BENG (HONS) ELECTRICAL AND ELECTRONIC ENGINEERING

Module Title: Embedded Systems

Module Number: EN0627

Coursework Title: Microcontroller Based Distance Measurement System

Module Tutor: Dr. Peter Harrington
Dr. Xuewu Dai

Academic Year: 2015 – 2016

Assignment Report

Team members:

Yeqiu Tang	w14029808	Hardware design
Tianwei Li	w14029820	Software development

Submission date: 18/01/2016

Contents

1. Introduction

1.1Background

TRISK, a local company is currently replacing their old control electronics with PIC controllers. This project requires us to design a PIC microcontroller based distance measurement system for them, which means the product will be applied in the industry environment. Also, the design and development of our product have to meet the requirement of TRISK. What's more, this distance indicator should use a Polaroid ultrasonic sensor to meter the distance from the infra-red heater to the vehicle.

1.2Basic theory

This application is based upon the reflection of ultrasound waves. All kinds of sound waves are defined as longitudinal pressure waves in the medium in which they are travelling. Subjects whose dimensions are larger than the wavelength of the impinging sound waves reflect them; the reflected waves are called the echo. If the speed of sound in the medium is known and the time taken for the sound waves to travel the distance from the source to the subject and back to the source is measured, the distance from the source to the subject can be computed accurately. This is the measurement principle of this application. Here the medium for the sound waves is air, and the speed of ultrasound in air is around 340m/s. [1]

The basic sonar illustration shows below.

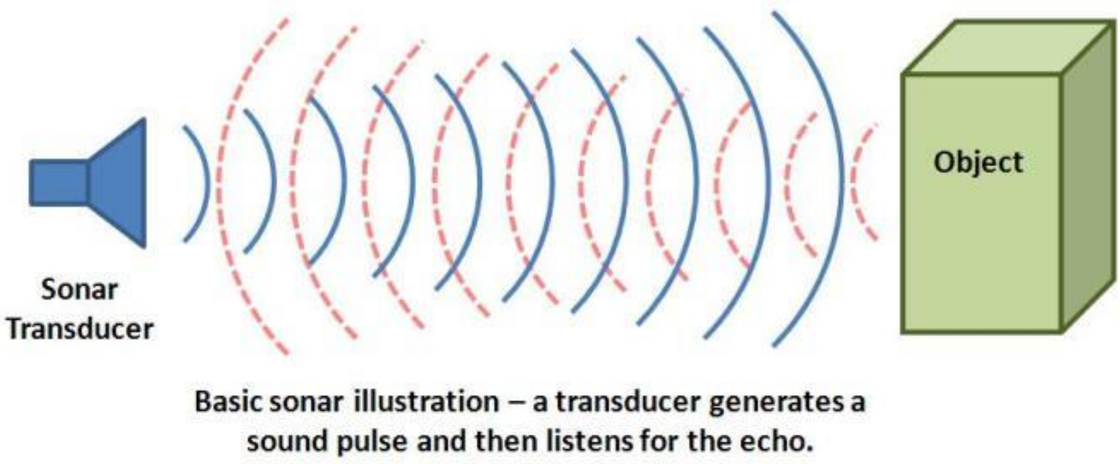


Figure 1 the basic sonar illustration [2]

The distance between the transducer and obstacle can be calculated by equation below.

$$Distance = {V_{sound\ in\ air} \times T}/{2}$$

2. Design requirement

Microcontroller Based Distance Measurement System	
Power supply	9V dc (battery)
Power consumption	0.5W
LED Green 'band'	735 – 810mm from panel
Dynamic range	500 – 1300mm
Radius of curvature of panel detected	100mm
Accuracy	97.5%
Cost target	£ 12
Enclosure size	100x50x25mm
Additional features	LCD1602 Display Module Downloader header External crystal oscillator

Table 1 Design specification

3. Development objectives

3.1Major objectives

a. Design an ultrasonic sensor driver circuit based on PIC microcontrollers.

b. Make this circuit can be connected to ultrasonic sensor via RS232 connector.

c. Make this circuit can be connected to computer directly via PICkit.

d. Design a circuit and relative program to indicate distance using with LEDs.

e. Design a circuit and relative program to adjust the dynamic range.

f. Design a DC power supply circuit powered by a 9V PP3 battery.

g. Power consumption of this driver circuit is less than 0.5W.

h. The total cost is within 12 GBP.

i. Enclosure size: 100 x 50 x 25mm

3.2Minor objectives

a. Design a user-friendly visual interface using with LCD1602 module.
b. Design an effective calibration algorithm for higher accuracy.
c. Use high frequency external crystal oscillator to improve the accuracy.

4. Hardware design

4.1Schematic design

4.1.1 Microcontroller selection

According to the requirements, we decided to use LCD1602 module as an additional feature to display the green ‘band’ range and current distance from object to sensor. Besides, the other additional feature is the onboard downloader port.

The number of I/O ports we need shows in the table below.

Component	Number of I/O ports	Description
LCD1602	11	RS, RW, E, D0-D7
Crystal oscillator	2	
Tact button	3	Reset, Plus, Minus
Sensor control	2	INIT, ECHO
Downloader port	3	Reset, ICDCLK and ICDDAT
LED	3	Green, Yellow, Red

Table 2 The I/O ports needed

The table above shows that we need at least 24 I/O ports if we need to implement our application. However, we noticed the number of pin of PIC16f1847 is only 18 (include VDD and VSS). For further development and additional features we need a chip with more pins.

Therefore, we selected a classic 40-pin microcontroller, PIC16F877A [3]. This chip has been used by thousands of engineers in recent years, which proves its capability and stability. Although the specification of PIC16F877A is not as good as the new chip PIC16F1847, its performance is totally enough for our application.

The specification of PIC16F877A is shown below.

Figure 2 PIC16F877A

Parameter Name	Value
Program Memory Type	Flash
Program Memory (KB)	14
CPU Speed (MIPS)	5
RAM Bytes	368
Data EEPROM (bytes)	256
Digital Communication Peripherals	1-UART, 1-A/E/USART, 1-SPI, 1-I2C1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	2 CCP
Timers	2 x 8-bit, 1 x 16-bit
ADC	8 ch, 10-bit
Comparators	2
Temperature Range (C)	-40 to 125
Operating Voltage Range (V)	2 to 5.5
Pin Count	40

Table 3 Parameters of PIC16F877A

4.1.2 Main circuit design

To implement the fundamental function of an ultrasonic distance meter, we need 5 basic circuits below. Proteus 8.1 pro can be used to design both the schematic and PCB.

A. The MCU external clock and reset circuit

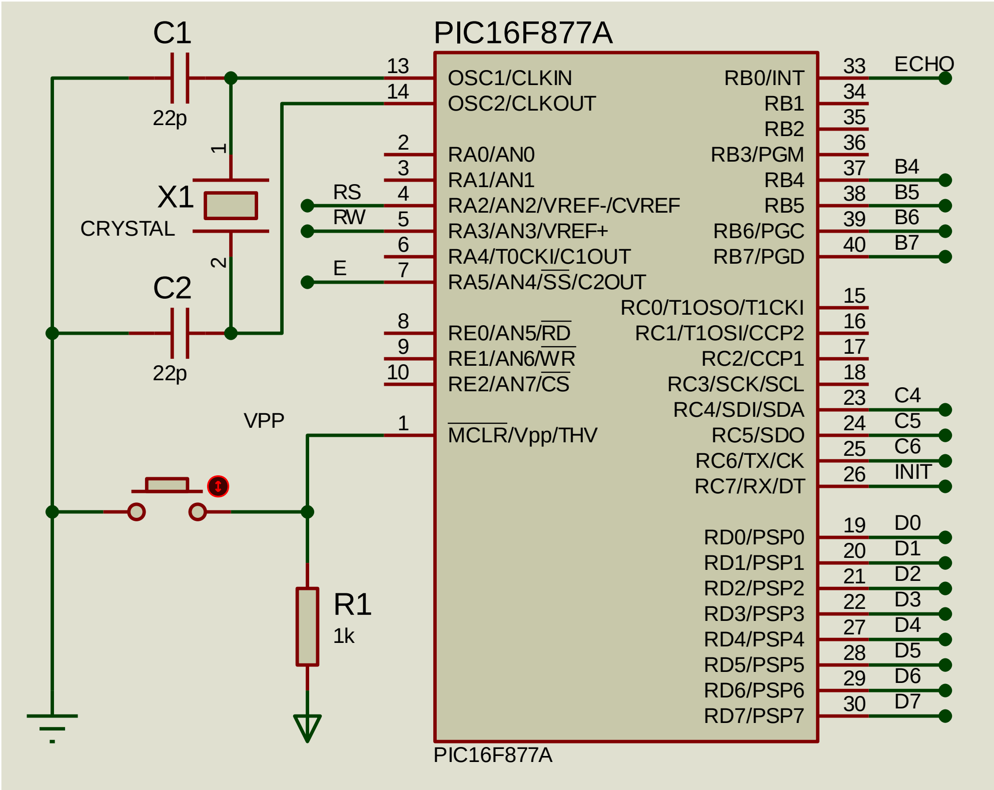


Figure 3 The MCU external clock and reset circuit

For the accuracy required, we selected an external 8MHz crystal oscillator connected with two 22pF capacitors (C1 and C2) to generate clock signal for our system, which will increase the accuracy of the timer because the crystal is more accurate comparing to internal RC oscillator. According to the datasheet, Pin 1 is the hardware reset port of the microcontroller, a reset button is necessary in case users need to restart the system in some conditions. The description of those ports is shown below. [3]

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI	13	14	30	32		ST/CMOS[4]	Oscillator crystal or external clock input.
OSC1					I		Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS.
CLKI					I		External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).
OSC2/CLKO	14	15	31	33		—	Oscillator crystal or clock output.
OSC2					O		Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO					O		In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP	1	2	18	18		ST	Master Clear (input) or programming voltage (output).
$\overline{MCLR}$					I		Master Clear (Reset) input. This pin is an active low Reset to the device.
VPP					P		Programming voltage input.

Table 4

B. The power supply circuit

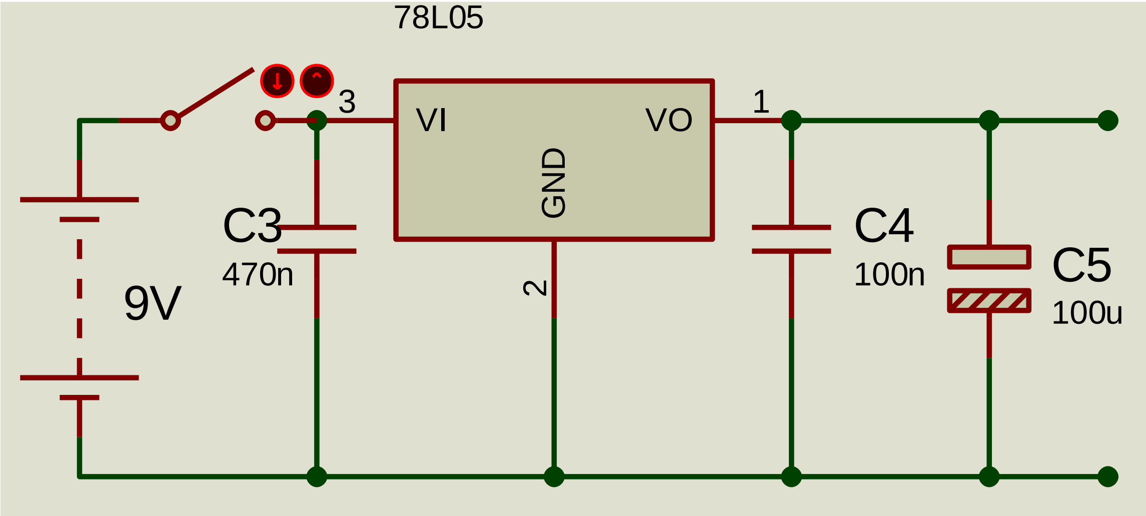


Figure 4 the power supply circuit

As we use 9V pp3 battery as our power source, 78L05 is a comparatively better choice compared with the voltage regulators. As a frequently used voltage regulator, 78L05 is cheap and easy to use. The main specification of 78L05 [4] is show in Table 5.

