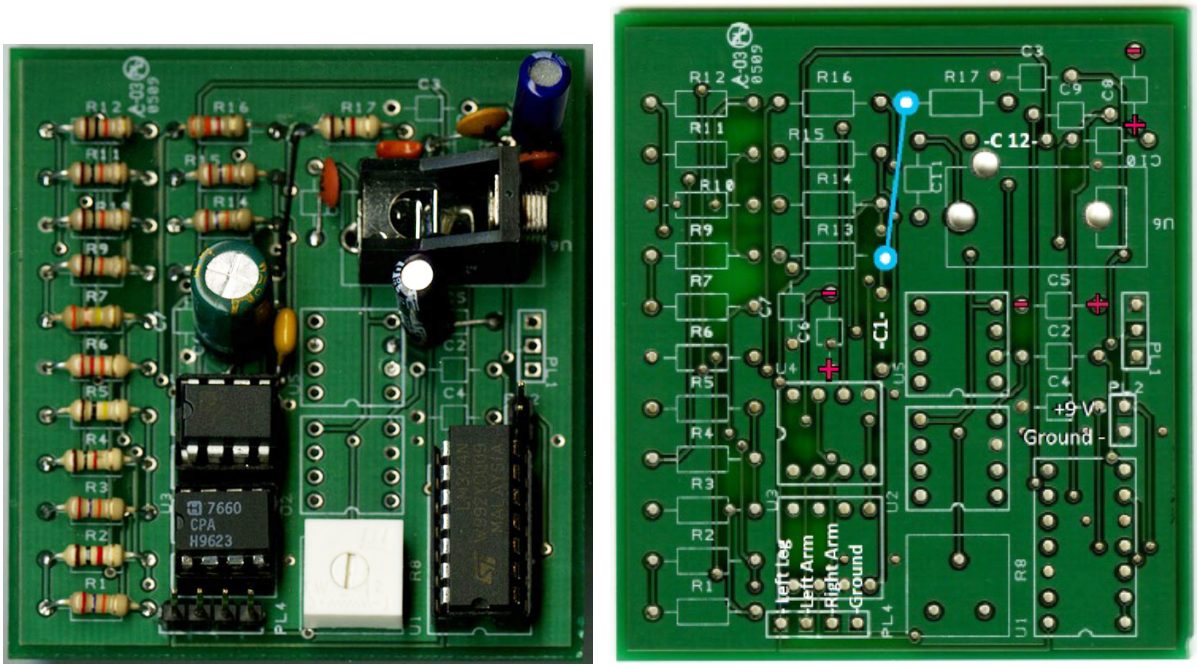
2. Make sure you have all the necessary components, and that they are oriented correctly.

Note: if you do not know/remember the pin assignment for a certain component, look up the datasheet and use it as a reference.

3. There are four jumper wires that need to be added to the circuit: one grounding wire on the front, and three signal wires on the back.

(U5, Pin 2 → U1, Pin 9) (U5, Pin 3 → U1, Pin 10) (U5, Pin6 → U1, Pin 8) (R13, Right side → R17, Left side)

