

Contents

1	Introduction	2
1.1	Features and specifications	2
1.2	Theory of operation	2
2	Usage	3
2.1	Basic usage	3
2.2	Tips for improved performance	3
3	Hardware Information	4
3.1	Block diagram	4
3.2	Schematics	5
3.3	Bill of materials	8
3.4	Board layout	9
3.5	Tuning procedure	11
4	Firmware Information	12
4.1	Calculation overview	12
4.2	Calibration procedure	12
4.3	Code listing	12

1 Introduction

1.1 Features and specifications

The ultrasonic rangefinder is an easy to use handheld device for measuring distances without needing to approach a target. A built in laser pointer helps locate the intended target. The device reads out distances in inches, centimeters, feet, and meters so there is no need to perform unit conversion.

Features

- Handheld cordless single button operation.
- Runs on a single 9V battery.
- Integrated laser pointer for easy target finding.
- Displays distances in 4 different units.

Specifications

Parameter	Value
Measurable distance	1 – 15 ft.
Minimum accuracy	±1 in.
Battery life (assuming 550 mAH alkaline)	3.9 hours continuous operation
Operating temperature	32 – 100°F (0 – 38°C)
Weight (without battery)	0.5 lbs.
Dimensions	$7\frac{5}{8} \times 3\frac{5}{8} \times 1\frac{1}{2}$ in.

1.2 Theory of operation

The device emits an ultrasonic burst and measures the time it takes the sound wave to travel to the target object and reflect back. Given the speed of sound in air and the time of flight, it is possible to compute the distance. The speed of sound in air is 1116 ft/s at sea level, so to achieve a resolution of at least 0.1 in, it is necessary to have a counting timebase faster than 130 kHz. This ultrasonic rangefinder counts faster than 500 kHz allowing high precision measurements.

2 Usage

2.1 Basic usage

Pressing and holding down the button on the front of the button will turn on the device and begin the measurement process. After pressing the button, use the laser pointer to align the device to the target object. The measurement should appear on screen after approximately one second. Distance is measured from the front of the transducers.