78L05		
Parameter	Value	Unit
Input Voltage	7-18	V
Output Voltage	5	V
Output Current	100(max)	mA

Table 5 Specification of 78L05

From the table above we can see the output of voltage and current is 5V and 100mA, what is enough for our system and the maximum power consumption will be less than 0.5W (5V x 100mA = 0.5W)

The capacitor C3 and C4 is used for clutter suppression. The electrolytic capacitor C5 is parallel at the output port to prevent the damage to regulator caused by soaring current at the time when the power is switched on.

C. LED circuit

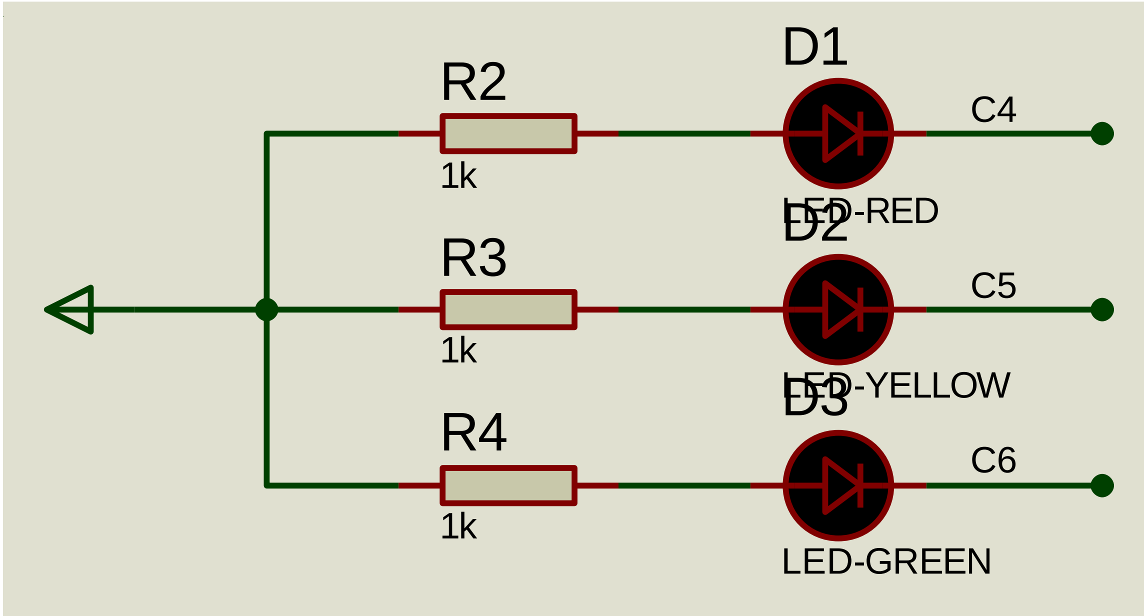


Figure 5 LED circuit

Three colors LEDs (Red, Yellow and Green) are connected in a common anode display, controlled by PORTC4, PORTC5, and PORTC6, according to the configuration shown below [3].

RC4/SDI/SDA	bit 4	ST	RC4 can also be the SPI data in (SPI mode) or data I/O (I^2C mode).
RC5/SDO	bit 5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit 6	ST	Input/output port pin or USART asynchronous transmit or synchronous clock.

Table 6

D. Tact switches circuit

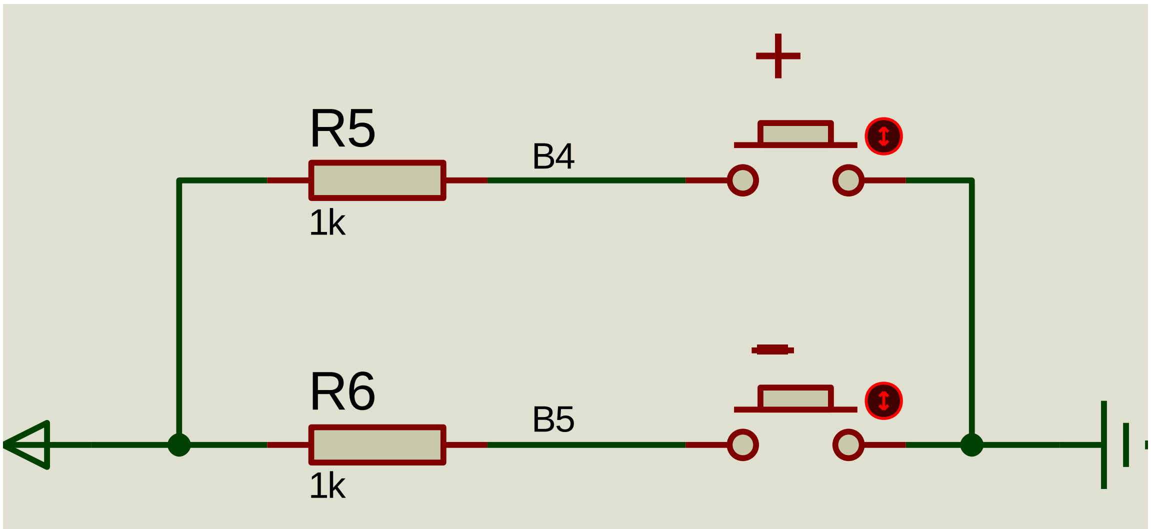


Figure 6 Tact switches circuit

According to the requirements we need two buttons (plus and minus) to change the dynamic range. For quick and accurate response, plus and minus buttons are connected to PORTB4 and PORTB5 due to PORTB's interrupt-on-change function. The description of them is shown below [3].

RB4	bit 4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5	bit 5	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.

Table 7

E. RS232 Connector port

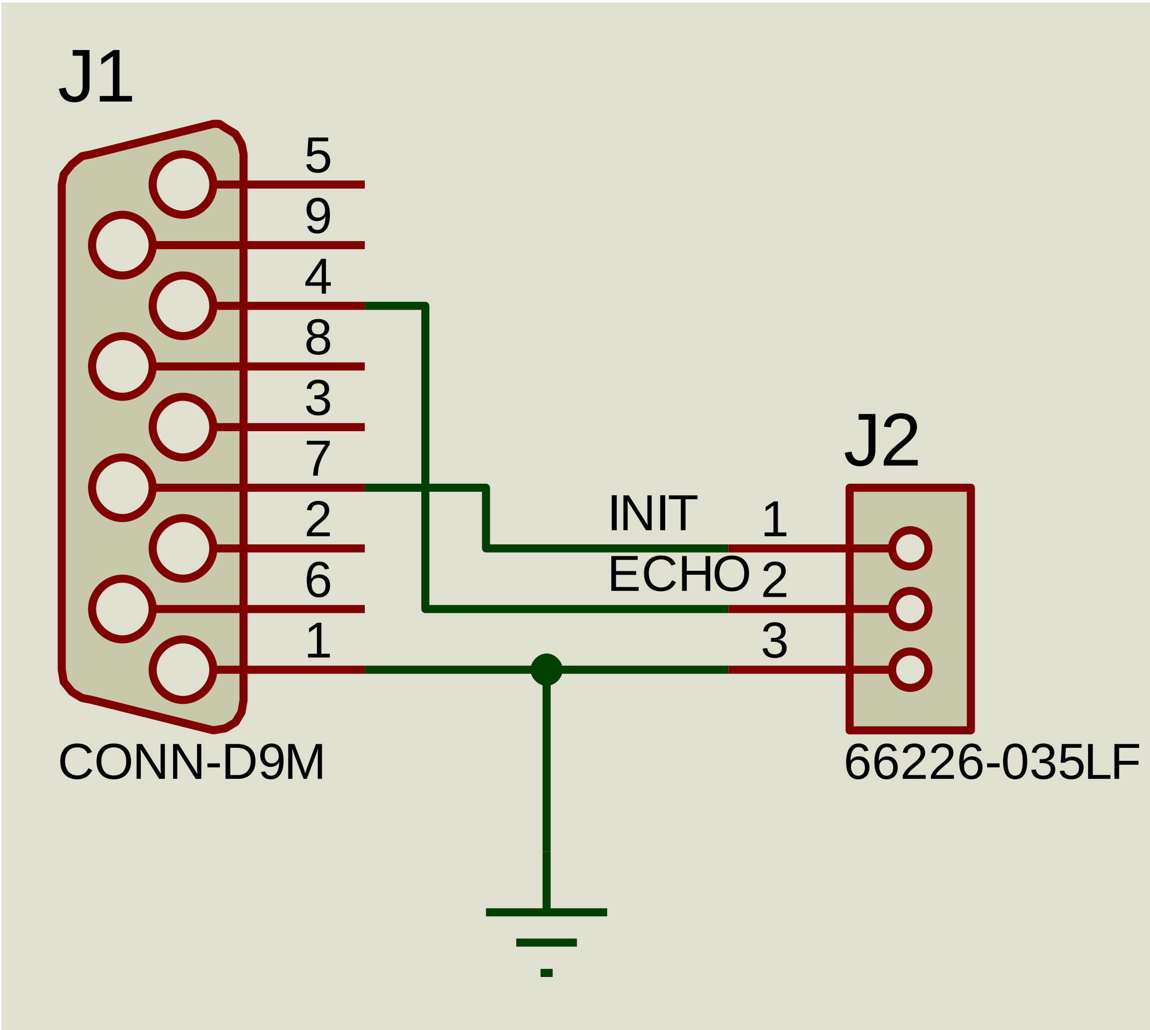


Figure 7 RS232 Connector port

The device we designed is connected to ultrasonic sensor via RS232 connector. In this way, INIT, ECHO are connect to PORTC7 and PORTB0 of the microcontroller. The specification is shown below [3].

RB0/INT	bit 0	TTL/ST(1)	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RC7/RX/DT	bit 7	ST	Input/output port pin or USART asynchronous receive or synchronous data.

Table 8

4.1.3 Additional features

A. LCD1602 socket design

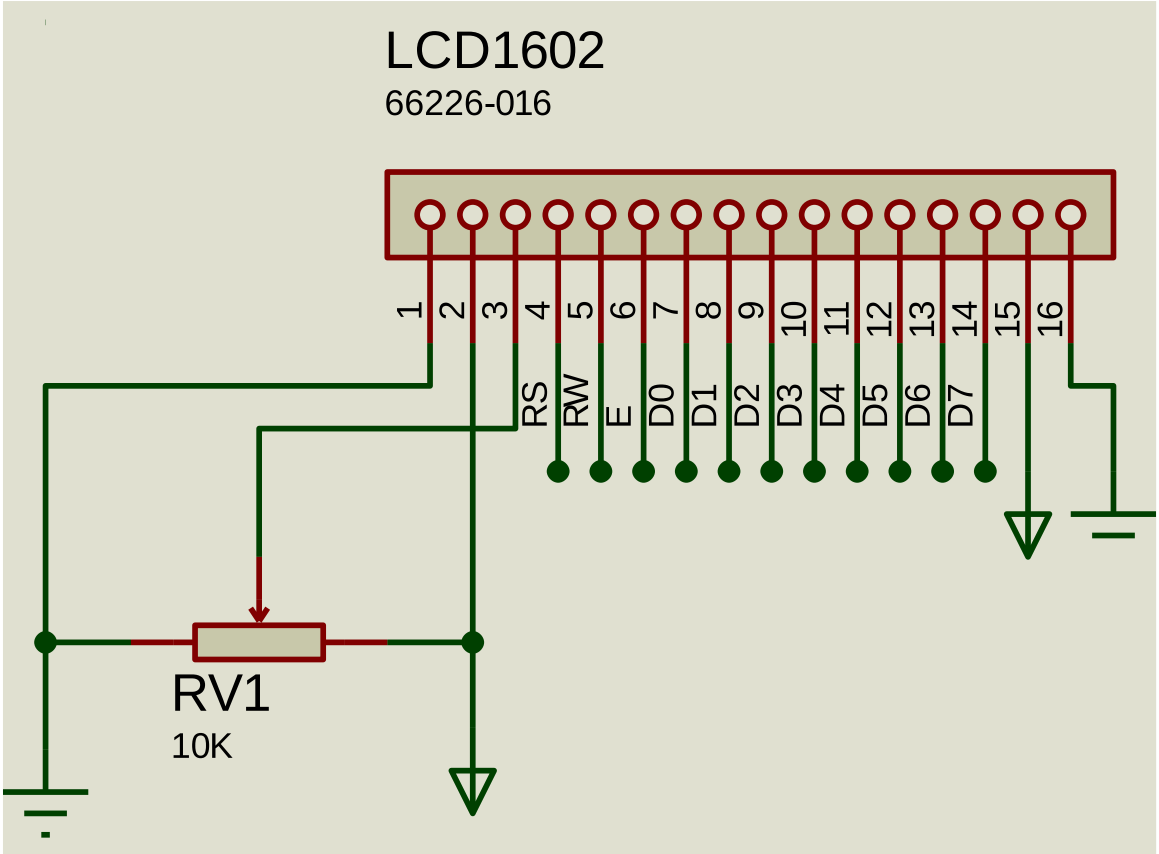


Figure 8 LCD1602 socket design

RV1 is the potentiometer used to adjust the contrast of LCD [5]. Along with the control port RS, RW and E connected to PORTA2, PORTA3 and PORTA5, the data port D0~D7 are connected to PORTD0~PORTD7. The configuration is shown below [3].