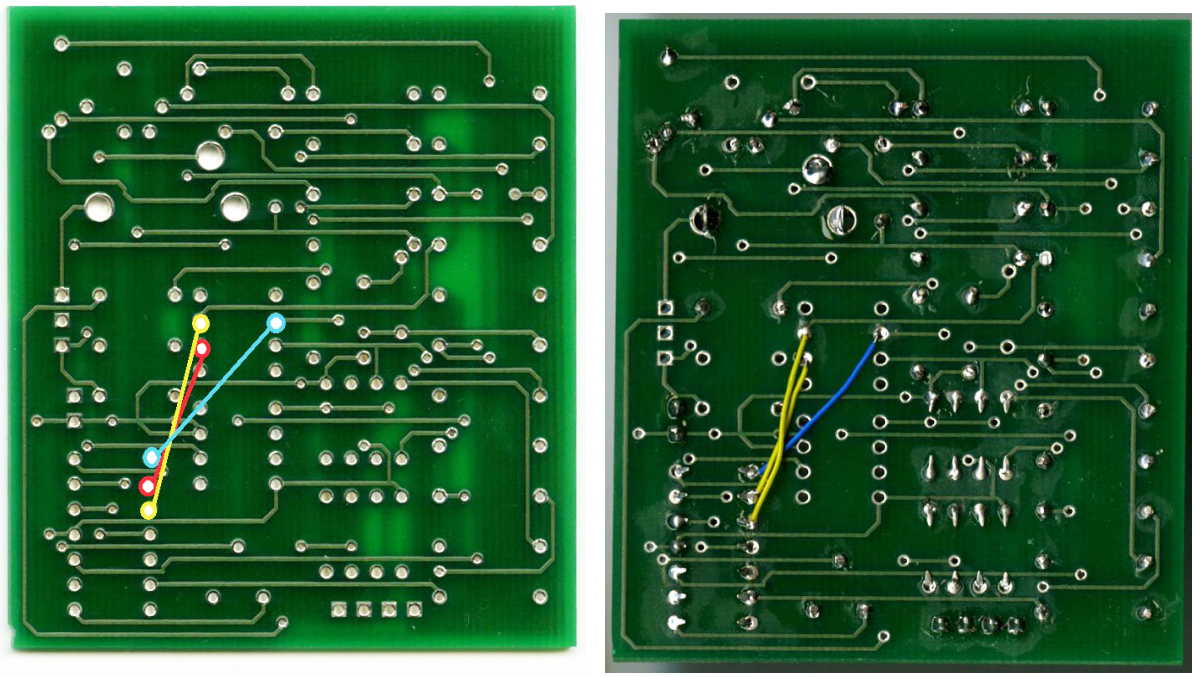
*There are four capacitors, and two chips that are no longer used. These should be left blank. Capacitors 1 and 12 are also not labeled on the PCB but are labeled in figure 5.3a. These should be what the final product looks like.



(a) Populated PCB. Note the wire placement.

(b) Blank PCB shown with the wire placement and the capacitor orientations.

Figure 5.3: PCB images with and without components.



(a) Note the placement of the wires you'll need to solder.

(b) The soldered wires.

Figure 5.4: Back of PCB images without components. Note wire placement.

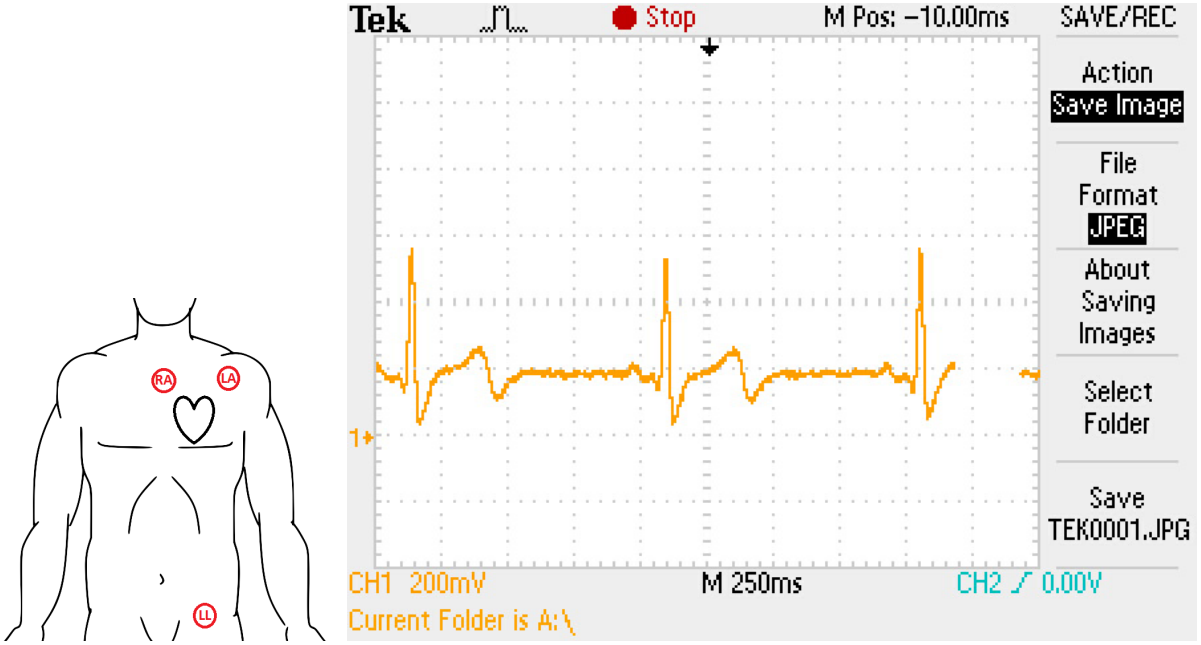
TASKS:

1. Now you can connect the leads to your p.c.b. To do this you need to attach electrodes to you or one of your group members. The leads will be connected on the right arm, left arm, and left leg. Figure 5.5a above indicates the most successful diagram, but you may want to vary these slightly to obtain best results. These leads are attached to the alligator clips, then to the four-pin connector on the p.c.b.