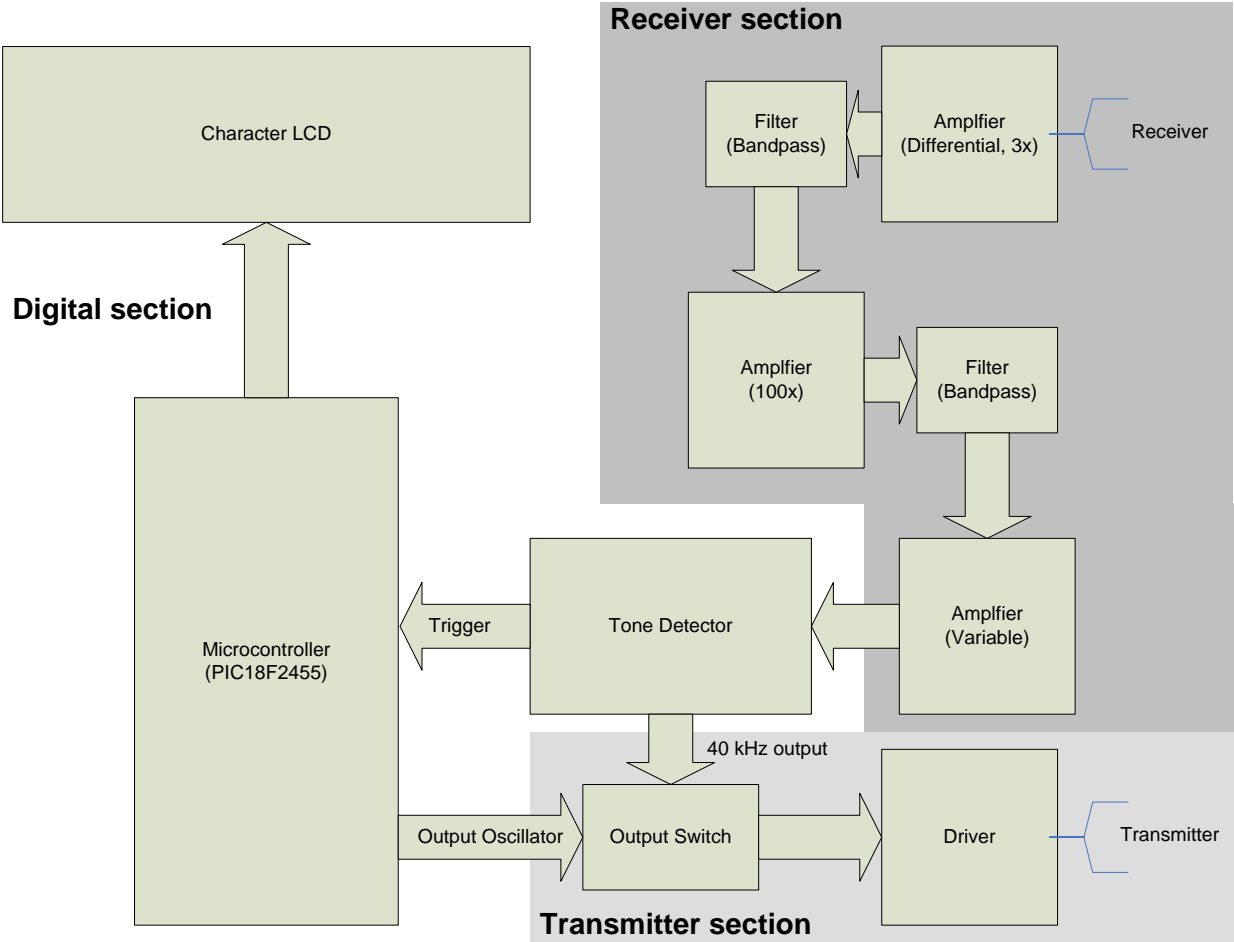
2.2 Tips for improved performance

- Whenever possible, try to measure distance to flat surfaces that are directly facing the transducers. This minimizes stray reflections from nearby objects that may cause spurious triggers and therefore lead to incorrect or unintended distances measured.
- When measuring distances to ends of enclosed spaces, keep the transducers pointed directly into the cavity to avoid reflections from the walls.
- Avoid moving the device when making measurements, since this can lead to incorrect measurements.

3 Hardware Information

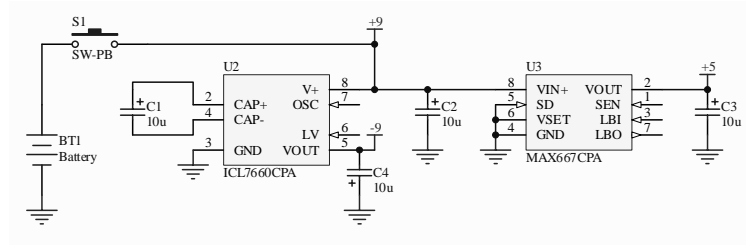
3.1 Block diagram

The device structure is separated into three distinct portions: receiver, transmitter, and microcontroller. Below is the overall system block diagram.



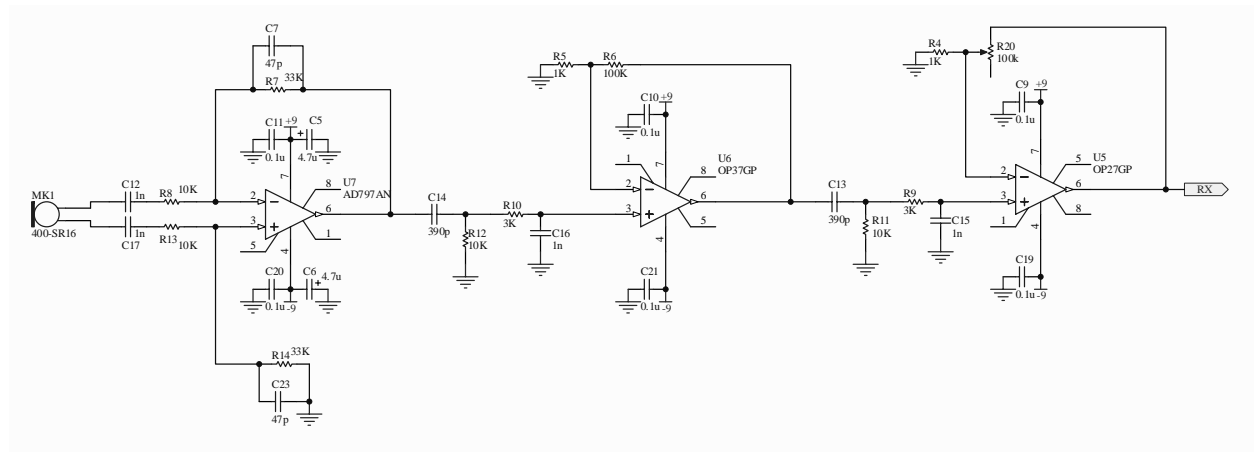
3.2 Schematics

Power supply



The power supply accepts an input voltage between 7V to 10V, so an ordinary alkaline 9V battery is suitable to power the circuit. A rechargeable 9V battery tends to produce only an 8V output, which is also acceptable. A negative rail for opamps is generated with the ICL7660 charge pump. A regulated 5V digital supply is produced using the MAX667 so that low battery voltages still allow operation of the circuit due to the low dropout characteristics of the regulator.