RA2/AN2/VREF-/CVREF	bit 2	TTL	Input/output or analog input or VREF- or CVREF.
RA3/AN3/VREF+	bit 3	TTL	Input/output or analog input or VREF+.

RA5/AN4/$\overline{SS}$/C2OUT	bit 5	TTL	Input/output or analog input or slave select input for synchronous serial port or comparator output.

RD0/PSP0	bit 0	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 0.
RD1/PSP1	bit 1	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 1.
RD2/PSP2	bit2	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 2.
RD3/PSP3	bit 3	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 3.
RD4/PSP4	bit 4	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 4.
RD5/PSP5	bit 5	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 5.
RD6/PSP6	bit 6	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 6.
RD7/PSP7	bit 7	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 7.

Table 9

B. Downloader header

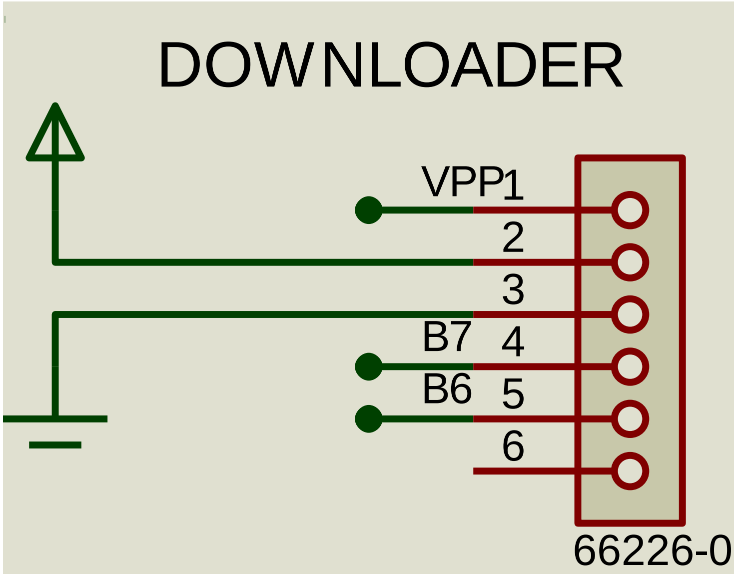


Figure 9 Downloader header

PICkit3 In-Circuit Downloader/Programmer is the official debug tool recommended by Microchip working along well with MAPLAB X IDE. Pin 1 to Pin 5 are connected to VPP, VDD, VSS, PORTB7 and PORTB6, shown in Figure 9 and Figure 10.

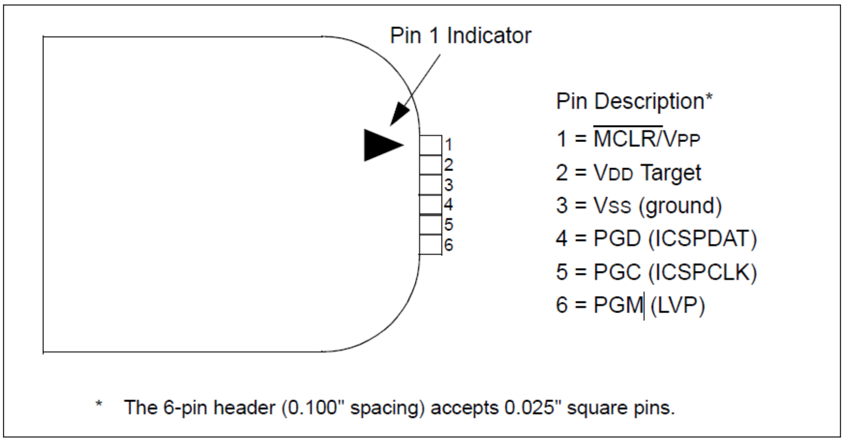


Figure 10

PICkit™ 3 PROGRAMMER CONNECTOR PINOUT

For the reason that debugger clock and data port cannot be multiplexed during debugging process, PORTB7 and PORTB6 are reserved for independent downloader ports. The configuration of PORTB6 and PORTB7 is shown Table 10 [3].

RB6/PGC	bit 6	TTL/ST(2)	Input/output pin (with interrupt-on-change) or in-circuit debugger pin. Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit 7	TTL/ST(2)	Input/output pin (with interrupt-on-change) or in-circuit debugger pin. Internal software programmable weak pull-up. Serial programming data.

Table 10

With programmer port we do not need to remove the chip from the board when we want to program it, which is convenient and can protect the chip from damage.

4.2Prototype design

For testing whether the schematic design and program can work, it is necessary to make a prototype first.

Figure 11 shows our prototype based on the universal circuit board.

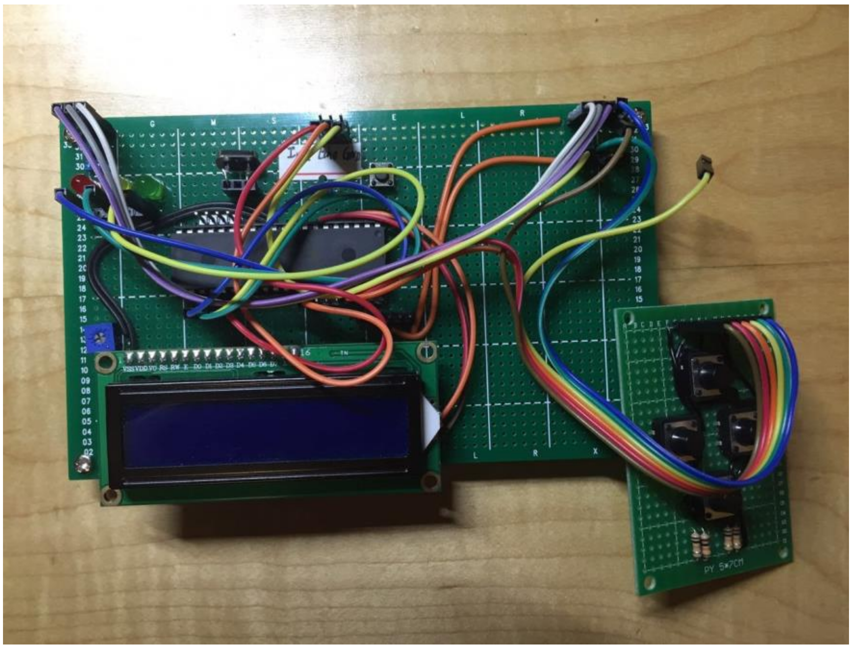


Figure 11 Prototype

The pinout of the PORTA, PORTB and PORTC makes us easier to adjust the port design by using jumper wires when developing. Furthermore, the power supply circuit is not included in the prototype because we don't need to waste the power of battery in the period of development. And actually we can use PICkit3 to supply power for the prototype, shown in Figure 12.

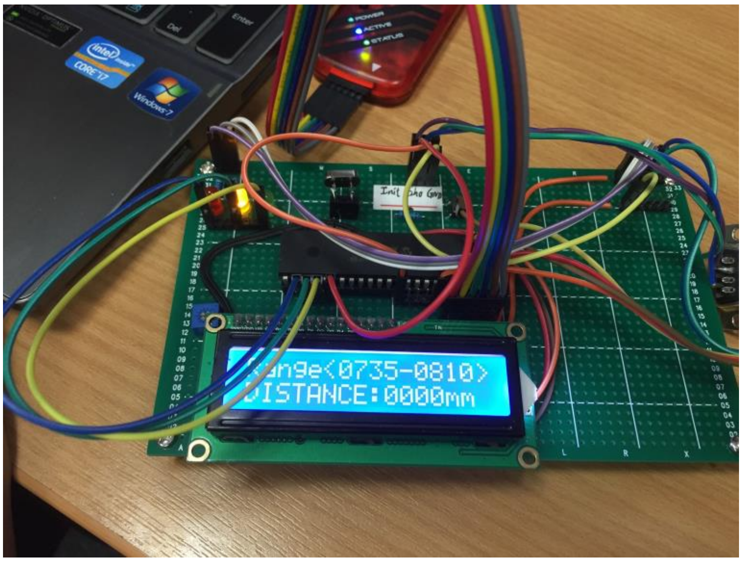


Figure 12

4.3PCB design

After developing the program and testing the schematic design in the prototype, the PCB design can get started. The separated schematic designs are shown in 4.1.

Figure 13 is the overall schematic.

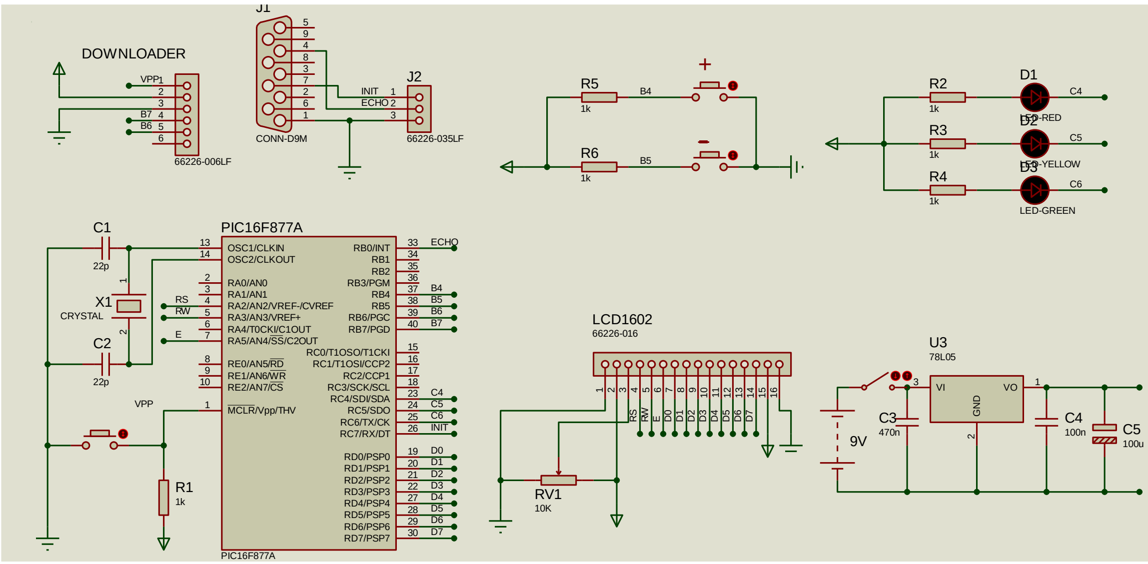


Figure 13 The overall schematic

4.3.1 1st generation (USM-1)