2. In order to see a visible ECG, the circuit needs to be attached to the oscilloscope. Connect the socket end of the audio jack to the p.c.b, and the other end to the oscilloscope.

3. Last, connect your battery to the p.c.b.

4. You should now be able to see a functional ECG, similar to the one shown in figure 5.5.



(a) Recommended electrode placement.

(b) Oscilloscope showing waveform.

Figure 5.5: Electrode attachment recommendation and resulting ECG waveform.

Lab 6: Digital Filters: Low Pass, High Pass, Median, and 60 Hz Notch

PURPOSE: To create digital filters using C++ coding instead of building them on your breadboard. The advantages to using digital filters are that they can be easily designed, tested, and changed without affecting the circuitry (hardware), whereas an analog filter can only be changed by redesigning the circuit. Another advantage is that digital filters are extremely stable and are not affected by the external environment and subject to temperature or component error like analog filter circuits.

GOAL: Today you will be adding code for High Pass, Low Pass, Median, and 60 Hz Notch filters to your previously existing code. To do this you must first add the given Low Pass Filter program to your master program and then modify it to produce the three programs described in the Tasks portion of the Lab.

PROCEDURE:

Hardware

1. You will be using both the circuit and p.c.b. from Lab 5.

Software

1. Open MPLAB and add Low Pass Filter program to your already existing master program.

TASKS:

1. You were given the code for a Low Pass Filter - now you need to modify that code to create High Pass, Median, and 60 Hz Notch Filters.

Low Pass (LP) Filters eliminate all frequencies above the predetermined cut-off frequency; while leaving the frequencies below the cut-off unchanged.
High Pass (HP) Filters are the opposite of a LP - they eliminate all frequencies below the predetermined cut-off frequency; while leaving the frequencies above it unchanged.

Median Filters are smoothing and moving filters which are used to eliminate noise while preserving the edge values of your data.
60Hz Notch Filters also eliminate noise but only noise at 60 Hz.

All of these filters will help clean up your ECG signal.

2. Once you have finished the three new filters you can add them to your master program.

3. You also need to add four new modes to your program for the LP, HP, Median, and 60 Hz Notch filters, as well as add some code for the LCD Display.

4. Once you have programmed your code you can connect leads to one of your group members and run the signal through your ECG p.c.b. and then through your four new filters. The pushbutton will allow you to switch between filters.

5. You should be able to view and ECG waveform after it has run through your filters on the oscilloscope.

Lab 7: QRS Detection

PURPOSE: To brings students one step closer to a fully functional heart rate meter. With the use of a buzzer and the QRS code, students will hear a beeping sound every time a peak is detected. By doing this, we will be almost finished creating a heart rate meter similar to the ones used by hospitals.

GOAL: To combine the ECG printed circuit board you created in Lab 5, with a previously written QRS Detection Program. This program was written to detect the QRS peak of an ECG waveform; the program sends a signal through the PIC to a Buzzer and LED every time a certain threshold has been reached. This will create the "beepĚbeepĚbeep" you hear on a hospital heart rate meter.

MATERIALS:

- TDB-12PN Buzzer
- LED
- 2N2222 NPN Transistor
- 470 Ω Resistor
- 10 kΩ Resistor

SCHEMATIC:

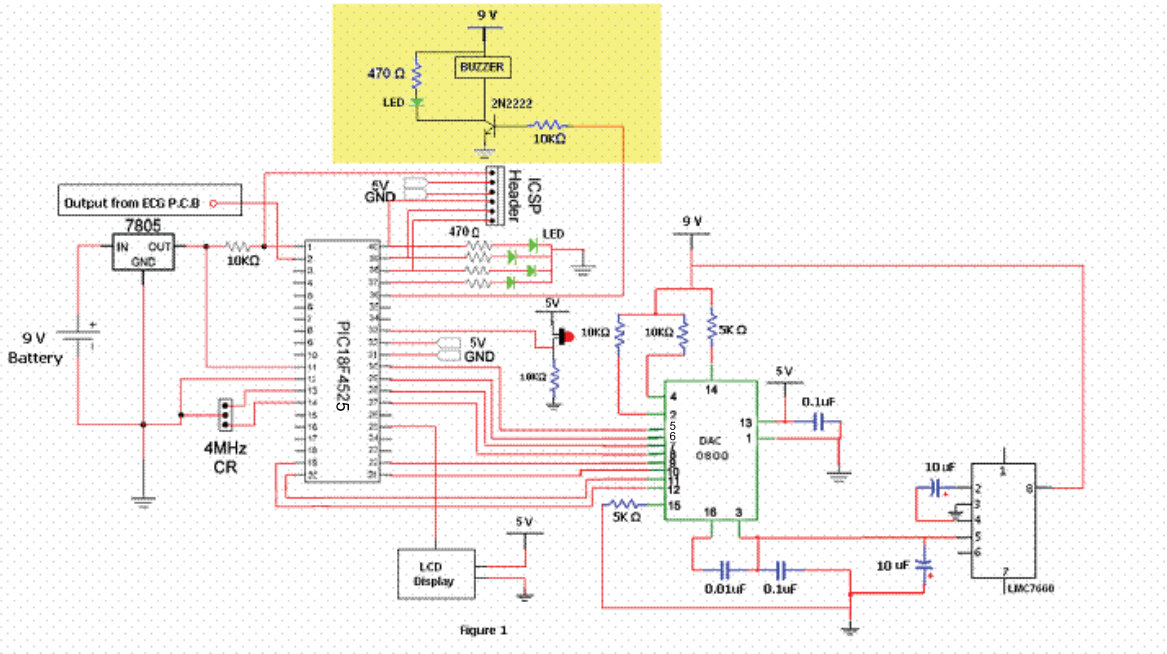


Figure 7.1: Schematic of completed circuit.

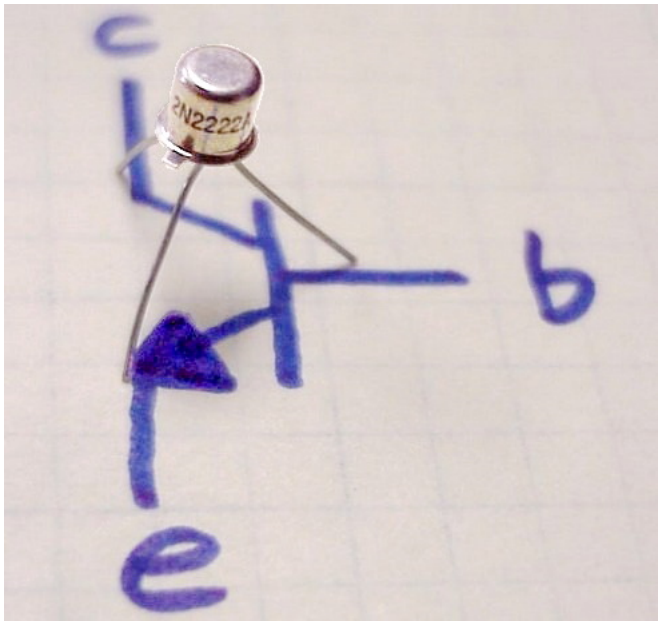


Figure 7.2: 2N2222 Transistor

PROCEDURE:

Hardware

1. Using your breadboard, add the highlighted section seen in the schematic to your previously ex-

isting circuit.
NOTE: The Buzzer is very loud, so in order to mute this device it is advisable that you cover the top hole with masking tape or another similar material.
Software

1. Open MPLAB and add the QRS Detection program to your master program.

2. Connect theMPLAB ICD3 programmer to the ICSP Header and to the PC, and download the program onto the PIC.

3. Disconnect theMPLAB ICD3 programmer from the ICSP Header.

TASKS:

1. Now that the output from your ECG p.c.b. is connected to pin 2 of the PIC, you can attach the ECG leads to one of your group members and the signal should run through the ECG circuit and then through the QRS Detection Program in your PIC. If your circuit is functioning correctly your LED and Buzzer will be turned on and off continuously.

2. You can now test your circuit using an oscilloscope; connect channel 1 at the output of your ECG p.c.b., and channel 2 at pin 36 of the PIC. This will show you the ECG signal as well as the function of the QRS Detection Program. **NOTE: The group member who is connected to the ECG should remain as still as possible.**
You will be using an algorithm known as the Multiplication of Backward Differences. As the name implies, rather than running a forward derivative, we'll be performing a backward derivative (in discrete time, a derivative is well approximated by a difference, i.e. a subtraction.) Those differences are then multiplied, providing a very robust peak detector. The following code will help implement the algorithm.