Receiver section



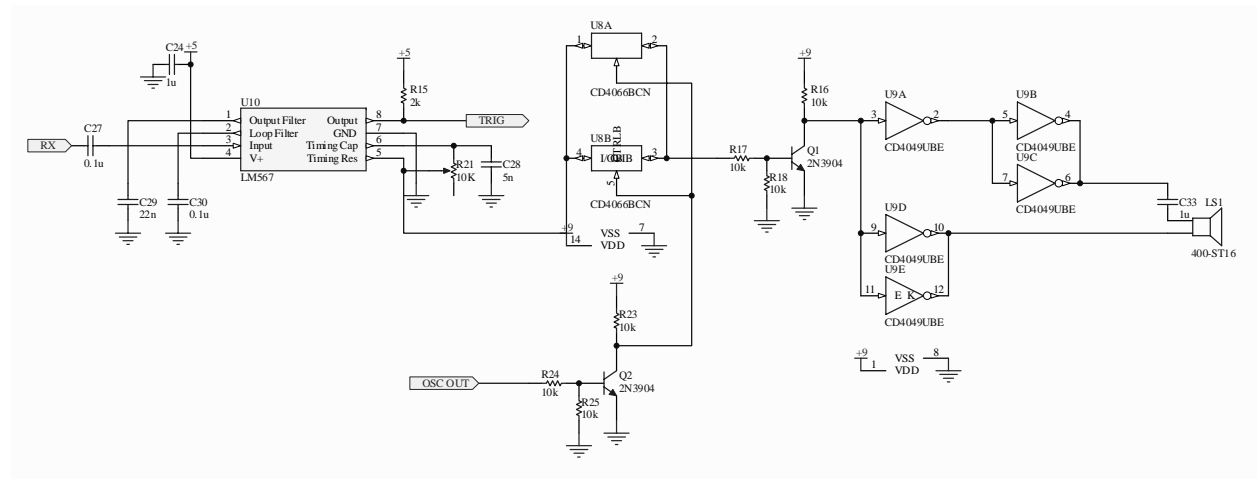
The receiver section consists of three gain stages. The first is a differential amplifier with a gain of 3 to reduce common mode noise. An ultralow noise opamp with complete bypass capacitors is used to further limit input noise.

The second and third gain stages are simple non-inverting amplifiers with gains of 100 and up to 100 respectively. The tunability of the final stage allows the input threshold level to be set.

Between each amplifier stage is a bandpass filter centered at 40 kHz consisting of cascaded low and high pass filters. At the input stage, there is additional filtering at the input and in the feedback loop to further cut out high frequencies. In total, the filters provide 60 dB/decade of rejection on either side of the passband.

The received signal is also AC coupled into the LM567 detector, and the output pin sinks the trigger pin to the microcontroller. The LM567 is set to have an input bandwidth of 5.6 kHz, which increases the sensitivity threshold and decreases the number of cycles before output; both desirable characteristics.