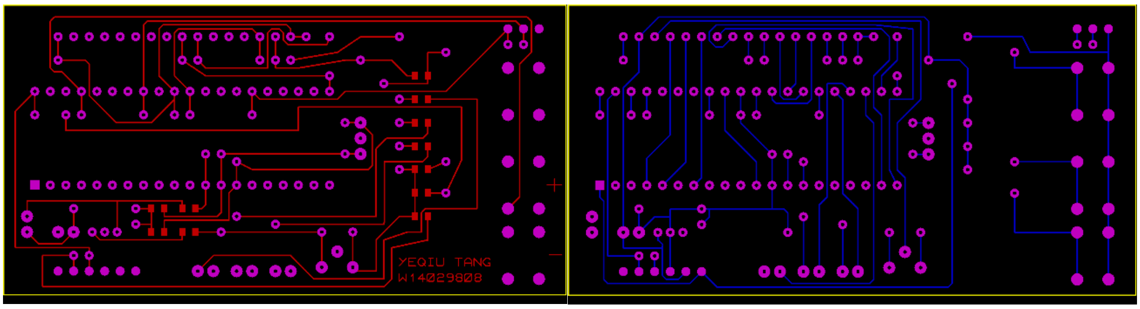


Figure 14 Top copper and bottom copper layers

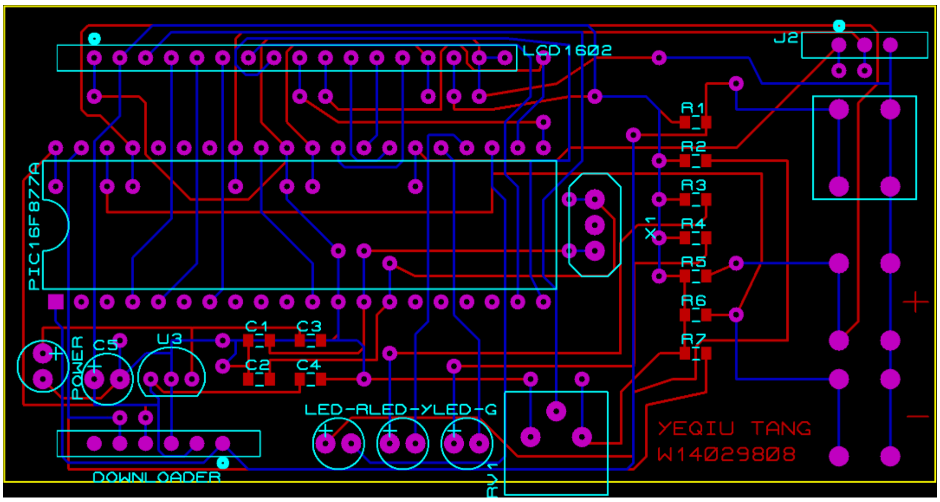


Figure 15 Overall layers

Considered the size of our product is limited by 100x50mm, the capacitors and resistors on PCB are all 0805 SMD, which can minimize the size of PCB. Therefore the size of USM-1 is 90x45mm, meeting the requirement. What's more, thanks to the auto-router function of Proteus 8.1 pro, after setting the layout of each component, the wires can be routed automatically by industry rules. The finished PCB is shown in Figure 16.

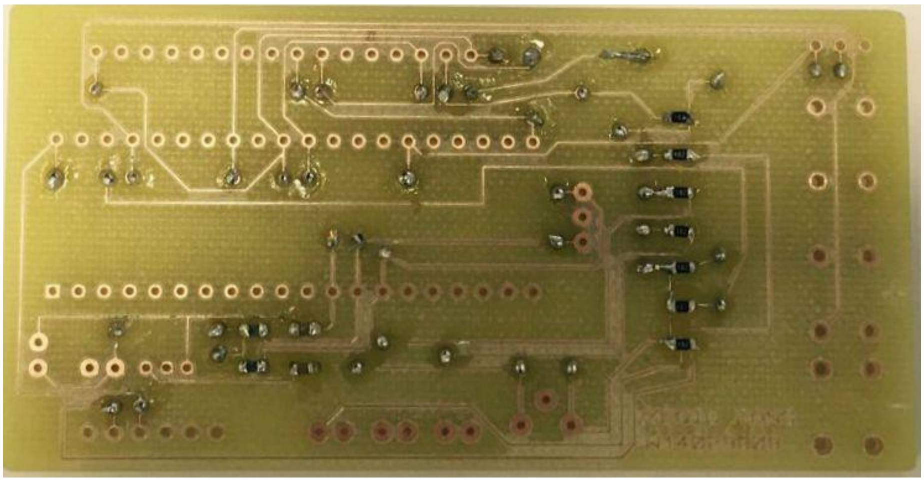


Figure 16

This PCB is made in the PCB and Drill Laboratory in EBE202. For some technical limitations of the PCB machine, there is not silk layer on our PCB and all of via holes have to be soldered manually. Fortunately, the USM-1 works properly after 5 hours' soldering work, shown in Figure 17.

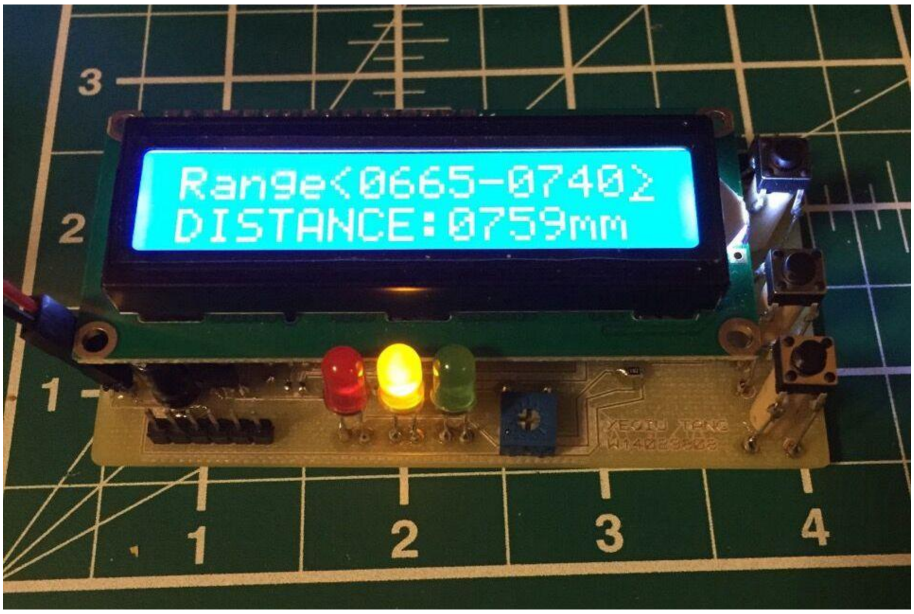


Figure 17

The size of 1st generation with battery is 110x45x21mm, where the length of it is slightly bigger.

4.3.2 2nd generation (USM-2)

After the success of 1st generation, the design of 2nd generation is raised into the schedule. The figure below shows the PCB of 2nd generation.

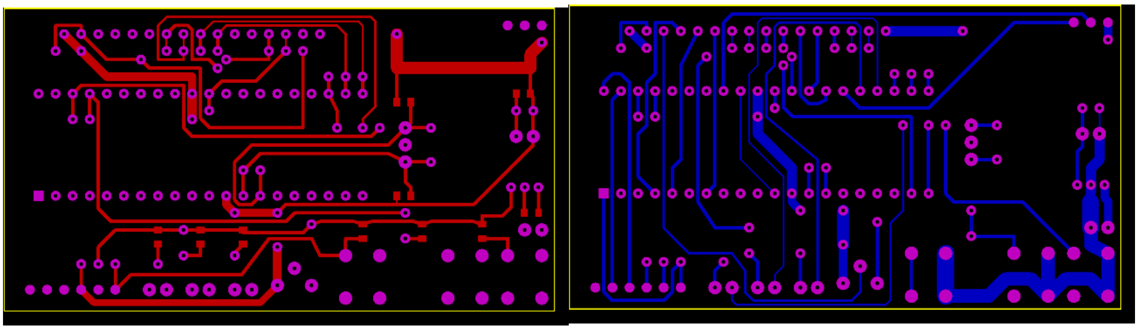


Figure 18 Top copper and bottom copper layers

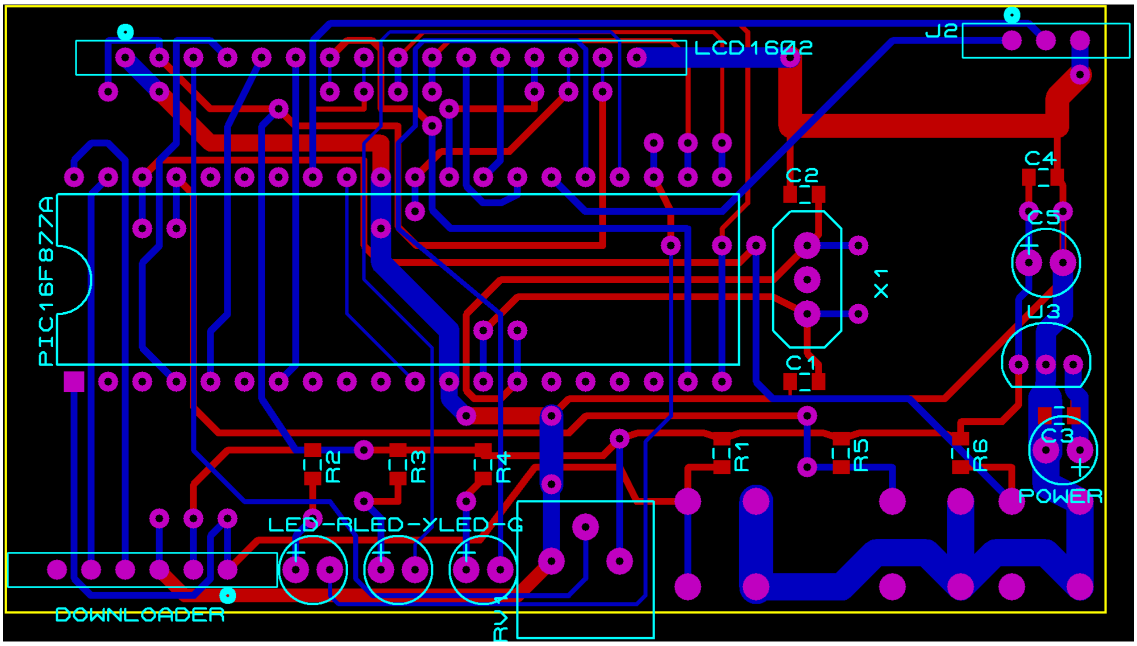


Figure 19 Overall layers

Based on 1st generation, the layout of 2nd generation is changed to make the PCB more compact. As a result, the size of this PCB is 80x45mm, smaller than 1st generation. In this way, the size of this PCB with battery (exclude box) is expected to be 94x45x21mm. Moreover, to improve system stability, all power wires are configured to more than 30 mil to avoid the circumfluence.

A note from YEQIU TANG, the hardware designer:

"Unfortunately, the producing of 2nd generation PCB is delayed because the PCB machine is broken. I can hardly solder this PCB and test it in time. In view of the coming deadline, TIANWEI LI and I decide to use 1st generation and make a box for it. Although the size of 1st generation is slightly bigger than what required, its performance and stability is beyond our expectation."

4.4 Case design

After testing the 1st generation, for better user experience, it is necessary to make a case for it. This case is made by soft wood board so its touch feeling is quite good. The whole system is shown below.

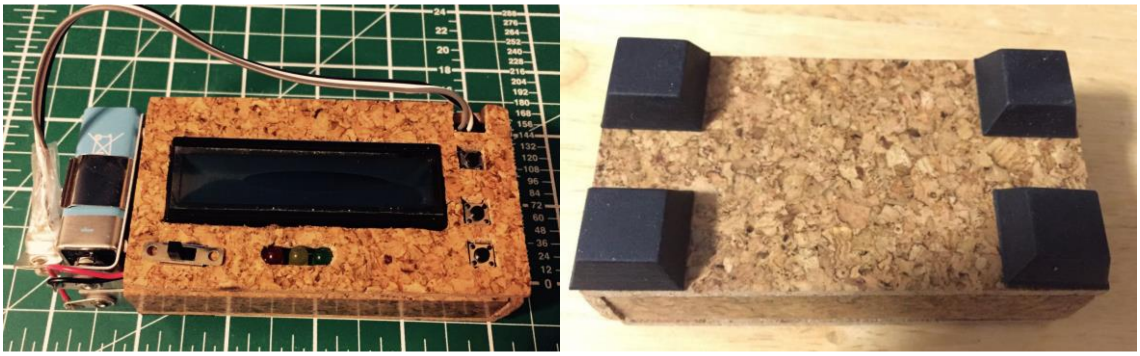


Figure 20The top and bottom view of the case

Moreover, with four skid blocks on the bottom we can place it on any smooth surfaces such as the top of ultrasonic sensor, shown in Figure 21.