Lab 8: Heart Rate Meter

PURPOSE: In this lab you will be combining all the work you have done throughout the semester to create a functional heart rate meter with an LCD screen that displays eight separate modes.

GOAL: To create the final mode of your project and combine it with all previous components.

PROCEDURE:

Hardware

1. You will be using the circuit you constructed in Lab 7. If you would like to view a larger version of the schematic it can be found in the file Final Schematic.

TASKS:

1. You will now need to add another mode for the Heart Rate Meter; this mode will correspond to the data from the QRS Detection. You will also need to write code for the LCD Display:

To do this you must write a function to take points from the input signal, create three consecutive positive differences, take the average, and then create an algorithm which uses this data to display the beats per minute on your LCD screen.

At this time your program should include the following eight modes:

1. ECG Simulation 2. Echo 3. Derivative 4. LP Filter 5. HP Filter 6. Median Filter 7. 60 Hz Notch Filter 8. Heart Rate Meter
The names of modes 1 -7 should be displayed on the LCD screen as you switch between modes. Mode 8 should display the name, as well as beats per minute.

Appendix A: Introduction to MPLab - Learning Exercise

A.1 Creating a New Project in MPLAB

In the MPLABX start-screen, click the `Quick Start` icon and follow the instructions.

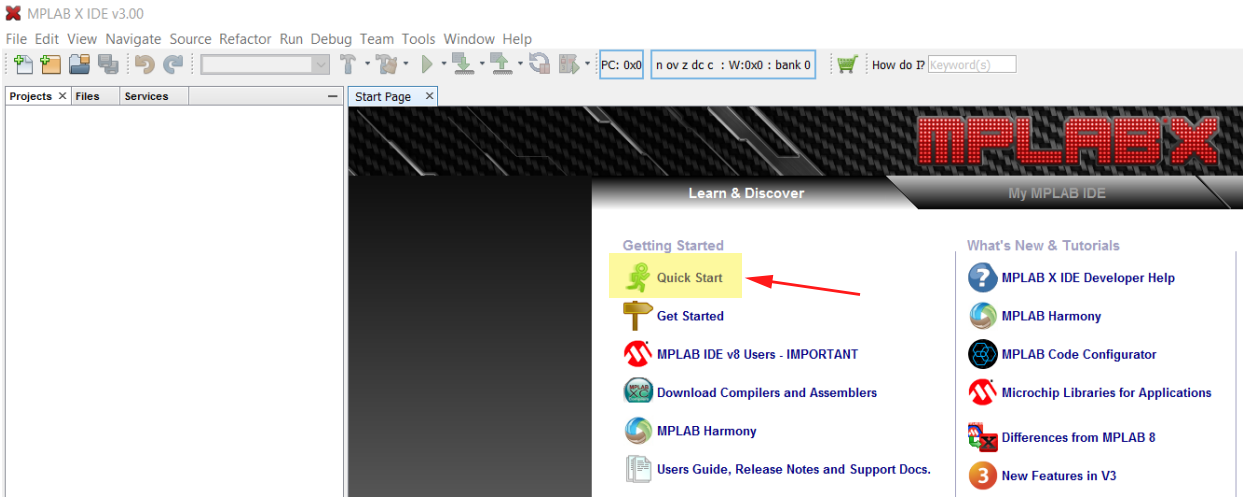


Figure A.1: There are comprehensive instructions to guide you to creating your first project successfully.

A.2 Constructing your Circuit and Connecting the Programmer

A schematic for the circuit is shown below:

Note: Before connecting the PIC, plug in the battery and test to make sure the correct voltages are visible at the appropriate locations. A clear sign that the wrong voltage is being supplied to the PIC is the D/A converter will heat up very rapidly. Supplying the wrong voltage to the PIC can permanently damage it, so continually check voltages throughout your use to prevent this from happening.