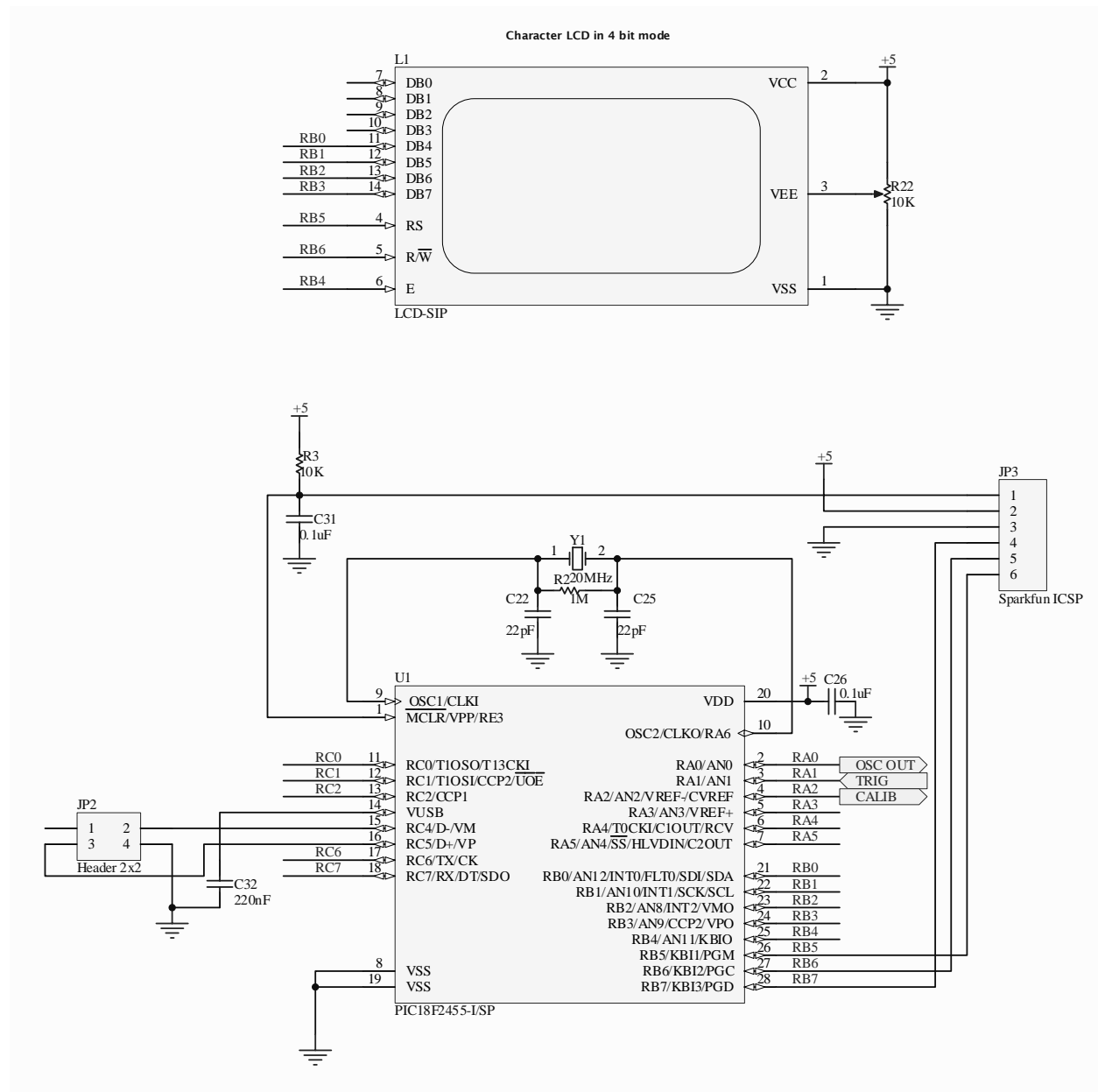
Transmitter section



The heart of the transmitter is the LM567 tone detector which contains a phase locked loop. The output of this tunable oscillator is passed through an analog switch so that the microcontroller may control the pulse width. The pulse width is then level shifted to the supply rail voltage for the CMOS driver circuit. The driver consists of two sets of two parallel CMOS buffers for added power, with one branch inverted relative to the other to achieve voltage doubling. When passed through a capacitor, the final voltage reaching the transmitter is $18 V_{PP}$.

The microcontroller outputs a control line to allow the oscillator output. This is level shifted to 9V to control the analog switch for better switching rejection.

Digital section



The microcontroller is a PIC18F2455 Flash based processor with onboard RAM and support of the C programming language, greatly simplifying design and development time. The MCU is clocked at 24MHz internally, with a net throughput of 6MIPS. The character LCD is connected in 4 bit mode on Port B. The control signals to the analog interface are provided on Port A. Additionally, in circuit serial programming is used to rapidly test new designs.