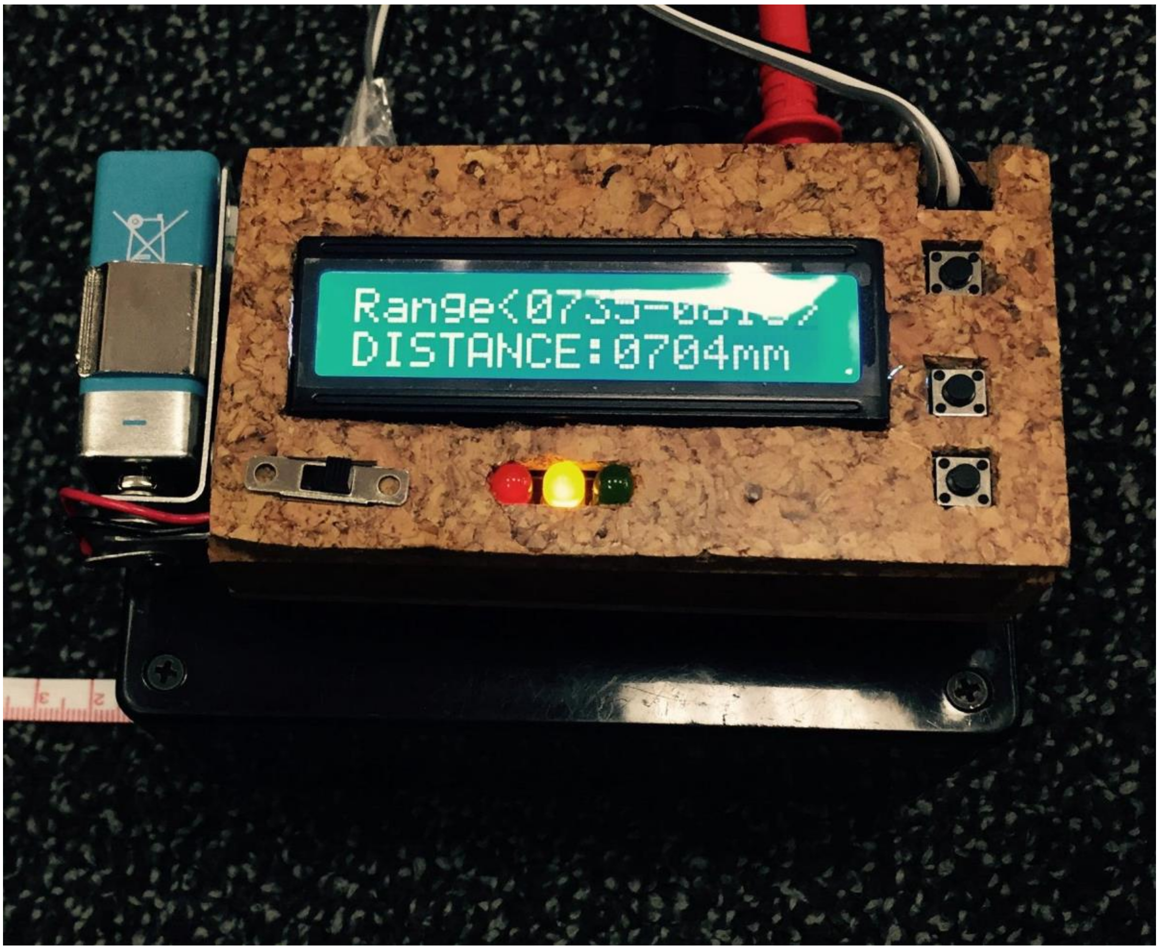


Figure 21

5 Software development

5.1 Methodology

5.1.1 Development Tools

At the beginning of development, the team has one PIC Kit II and two PIC Kit III in preparation. These tools are used for downloading the program into the chip and real-time simulation. Unfortunately, none of these is qualified as a real-time debugger.

In premise of this situation, the team decided to develop the LCD1602 drive program in advance. All of the following debug processes are based on the content of display.

It is true that the MAPLAB has a function of software emulation. However, this function cannot response to any change of the input. Needless to say, it is not capable of investigating the problem of time sequence.

5.1.2 Re-locatable Programming

Once the program is finished, it is not over. For reuse, further development or even just debugging, it is necessary to divide the program into different sections based on the functions. Talking about re-locatable programming, it is not only about division, but also about mitigating the coupling relationship between blocks, such as decreasing the amount of global variables.

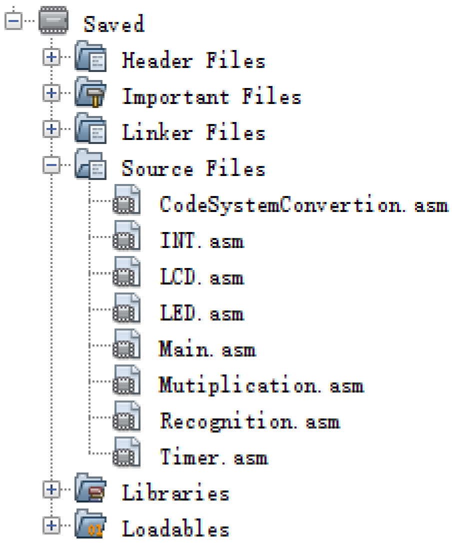


Figure 22

The figure above shows all the sources files of the project whose name indicates its own function.

```
INIT
    CALL Variable_INIT
    CALL LED_INIT
    CALL LCD_INIT
    CALL Timer_INIT
    CALL INT_Init
MAIN
        clrf TMR1H
        clrf TMR1L
        bcf STATUS, RP0
        bcf STATUS, RP1
        bsf T1CON, 0
        bsf PORTC, 7
Wait    btfss PORTB, 0
        goto Wait
        bcf T1CON, 0
        call loadAB
        call multiply
        call LoadBin
        call BtoBCD
        call Result_Adjust
        call BCD_Display
        call Range_Display
        call Compare
        bcf PORTC, 7
        CLRWDT
        SLEEP
        NOP
        goto MAIN
```

Figure 23

The figure above indicates the whole process of the programethodology.

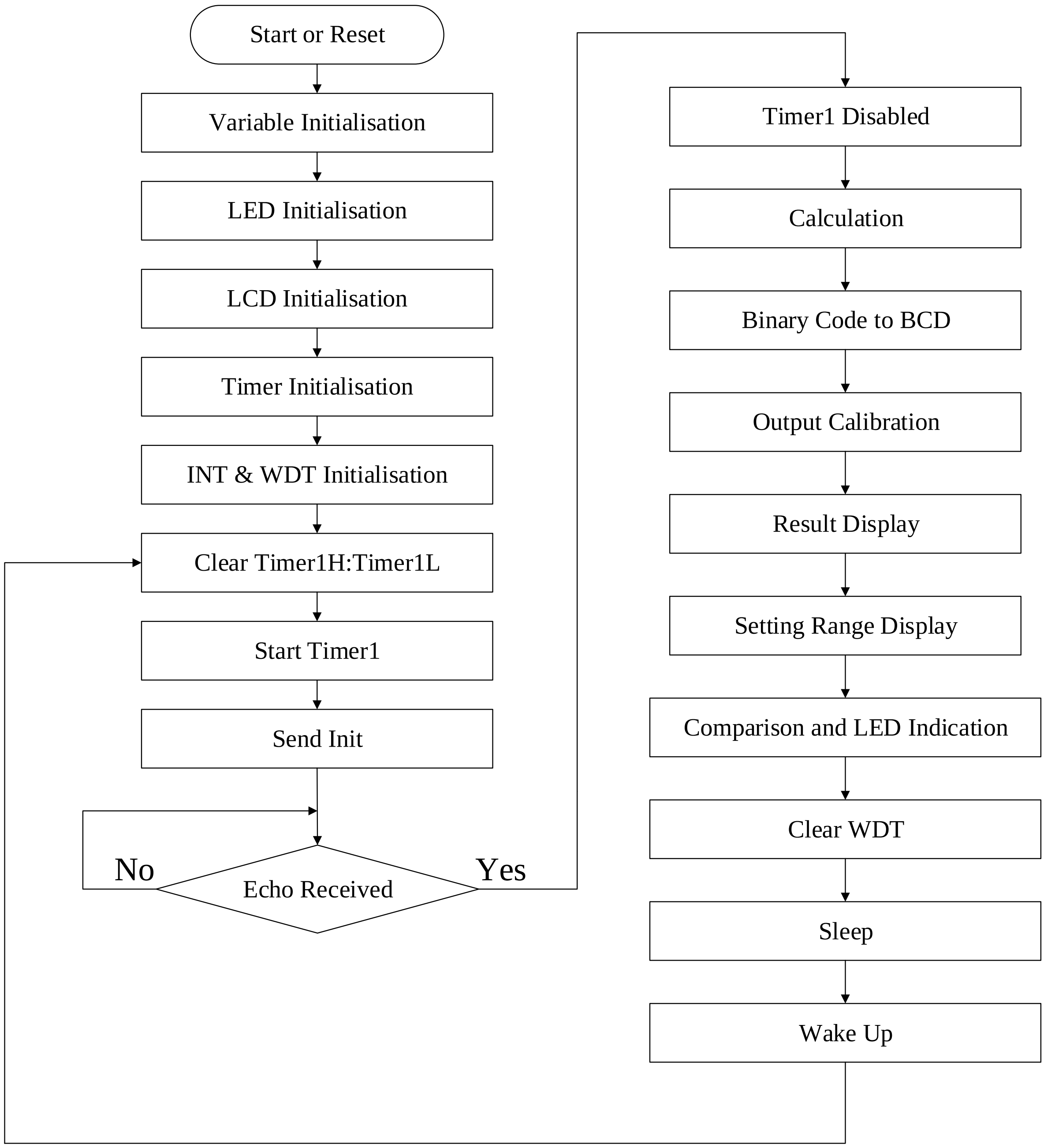
5.1.3 Configuration Bits

In prior to development, the configuration bits need to be set properly to meet the application requirement.

Field	Option	Category	Setting
FOSC	HS	Oscillator Selection bits	HS oscillator
WDTE	ON	Watchdog Timer Enable bit	WDT enabled
PWRTE	OFF	Power-up Timer Enable bit	PWRT disabled
BOREN	ON	Brown-out Reset Enable bit	BOR enabled
LVP	ON	Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit	RB3/PGM pin has PGM function; low-voltage programming enabled
CPD	OFF	Data EEPROM Memory Code Protection bit	Data EEPROM code protection off
WRT	OFF	Flash Program Memory Write Enable bits	Write protection off; all program memory may be written to by EECON control
CP	OFF	Flash Program Memory Code Protection bit	Code protection off

Figure 24 Configuration Bits

5.2 Summary



Flowchart 1

The Flowchart 1 shows how the program works in common condition.

When the chip is powered up or reset, the program initializes all the variables and peripherals, such as LED and LCD1602. After that, the registers of Timer, Interrupt and watchdog timer will be adjusted into demanding status.

When the process of initialization is done, Timer1H and Timer1L will be cleared. Then Timer1 starts to count, and the init signal is sent. At this moment, the chip will do nothing but wait for the echo signal. The reason not using interrupt in this case is because interrupt generally has a latency of four cycles. In order to achieve the highest accuracy, interrupt is not adopted. Therefore the system error will be reduced to only one cycle because the chip cannot send the init signal and start the timer at one single instant.

When the echo is received, timer1 will be disabled and the time interval will be captured to do the calculation with the result in binary-coded form. Afterwards, the result will be converted within the form of binary-coded decimal (BCD), followed by a routine of calibration which is developed based on the experiments.

Finally, the number will be displayed on the LCD. Meanwhile, the result will be compared with the preset upper and lower limits to lighten the corresponding LED and dim the left LEDs

After the process is finished, the watchdog timer will be cleared and system will turn into a sleep mode. When the system is awaken by the time out, the whole process starts over again.

5.3LCD 1602 Display

LCD 1602 is a preliminary equipment in embedded system, which can display two rows of reading and writing. The reading operation is made up of reading status (status register) and reading data (RAM), which will not be implemented in this case. The writing operation contains writing instruction and data.