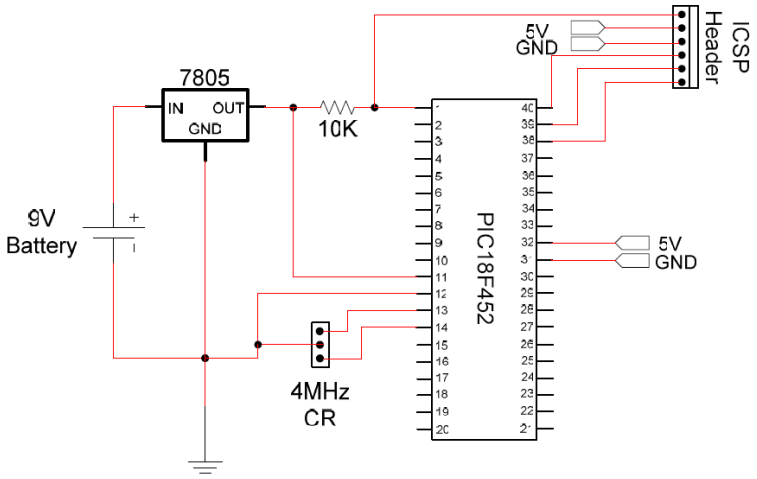


Figure A.2: Minimum working example (MWE) circuit.

Appendix B: Oscilloscopes 101

By now, you have undoubtedly seen an oscilloscope. You may even have played a bit with more of the knobs and buttons than just the auto-set. However, you may not know how to set all the parameters to insure that you will see the signal you're interested in seeing. For instance, having the oscilloscope set to AC coupling instead of DC coupling will give a completely different waveform. If that last sentence has you scratching your head, keep reading.

There is a fairly comprehensive tutorial written by the folks at Tektronix (the company that makes the oscilloscopes we use) at `www.tek.com/Measurement/App_Notes/XYZs/03W_8605_2.pdf`. It's 64 pages long and worth a look. As shorter, more compact reference can be found at `http://oscilloscope-tutorials.com/Oscilloscope/controls.asp`. We'll address some of the key points and most often overlooked settings when using an oscilloscope - and keep it to just a few pages.

B.1 Time Scale

We begin with the time scale setting since this is the one that is most likely to be misunderstood. Each block on the vertical axis of the oscilloscope (there are grid lines in the horizontal and vertical axes, called ticks) has a time scale. This means that each block represents a certain period of time. If the time scale is set to 10 microseconds and there are 10 blocks on the screen, the signal on the screen represents 100 microseconds of the signal. If the signal you are interested in viewing is very long, upwards of a second (such as a QRS complex), it would be necessary to set the time scale resolution to 0.1 seconds in a 10 block window.

B.2 Volt Scale

Obviously, the volt scale allows you to make the signal you are viewing appear bigger or smaller as a result of changing the scale. But what often presents a problem is that the probe you are using may have a gain setting of 10 times the signal you are using. This is handy if you are measuring really low amplitude signals, nanovolts or less. But for most signals, your probe should be set to 1 times gain. Regardless of which setting you are using, make sure the oscilloscope setting matches the same value as the probe.

B.3 Coupling

There are three types of coupling: AC, DC, and ground coupling. AC coupling allows you to see a signal at 0 mean (no DC bias or offset) and allows for faithful representation of an oscillatory signal. If one

were to have a pure sinusoid that was riding on a 5 volt DC bias and viewed the sinusoid in AC coupling mode, the signal would appear on the scope to be oscillating about 0 volts. Use this mode if all you want to view is the oscillatory pattern.
Similarly, if you are viewing DC signals, i.e. signals composed of discrete DC values, you should use DC coupling. This does not mean that you will not see oscillatory patterns. On the contrary, if your signal has oscillations but are described at specific DC offsets (like a simulated QRS complex) you will be able to determine where the signal is rising and falling against its DC offset. If one were to look at such a signal in AC coupled mode, the signal would oscillate about 0 volts and would look different than the DC mode.
Ground coupling literally ties the signal you are trying to measure to ground. It really is just a way to measure your signal against ground. Remember, ground is a relative term, it may not be 0 volts.

B.4 Trigger

Triggering is very helpful when you want to see a stable waveform. Often, digital oscilloscopes have difficulty sampling a high-frequency signal fast enough to maintain the waveform structure (fig. B.1). By setting the moving the trigger cursor (ellipse) up to the waveform, the oscilloscope freezes the waveform at the first rising edge encountered by the cursor (fig. B.2). This oscilloscope, like others, allows for a host of triggering options.
Similar results can be achieved with the run/stop button. The problem with this approach is that there is a delay between the time you press the button and the capture. This leads to the possibility of capturing a portion of the waveform that was not intended.