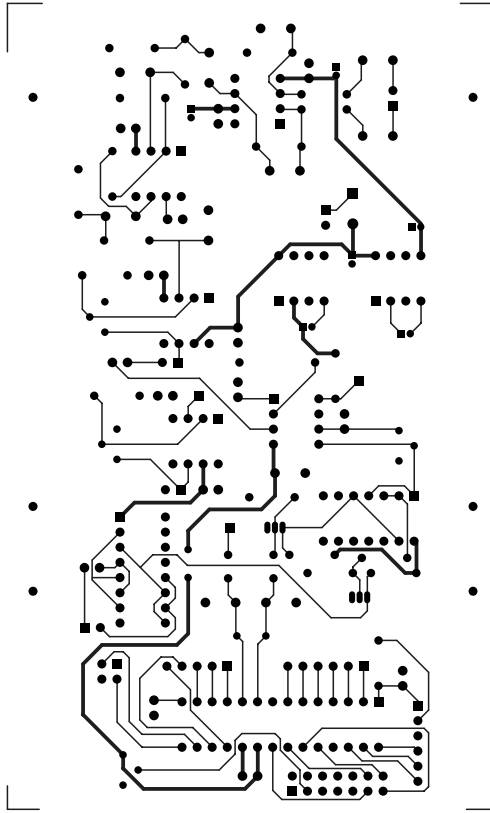
3.3 Bill of materials

Description	Designator	Value
Crystal Oscillator	Y1	20Mhz
Potentiometer	R20	100k
Potentiometer	R21	10K
Potentiometer	R22	10K
Switch	S1	
400-ST16 Ultrasonic transmitter	LS1	
400-SR16 Ultrasonic receiver	MK1	
OP27GP Low-Noise, Precision Opamp	U5	
OP37GP Low-Noise, Precision, High-Speed Opamp	U6	
AD797AN Ultra-Low Distortion/Noise Opamp	U7	
2x16 LCD with SIP connector	L1	
Header, 2-Pin, Dual row	JP2	
Header, 6-Pin	JP3	
PIC18F2455-I/SP Enhanced Flash Microcontroller	U1	
CD4049UBE Hex Inverting CMOS drivers	U9	
CD4066BCN Quad CMOS bilateral switch	U8	
LM567 Tone Detector	U10	
ICL7660CPA CMOS Voltage Converter	U2	
MAX667CPA 5V Voltage Regulator	U3	
Capacitor	C12	1n
Capacitor	C13	390p
Capacitor	C14	390p
Capacitor	C15	1n
Capacitor	C16	1n
Capacitor	C17	1n
Capacitor	C22	22pF
Capacitor	C23	47p
Capacitor	C24	1u
Capacitor	C25	22pF
Capacitor	C7	47p
Capacitor	C10	0.1u
Capacitor	C11	0.1u
Capacitor	C19	0.1u
Capacitor	C20	0.1u
Capacitor	C21	0.1u
Capacitor	C26	0.1uF
Capacitor	C27	0.1u
Capacitor	C28	5n
Capacitor	C29	22n

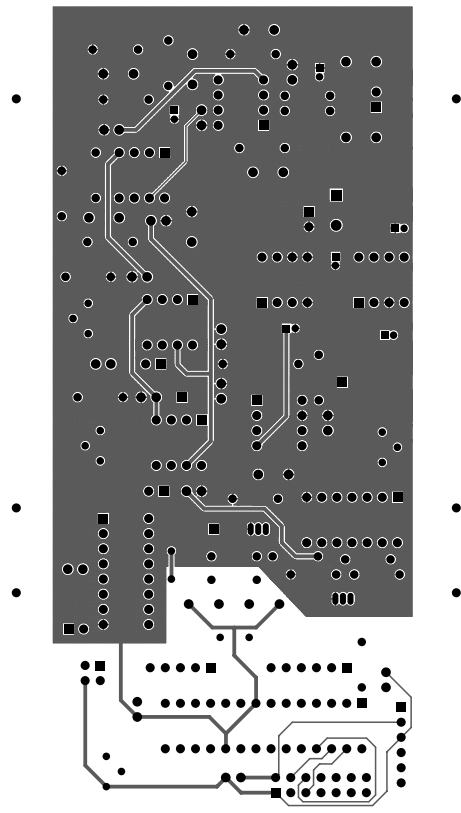
Capacitor	C31	0.1uF
Capacitor	C32	220nF
Capacitor	C33	1u
Capacitor	C9	0.1u
Polarized Capacitor (Radial)	C1	10u
Polarized Capacitor (Radial)	C2	10u
Polarized Capacitor (Radial)	C3	10u
Polarized Capacitor (Radial)	C4	10u
Polarized Capacitor (Radial)	C5	4.7u
Polarized Capacitor (Radial)	C6	4.7u
2N3904 NPN General Purpose Amplifier	Q1	
2N3904 NPN General Purpose Amplifier	Q2	
9V Battery	BT1	
Capacitor	C30	0.1u
Resistor	R10	3K
Resistor	R11	10K
Resistor	R12	10K
Resistor	R13	10K
Resistor	R14	33K
Resistor	R15	2k
Resistor	R16	10k
Resistor	R17	10k
Resistor	R18	10k
Resistor	R2	1M
Resistor	R23	10k
Resistor	R24	10k
Resistor	R25	10k
Resistor	R3	10K
Resistor	R4	1K
Resistor	R5	1K
Resistor	R6	100K
Resistor	R7	33K
Resistor	R8	10K
Resistor	R9	3K