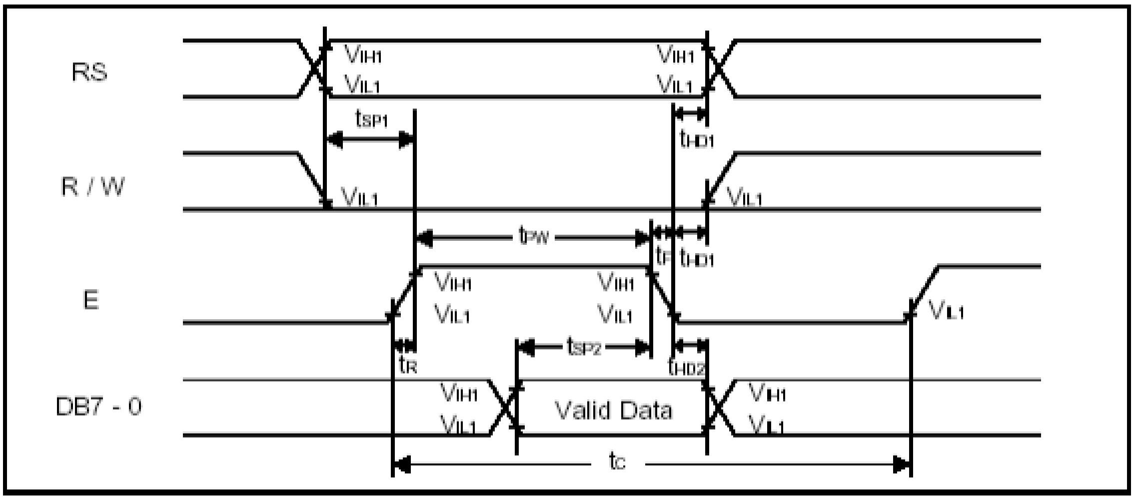


Figure 25

As can be seen in the diagram above, the operation of writing happens only when E is high. It is necessary to keep the time interval of E being high level long enough, otherwise the writing operation will fail. In this case, the time interval is set as 20ms.

The second thing needs to be paid attention to is that before writing data, the data pointer address has to be oriented in advance. Afterwards, it is time to send characters. The mode is set that the pointer will increase by one each time a character is sent.

In the routine of LCD1602, Pin 2, 3 and 5 in Port A are used as status console, meanwhile port D is working for data transmission.

The timing requirement is shown as below.

Operation	Timing Requirement	Output
Write Instruction	RS=L, RW=L, D0~D7= Instruction Code, E= High Impulse	None
Write Data	RS=H, RW=L, D0~D7= Instruction Code, E= High Impulse	None

Table 11 Timing requirement

The operations in common use has been packed into several functions:

1. Function SEND_DATA sends the characters..
2. Function ENABLE is used to send the instructions.
3. Function CLEAR clears all the characters in display.
4. Function Locate1 and Locate2 moves the cursor at the first bit of the first and second row respectively.

5.4 Distance Calculation

The calculation of distance basically depends on the how long the ultrasonic travels in the air. The time interval multiplied by the velocity of the sound in air simply gives the distance. Because the ultrasonic actually goes forward and bounces back, the result should be cut into half.

$$Distance = \frac{V_{sound\ in\ air} \times T}{2} = \frac{V_{sound\ in\ air}}{2} \times T$$

Since the accuracy of distance calculation depends on the time interval, the way to capture the instant of echo is a major problem. Generally speaking, interrupt is a quite response of embedded system. However, within an accuracy-oriented assignment, it is necessary to point out that even an interrupt has a latency of four cycles. Therefore, instead of using an interrupt, the routine turns on the timer1, sends the Init signal and just keeps checking whether the echo is received. In this case, the system error will be cut down to only one cycle which cannot be eliminated since the PIC can't turn on the timer and send the signal at a very instant. [6]

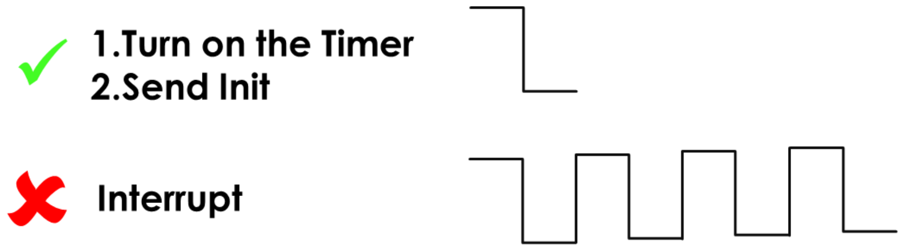


Figure 26

The Figure 27 below shows the waveform of signal Init and Echo generated from the oscilloscope. The signal are driven by a test routine which simply sends an Init to the ultrasonic sensor and the sensor give an Echo with a few delay in response.

The waveform in yellow color indicates the Init signal. Because the Init signal's power supply is a battery, it contains some ripples in both high and low levels. In contract, the waveform performs to be steadier and purer since it is supplied by a constant voltage source.

According to the waveform, the relationship between the two signals are:

1. When the Init goes high, after a delay of Tms (which is used to measure the distance), the Echo starts to go high.
2. When the Init signal is set to be low voltage, the Echo turns low immediately.

Figure 27

In order to improve the accuracy of the measurement, the double-precision multiplication method is adopted. The basic theory is to use addition and loop instead of direct multiplication.

16-bit × 16-bit multiplication constructs a 32-bit result. In assembly language, the result is divided into 4 8-bit variables.

Program

```
;*****************************************************************************
; Load time value to ValueA and Scaling factor values to ValueB
loadAB
        movf    TMR1H, w
        movwf   ValueAH
        movf    TMR1L, w
        movwf   ValueAL

        movf    ScaleH, w
        movwf   ValueBH
        movf    ScaleL, w
        movwf   ValueBL
        return

```

```
;**************************************************************************
;32-bit outcome with higher significant value in ValueBH:ValueBL and lower
;significant value in ValueCH:ValueCL
multiply
            call    setup                   ;rotate ValueD
mloop       rrf     ValueDH, f
            rrf     ValueDL, f
            btfsc   STATUS, C               ;if carry bit is set, offset will be done
            call    add
            rrf     ValueBH, f
            rrf     ValueBL, f
            rrf     ValueCH, f
            rrf     ValueCL, f
            decfsz  temp_shift, f           ;loop untile all bits are finished
            goto    mloop
            retlw   0

;-----------------------------------------
setup       movlw   .16                     ;check for 16 times
            movwf   temp_shift
            movf    ValueBH, w
            movwf   ValueDH
            movf    ValueBL, w
            movwf   ValueDL
            clrf    ValueBH
            clrf    ValueBL
            retlw   0
;-----------------------------------
;addtion value ValueB + ValueA ->ValueB
add         bcf     Flags, 0
            movf    ValueAL, w              ;add LSB
            addwf   ValueBL, f              ;carry bit offset
            btfss   STATUS, C
            goto    add1
            incf    ValueBH
            btfsc   STATUS, Z
            bsf     Flags, 0
add1
            movf    ValueAH, w              ;ValueB= ValueA + ValueB
            addwf   ValueBH, f              ;add MSB
            btfsc   Flags, 0
            bsf     STATUS, C
            retlw   0
;*************************************************
            return
            end
```

5.5 Code System Conversion

5.5.1 Binary-to-BCD Conversion

Apparently, a result in a format of binary code is not the end. In order to get the final display on the LCD, the binary value need to be converted into binary-coded decimal for further conversion.

```
    CODE
BtoBCD
    bcf STATUS, C
    movlw .32           ;preset a 32 times loop
    movwf temp_shift2
    clrf BCD4           ;Clear all the outputs
    clrf BCD3
    clrf BCD2
    clrf BCD1
    clrf BCD0
Loop32
    rlf Bin0, f         ;rotate to the left
    rlf Bin1, f
    rlf Bin2, f
    rlf Bin3, f
    rlf BCD0, f
    rlf BCD1, f
    rlf BCD2, f
    rlf BCD3, f
    rlf BCD4, f
    decfsz temp_shift2, f
    goto ADJDEC
    retlw 0
```

```
ADJDEC
    movlw BCD0
    movwf FSR
    call ADJBCD
    movlw BCD1
    movwf FSR
    call ADJBCD
    movlw BCD2
    movwf FSR
    call ADJBCD
    movlw BCD3
    movwf FSR
    call ADJBCD
    movlw BCD4
    movwf FSR
    call ADJBCD
    goto Loop32


ADJBCD
    movlw 0x03
    addwf INDF, w
    movwf temp_BCD
    btfsc temp_BCD, 3
    movwf INDF
    movlw 0x30
    addwf INDF, w
    movwf temp_BCD
    btfsc temp_BCD,7
    movwf INDF
    retlw 0
```

5.5.2 BCD-to-ASCII Conversion

The display of letters are always preset, such as the user interface in this case is ‘Range from :< >’, ‘Distance: mm’. However, the result of measurement is not predictable.

Natural Number	Position in ASCII Table	
0	48	0x30
1	49	0x31
2	50	0x32
3	51	0x33
4	52	0x34
5	53	0x35
6	54	0x36
7	55	0x37
8	56	0x38
9	57	0x39

Table 12

As can be seen in the above table, the difference between a natural number and its corresponding position in ASCII table is 48 in decimal, 0x30 in hexadecimal. A BCD only needs to be increased by 0x30 will become its character.

In addition, for compressed BCD which contains two BCDs in one 8-bit variable, the separation step would be taken first before the addition. The separation step shows as follows.

```
BCD_to_ASCII
    clrf ASCII_H
    clrf ASCII_L
    movfw Compressed_BCD
    andlw 0x0f
    iorlw 0x30
    movwf ASCII_L
    swapf Compressed_BCD, w
    iorlw 0x30
    movwf ASCII_H
    return
```

5.6 Dynamic range

The upper and lower limit of the range are all performed in the form of BCD, having four digits in each limit. The change of the range are controlled by two push buttons which are indicated by '+' and '-'. If the button '+' if pressed, the upper and lower limit will both increased by 10 with the unit of millimeter. And absolutely, of the button indicated by '-' is triggered, both limits will be reduced by 10.

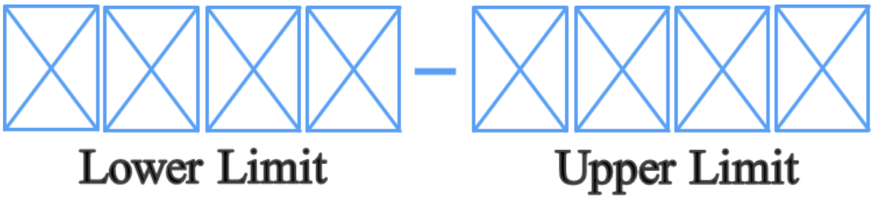
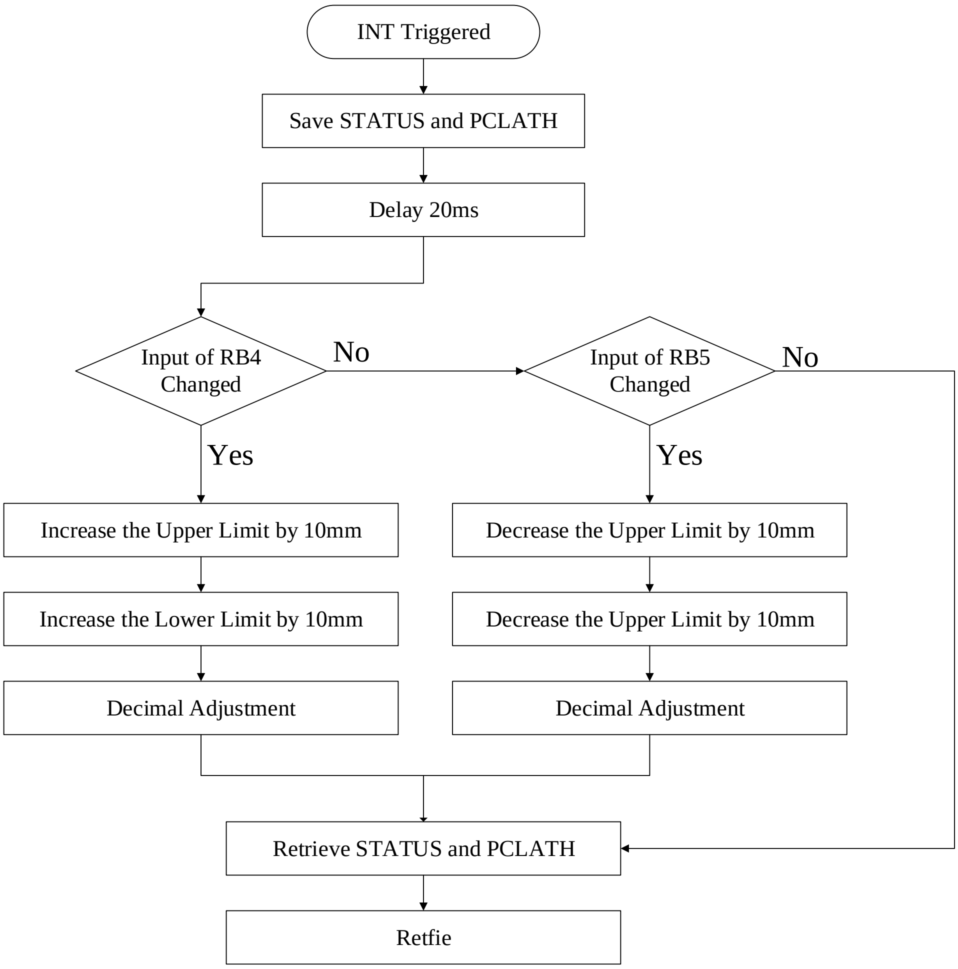


Figure 28

With the respect of software, the routines of dynamic range are all triggered by interrupt. Meanwhile, since the function is not really complicated, the whole process stays in interrupt service routine.