B.5 Autoset

The longtime favorite of students around the globe. The oscilloscope's software analyzes the signal, looking for DC bias, rising and falling edges, and frequency content. Once it has these parameters calculated, it sets the oscilloscope tick spacing (in both voltage and time) and sets the correct type of coupling, which will generally either be AC or DC. It MAY NOT be the best way to view your signal, but can often get you close so that your "tuning" will be minimized.

B.6 Other Functions

Some oscilloscopes have a built-in math function that allows you to perform some type of transformation, such as a Fourier transform (fig. B.3). You typically also have control over where to place your cursors to allow you to reference your signal in some tighter interval. The more expensive the oscilloscope, the more options. Measuring very fast signals (high frequency, maybe transient) with high bandwidth often means very expensive oscilloscopes. For instance, to measure high frequency laser light (10s of gigahertz) it is not uncommon for oscilloscopes to cost upwards of $100k.

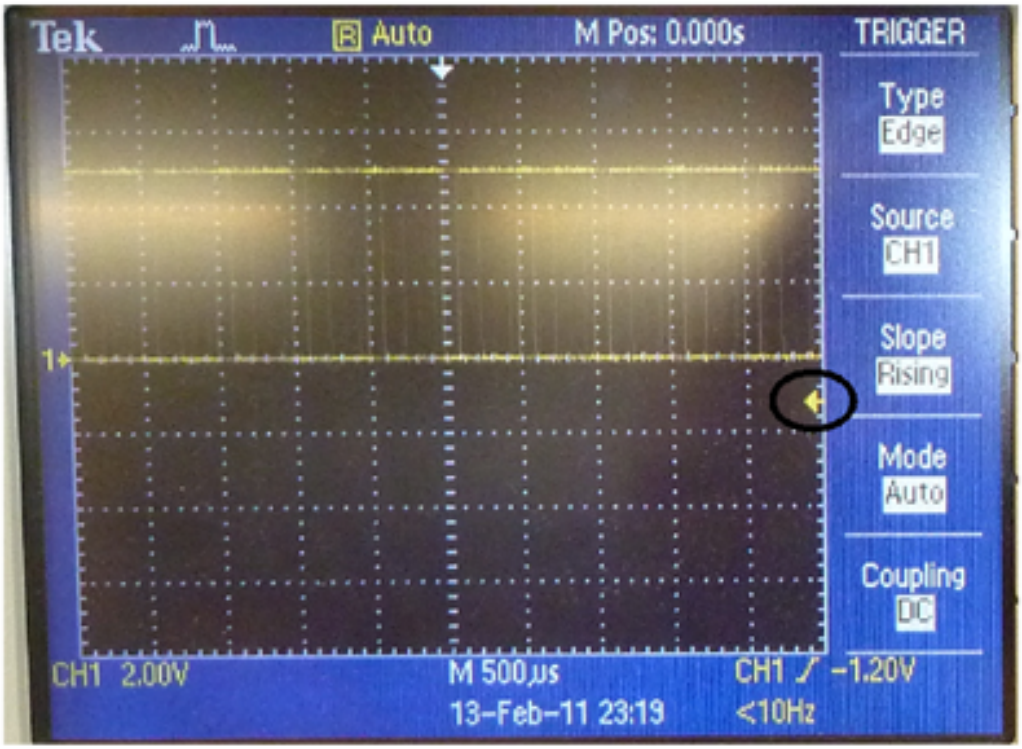


Figure B.1: Waveform appears smeared or blurred.