3.4 Board layout

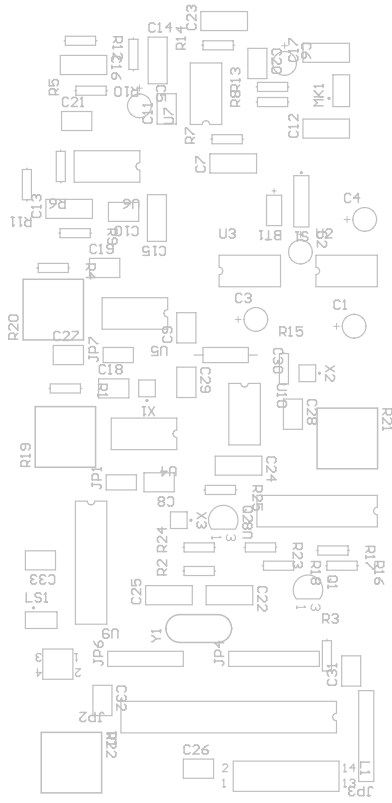
Below are the layouts for the printed circuit board in actual scale.



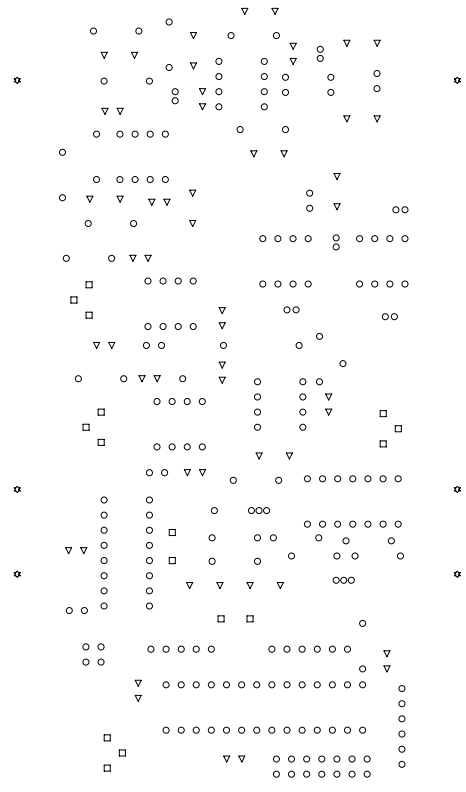
Top layer



Bottom layer



Top overlay



Drill drawing

3.5 Tuning procedure

After assembly, the device must be tuned to the proper frequency and sensitivity threshold. Begin by placing an appropriately grounded oscilloscope probe on pin 5 of U10 and tuning R21 until the frequency of the observed square wave is 40 kHz (Period of 25 μ s).

Next adjust the LCD contrast by tuning R22 while the device is on until the displayed text is comfortably viewable.

Now attach a scope probe to pin 8 of U10, and another on pin 3 of U10 for triggering. When the device is operating, the amplified receiver pulses are on pin 3. Adjust R20 so that pin 8 of U10 is consistently high (at +5V) when no pulse is received on pin 3. This sets the gain of the final amplifier stage, which should be maximized but not high enough to cause false triggering.

4 Firmware Information

4.1 Calculation overview

The firmware on the internal microcontroller emits a pulse of 400 μs on pin 2 (RA0) to allow the transmitter to output 16 cycles of a 40 kHz square wave. After a delay (830 μs) to allow the transmitter settle back to its turned off state, the microcontroller polls the trigger pin (RA1) repeated waiting for a transition, incrementing a count each time. This is done 16 times for each displayed reading.

If a maximum count (corresponds to 20 ft.) is exceeded, the target object is deemed too far away to measure, and the count is discarded. If all 16 counts were discarded, “Unmeasurable” is displayed.