For PIC16f series, the INT interrupt only sticks to RB0, which is insufficient for the requirement. Therefore, RB port change interrupt is adopted instead. In this case, RB4 is used for '+' pushbutton, and RB5 is working with pushbutton '-'.



Flowchart 2

Whenever the pushbuttons are pressed, the program will go to the interrupt service routine (ISR) even if the chip is in a sleep mode.

The Flowchart 2 above shows how the interrupt service routine works to perform dynamic range.

During an interrupt, only the return PC value is saved on the stack. Therefore, these variables will be defined to save the contents of STATUS, W and PCLATH registers.

After saving the key registers, the first step is to delay 20ms in order to get rid of the effect of jitter or disturbing. Then, the chip will determine which push button is triggered to increase or decrease the range correspondingly, followed by a function of decimal adjustment.

Before exiting the ISR, the former contents of the key registers will be retrieved to improve the system stability.

Decimal Adjustment:

As mentioned before, the digit in upper and lower limits are all in the form of BCD numbers. While doing the addition of decimal in binary system, thing gets a little bit complex.

Decimal	BDC	Hexadecimal
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	/	A
11	/	B
12	/	C
13	/	D
14	/	E
15	/	F

Table 13

As shown in the truth table above, BCD is segment of Hexadecimal. While adding a 1 to 9, the result is A. In order to get the result of 0 with a carry bit set, the process to cross the period from A to F is so called decimal adjustment.

The mechanism of decimal adjustment is quite clear. While adding 1 to 9, an extra 6 will be added as well for offset and for decrement, vice versa.

5.7 LED Indication

5.7.1 LED Cycle Sequentially

The whole program starts with a LED cycle sequentially. It basically just turns on the LEDs in turns and turns them off reversely. This routine is used to indicate that the chip's starting over goes well.

5.7.2 LED Region Indication

The theory is quite easy and meets the requirement very well. While the distance measured is between the lower limit and upper limit, the chip will turn on the green LED.

While the result is not within the range, the program will turn on the yellow LED instead, provided that the measured result is not too far away from the upper and lower limits. The more specific definition shows as follows:

$$Lower\ Limit1_{yellow} = Lower\ Limit_{green} - 100$$

$$Upper\ Limit1_{yellow} = Lower\ Limit_{green}$$

$$Lower\ Limit2_{yellow} = Upper\ Limit_{green}$$

$$Lower\ Limit2_{yellow} = Lower\ Limit_{green} + 100$$

Moreover, it is absolutely the left region will be indicated by red LED.

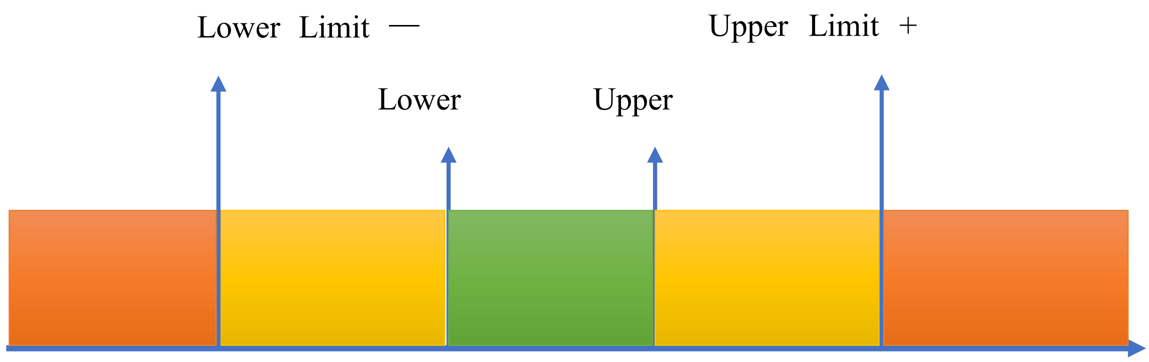
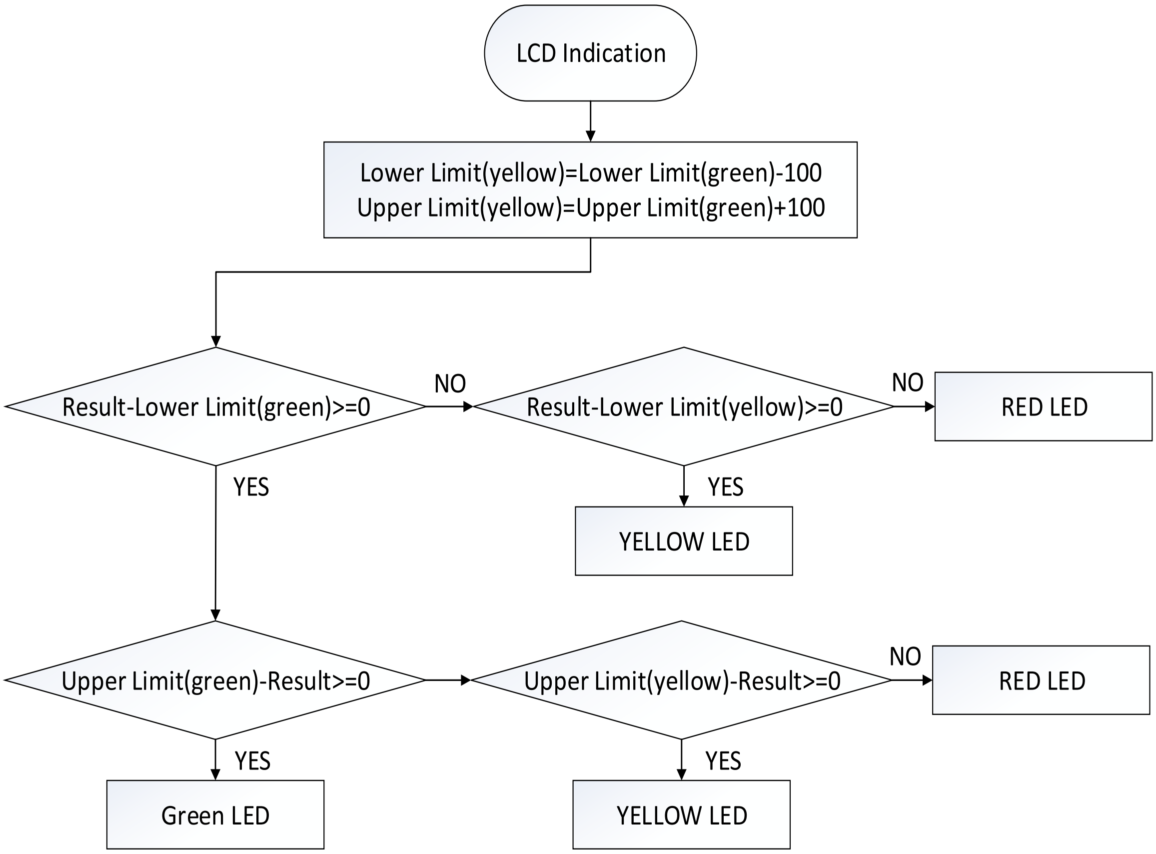


Figure 29



Flowchart 3

Because the value of result, upper and lower limits of green and yellow are all performed in BCD numbers, the flowchart in reality will be more complicated, comparing the values digit by digit. The basic process is shown as above.

5.8 Sleep Mode

Sleep mode is very essential for the embedded system to save energy for longer working time. Meanwhile as required, the shortest time interval between two detection is 80ms. Instead of using a time delay, the sleep mode is adopted.

Each time after one whole process is finished, the chip will go into a sleep mode, waiting for a watchdog timeout to wake up itself.

Generally speaking, a watchdog timeout will be 18ms. In order to compatible with the time delay before, the prescaler is set attached to WDT (watchdog timer) and configured as the rate of 64:1. In this case, the sleep time for PIC is roughly 1.2s.

```
bsf OPTION_REG, 3 ;PSA prescaler to WDT
bsf OPTION_REG, 2 ; rate 64
bsf OPTION_REG, 1 ;
bcf OPTION_REG, 0
```

6 Experiment

After the development of software, it is time to test the accuracy. The scene of the experiment shows as follows.

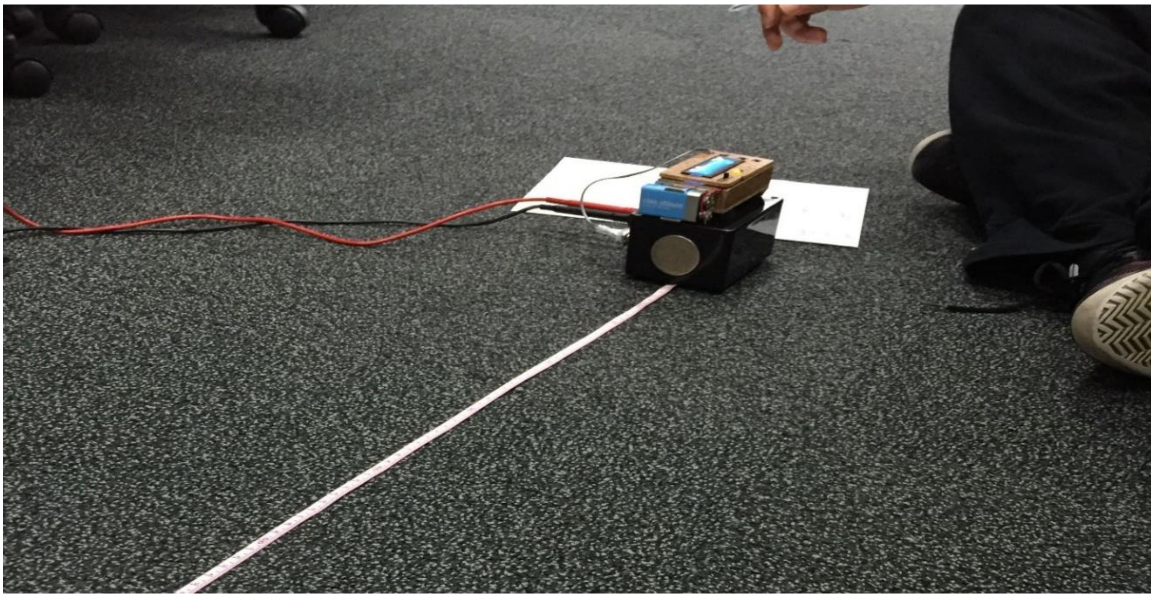


Figure 30

6.1 Experiment before Calibration

Distance	500	600	700	800	900	1000	1100	1200	1300
Output	520	614	708	802	909	1015	1122	1230	1334
Error	**4.00%**	2.33%	1.14%	0.25%	1.00%	1.50%	2.00%	**2.50%**	**2.62%**

Table 14

As can be seen in the table above, the numbers performed in bold font don't fit the accuracy requirement (97.5%).