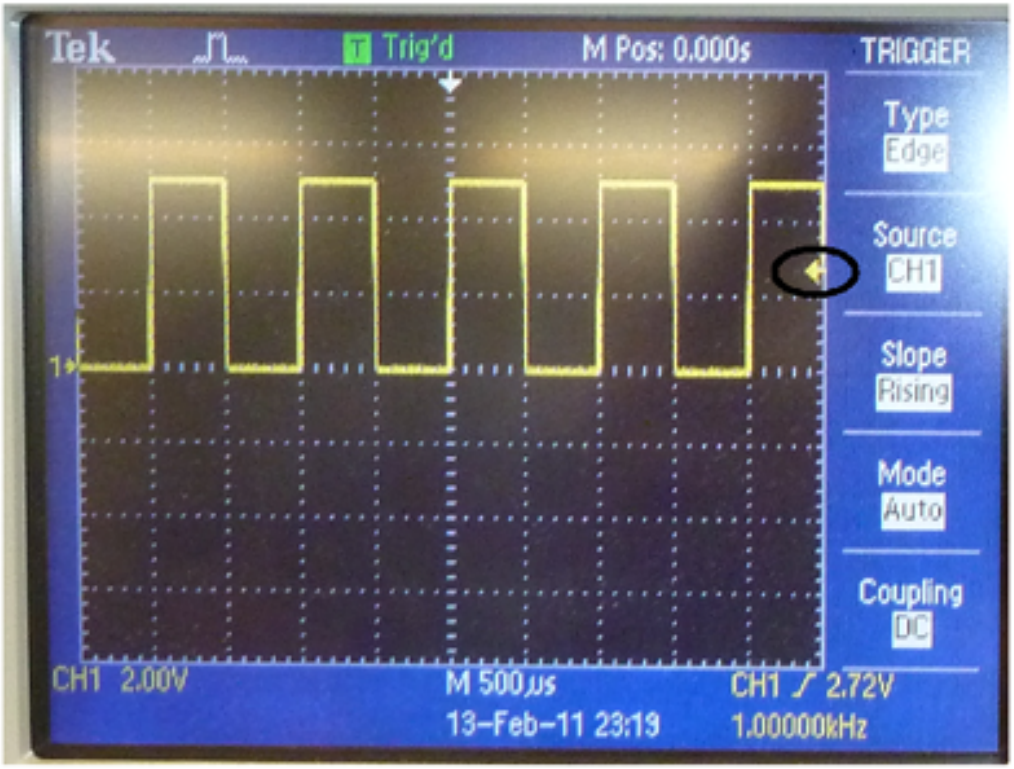


Figure B.2: Stable waveform.

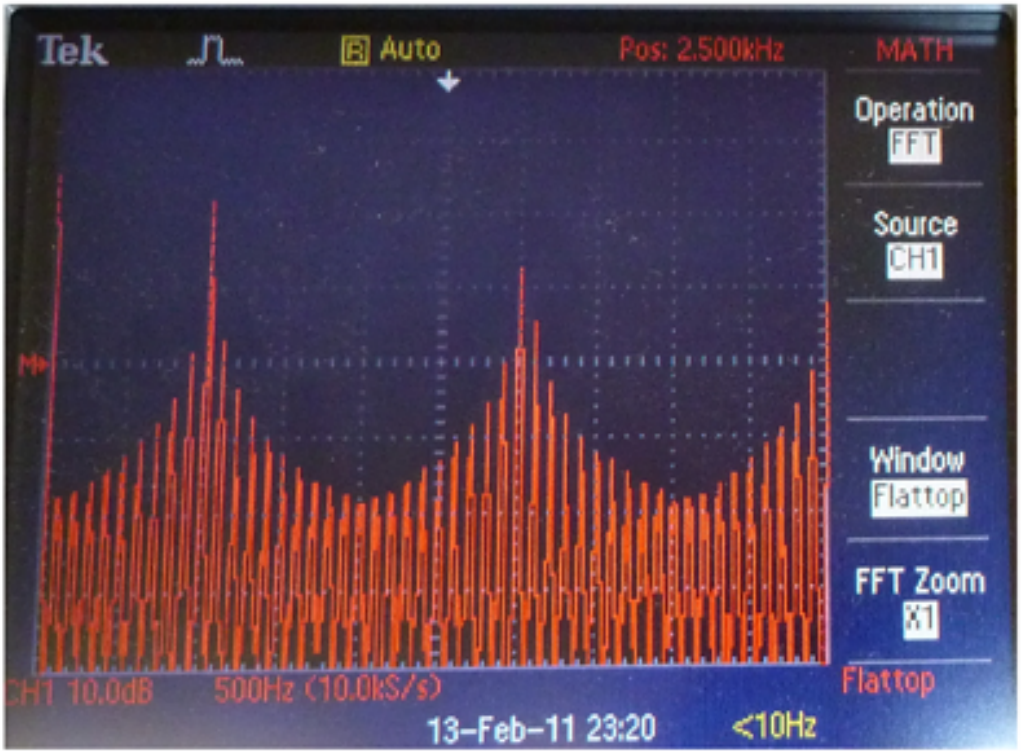


Figure B.3: Fourier transform of the signal in figure B.2. Notice that the time scale is set to 500 Hz per block and the first peak occurs at the second block, i.e. 1000 Hz. This is to be expected as the signal is a 1 kHz square wave. The other peaks are referred to as the harmonics and occur at ODD integer multiples of the fundamental.