Of the counts registered, the counts greater than half the maximum count are averaged to obtain the final count to avoid early triggering from noise. The final count is then converted to a distance through a linear transform of the form

$$\text{Distance} = \frac{B}{C}(\text{Count} + A) - D$$

Refer to the code listing for the precise numbers used. The numbers were determined from the calibration procedure outlined below.

4.2 Calibration procedure

To aid in the calibration procedure, pin 4 (RA2) of the microcontroller toggles on each count increment. This pin can be used to determine the delay and period of the count so that the factors for converting counts to distance can be determined.

However, a more accurate way is to simply record the distances read for various distances, and then try to fit corrections to the existing set of parameters (A , B , C , and D) using a linear solver such as Microsoft Excel’s Solver function. This process can be iterated several times to yield very precise readings.

4.3 Code listing

The bulk of the code resides in a single file `main.c`, with auxiliary functions in several other files. The LCD code is included in the MCC18 libraries and so is not provided.

```
/*  
 * This is main.c, the main loop code  
 */  
  
#include <p18f2455.h>  
#include "xlcd.h"  
#include <delays.h>
```

```

#include <portb.h>

/* Number of samples to take for each displayed reading */
#define SAMPLES 24

/* The maximum count, beyond which is deemed too long */
#define MAX_COUNT 6000 /* Corresponds roughly to 20 feet */
#define MAX_POW10 6000

/* These are the four constants used to convert from counts to lengths */
#define COUNT_OFFSET_IN 303
#define COUNT_FACTOR_NUM_IN 578
/* make the denominator a tenth of what it should be,
   since we divide by 10 when printing it */
#define COUNT_FACTOR_DEN_IN 2022
#define CORRECTION 34

/*
Connect LCD as follows:
DATA4 - RB0
DATA5 - RB1
DATA6 - RB2
DATA7 - RB3
E      - RB4
RS     - RB5
R/W   - RB6
*/

/* LCD command set */
#define LCD_CMD_CLEAR 0x01
#define LCD_CMD_CURSOR_HOME 0x02
#define LCD_CMD_CURSOR_LEFT 0x04
#define LCD_CMD_CURSOR_RIGHT 0x06
#define LCD_CMD_SCROLL_LEFT 0x18
#define LCD_CMD_SCROLL_RIGHT 0x1c
#define LCD_CMD_CURSOR_OFF 0x0c
#define LCD_CMD_LINE1 0x80
#define LCD_CMD_LINE2 0xc0

#pragma config PLLDIV = 5
#pragma config CPUDIV = OSC3_PLL4
#pragma config USBDIV = 2
#pragma config FOSC = HSPLL_HS
#pragma config PWRT = ON
#pragma config WDT = OFF
#pragma config CP0 = OFF
#pragma config CP1 = OFF
#pragma config CP2 = OFF
#pragma config CPB = OFF
#pragma config CPD = OFF

```

```

#pragma config CCP2MX = ON
#pragma config PBADEN = OFF
#pragma config LVP = OFF

#pragma code

/*
This function prints a tenth of a number (n/10) to the LCD
*/
void print_num(long int n){
    long int pow10 = MAX_POW10;
    unsigned char q,mod;

    while(pow10 > n){ pow10 /= 10; }
    while(pow10 > 1){
        q = n / pow10;
        n = n % pow10;
        pow10 /= 10;
        q += '0';
        while(BusyXLCD()); WriteDataXLCD(q);
    }
    while(BusyXLCD()); WriteDataXLCD('.') ;
    n += '0';
    while(BusyXLCD()); WriteDataXLCD(n);
}

/*
This function prints a thousandth of a number (n/1000) to the LCD
*/
void print_num_div_hundred(long int n){
    long int pow10 = MAX_POW10;
    unsigned char q,mod;

    while(pow10 > n){ pow10 /= 10; }
    while(pow10 > 100){
        q = n / pow10;
        n = n % pow10;
        pow10 /= 10;
        q += '0';
        while(BusyXLCD()); WriteDataXLCD(q);
    }
    q = n / pow10;
    n = n % pow10;
    while(BusyXLCD()); WriteDataXLCD('.') ;
    q += '0';
    while(BusyXLCD()); WriteDataXLCD(q);
    pow10 /= 10;
    n = n / pow10;
    n += '0';
    while(BusyXLCD()); WriteDataXLCD(n);
}

```

```

}

void main (void){
    char data;
    long int count[SAMPLES];
    unsigned char i, j;
    long int cur;
    long int max, sum;

    /* Initialize all port bits and directions */
    ADCON1 = 0x0f;
    ClosePORTB();
    TRISAbits.TRISA0 = 0;
    TRISAbits.TRISA1 = 1;
    TRISAbits.TRISA2 = 0;
    PORTAbits.RA0 = 1;

    /* Initialize the LCD */
    OpenXLCD(FOUR_BIT & LINES_5X7);
    while(BusyXLCD());
    WriteCmdXLCD(BLINK_ON);

    /* Display a startup message */
    while(BusyXLCD()); putsXLCD("Measuring ... ");

    /* This is the main loop which takes readings */
    while(1){
        for(i = 0; i < SAMPLES; i++){ /* Take SAMPLES per reading */
            /* This outputs the oscillator output pulse */
            LATAbits.LATA0 = 0;
            /* 16*150 instruction clock cycles at 24Mhz internal
               speed (6 MIPS) for a 16 cycle 40khz wave */
            Delay100TCYx(24);
            LATAbits.LATA0 = 1;

            /* Delay here to let the oscillator settle down */
            Delay100TCYx(50);

            /* Begin the counting loop
               Terminates when RA1 = 0 */
            cur = 1;
            while(PORTAbits.RA1 == 1 && cur < MAX_COUNT){
                cur++;
                LATAbits.LATA2 ^= 1;
            }
            if(cur == MAX_COUNT){ /* timed out; undetectable */
                count[i] = 0;
            } else { // We actually got a distance
                count[i] = cur;
            }
        }
    }
}

```

```

        /* Give some time between samples to let echoes die */
        Delay10KTCYx(22);
    }

    /* Find the maximum value captured */
    max = 0;
    for(i = 0; i < SAMPLES; i++){
        if(max < count[i]){ max = count[i]; }
    }
    max >>= 1; /* max contains the half-max value now */
    /* The above value is the threshold for a "valid" sample */

    /* Now find the "valid" samples and average them */
    sum = 0;
    j = 0;
    for(i = 0; i < SAMPLES; i++){
        if(count[i] > max){
            sum += count[i];
            j++;
        }
    }

    while(BusyXLCD()); WriteCmdXLCD(LCD_CMD_CLEAR);
    while(BusyXLCD()); WriteCmdXLCD(LCD_CMD_LINE1);

    if(j == 0){
        /* Too far; display message */
        while(BusyXLCD()); putsXLCD("Unmeasurable");
    } else {
        /* Display the (converted) data */
        cur = sum/j;
        cur += COUNT_OFFSET_IN;
        cur *= COUNT_FACTOR_NUM_IN;
        cur /= COUNT_FACTOR_DEN_IN;
        cur -= CORRECTION;
        max = cur;
        print_num(cur);
        while(BusyXLCD()); putsXLCD("in = ");

        cur /= 12;
        print_num(cur);
        while(BusyXLCD()); putsXLCD("ft");
        while(BusyXLCD()); WriteCmdXLCD(LCD_CMD_LINE2);

        cur = max;
        cur *= 127;
        cur /= 50;
        print_num(cur);
        while(BusyXLCD()); putsXLCD("cm = ");

        print_num_div_hundred(cur);
    }

```

```

        while(BusyXLCD()); putsXLCD("m");
        while(BusyXLCD()); WriteCmdXLCD(LCD_CMD_CURSOR_OFF);
    }

    /* Wait to let echoes die down */
    Delay10KTCYx(22);
}

/*
 * This is lcd_delay.c, implementing the necessary delay functions
 * for the library LCD functions.
 */

#include <p18f2455.h>
#include <delays.h>

void DelayFor18TCY(void){
    Nop(); Nop(); Nop(); Nop(); Nop();
    Nop(); Nop(); Nop(); Nop(); Nop();
    Nop(); Nop(); Nop(); Nop(); Nop();
    Nop(); Nop(); Nop(); Nop(); Nop();
}

void DelayPORXLCD(void){
    Delay1KTCYx(90);           // delay of 15ms
                               // cycles = TimeDelay * Fosc /4
                               // Cycles = 15 ms * 24MHz / 4
                               // Cycles = 90,000

    return;
}

void DelayXLCD(void){
    Delay1KTCYx(30);           // delay of 5ms, a third of above
    return;
}

```