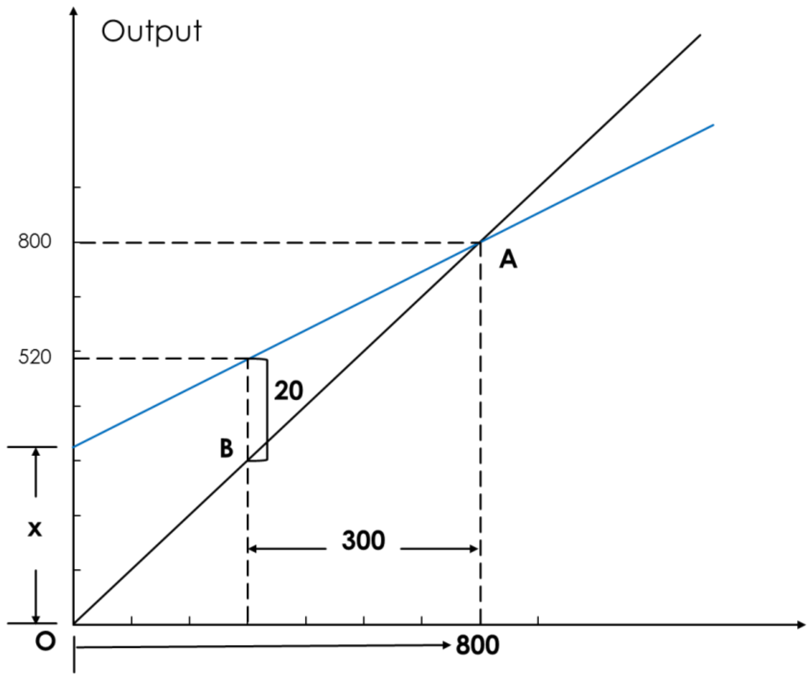


Figure 31

The output of the PIC fits the blue line very well and the black line is the desirable output.

Here is where geometry works.

$$\frac{20}{x} = \frac{AB}{AO} = \frac{300}{800}$$

$$x = 53.3$$

Meanwhile, the slope of the blue line is not high enough.

$$k = \frac{800 - x}{800} = 0.933$$

$$\frac{1 - 0.933}{0.933} = 7.143\%$$

Which means the parameter needs to be increased by 7.143%.

6.2 Experiment after Calibration

Distance	500	550	600	650	700	750	800	850	900
Output	504	553	604	653	704	753	803	853	903
Error	0. 800%	0. 545%	0. 667%	0. 462%	0. 571%	0. 400%	0. 375%	0. 353%	0. 333%
Difference	4	3	4	3	4	3	3	3	3
Distance	950	1000	1050	1100	1150	1200	1250	1300	
Output	953	1004	1054	1103	1154	1204	1254	1304	
Error	0. 316%	0. 400%	0. 381%	0. 273%	0. 348%	0. 333%	0. 320%	0. 308%	
Difference	3	4	4	3	4	4	4	4	

Table 15

The Table 15 clearly indicates that the errors of the output after calibration are all smaller than 1% which means the accuracy of the measurement is above **99%**. In addition, the data of difference also reveal that the error is technically linear. If the origin shifts a little bit, there will be roughly no error anymore.

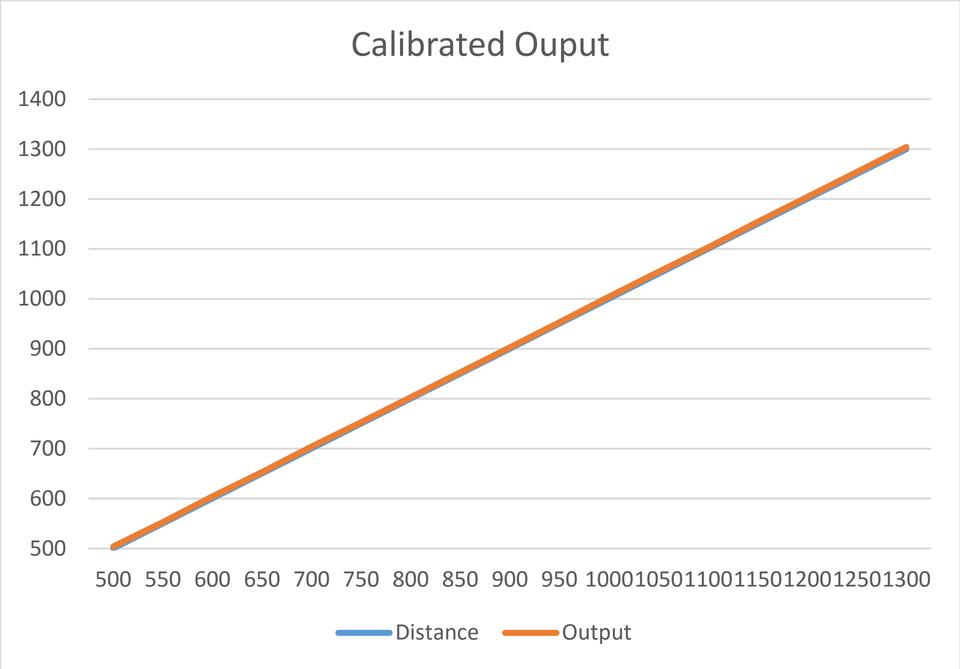


Figure 32

6.3 Experiment of Radius of curvature detected

The vertical view of the method of measuring radius of curvature shows as below. The LCD1602 and ultrasonic sensor are on the left side. It is basically about the maximum length from the axis that the sensor can detect and give a reasonable output while the projection of the object on the axis is at different position.

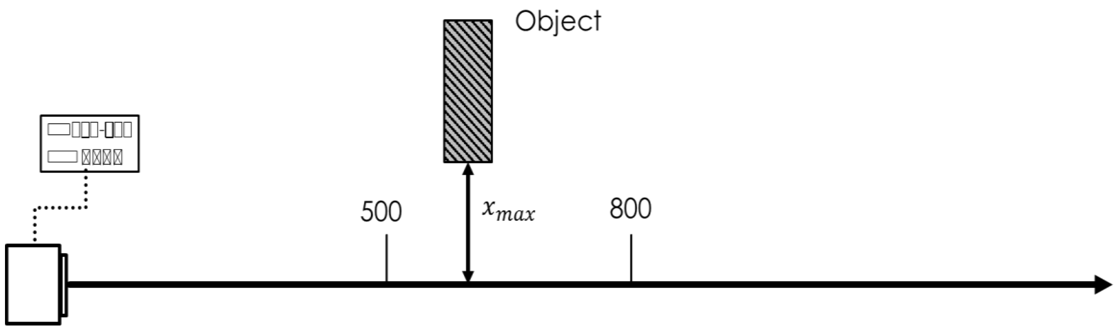


Figure 33

In order to get a convincing result and achieve the pervasive phenomenon, the measurement is repeated three times.

The way to determine whether the output given by the LCD is reasonable or not is:

1. Depending on the function of dynamic range, let the distance of object's projection on the axis from the sensor in the middle of the upper and lower limit of the green band.
2. Move the object away from the axis vertically and slowly.
3. When the amber LED is turned on, then the output is unreasonable, measure the length of x_{max}.

Distance	500	600	700	800	900	1000	1100	1200	1300
Experiment1	103	124	146	165	184	203	221	230	250
Experiment2	112	122	140	157	183	196	222	233	254
Experiment3	115	137	150	160	190	206	224	239	262
Average	110	128	145	161	186	202	222	234	255

Table 16

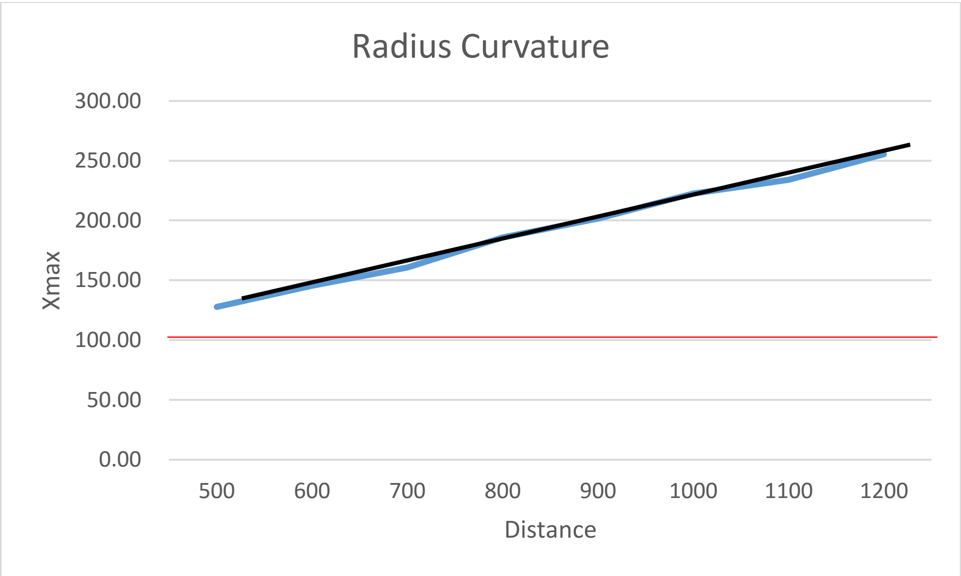


Figure 34

Figure 34 illustrates that the radius of curvature detected larger than 100mm.
In addition, the result of the experiment actually fits the black line very well.
In rough calculation,

$$\theta = tan^{-1}(\frac{255 - 110}{1300 - 500})$$

$$\theta = 10.27°,\ 2\theta = 20.54°$$

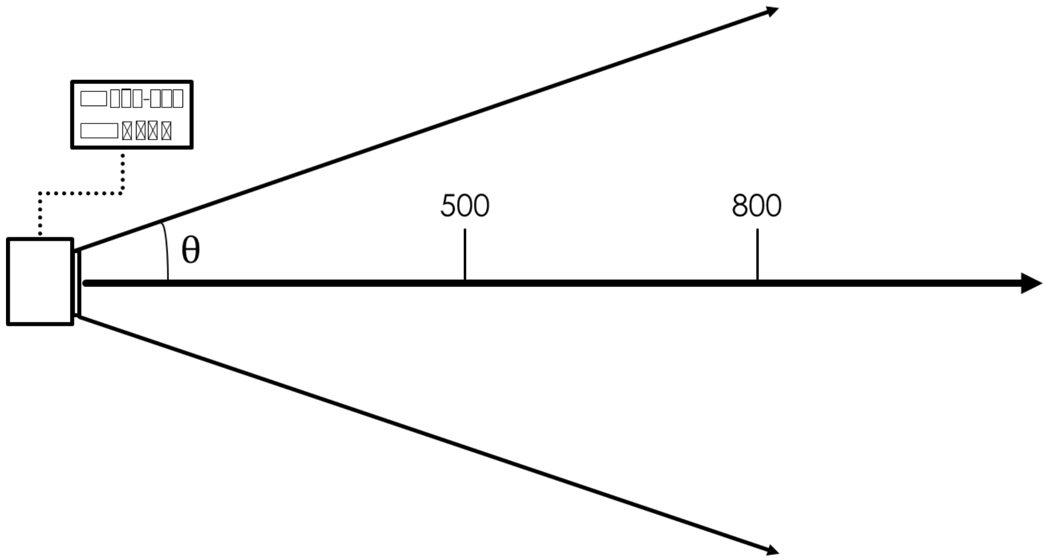


Figure 35

In conclusion, the angle of radiation of the sensor is roughly20.54°.

7 Conclusion

The test and experiment have proved that our USM (Ultrasonic meter) series devices meet the requirement of this project. To improve the accuracy and user feeling, the downloader port allowed the users or technicians to upgrade or customize the control system conveniently. Moreover, the LCD1602 module and the well-designed layout of PCB provide a user-friendly interface. Also, with the hand-made soft wood case, this product is not only reliable and stable, but also has a good touch feeling.

Nowadays, it becomes easier for engineers to develop an embedded system. For instance, the prevalence of Arduino, moreover, module based design in Matlab, by which engineers don't even need to write a single line of code. In this perspective, the possibility of using assembly language in future life may be quite small.

Nevertheless, it may be easy to development a program in high level language, which does not mean that learning assembly language is useless. The development of program is only the first step and may be a small part of a whole project, the process of debugging and implementation usually takes the majority of effort. With the deepen insight from learning assembly language, the engineer would be more capable of solving the potential problems.

8 Reference

[1] Ultrasonic Distance Measurement with the MSP430
http://www.ti.com/lit/an/slaa136a/slaa136a.pdf

[2] Robot obstacle detection and avoidance with the devantech srf05 ultrasonic range finder.
http://picaxe.hobbizine.com/srf05.html

[3] PIC16F87XA datasheet
http://ww1.microchip.com/downloads/en/DeviceDoc/39582C.pdf

[4] LM78LXX Series 3-Terminal Positive Regulators

http://www.ti.com/lit/ds/symlink/lm78l05.pdf

[5] LCD1602 MOUDULE SPECIFICATION

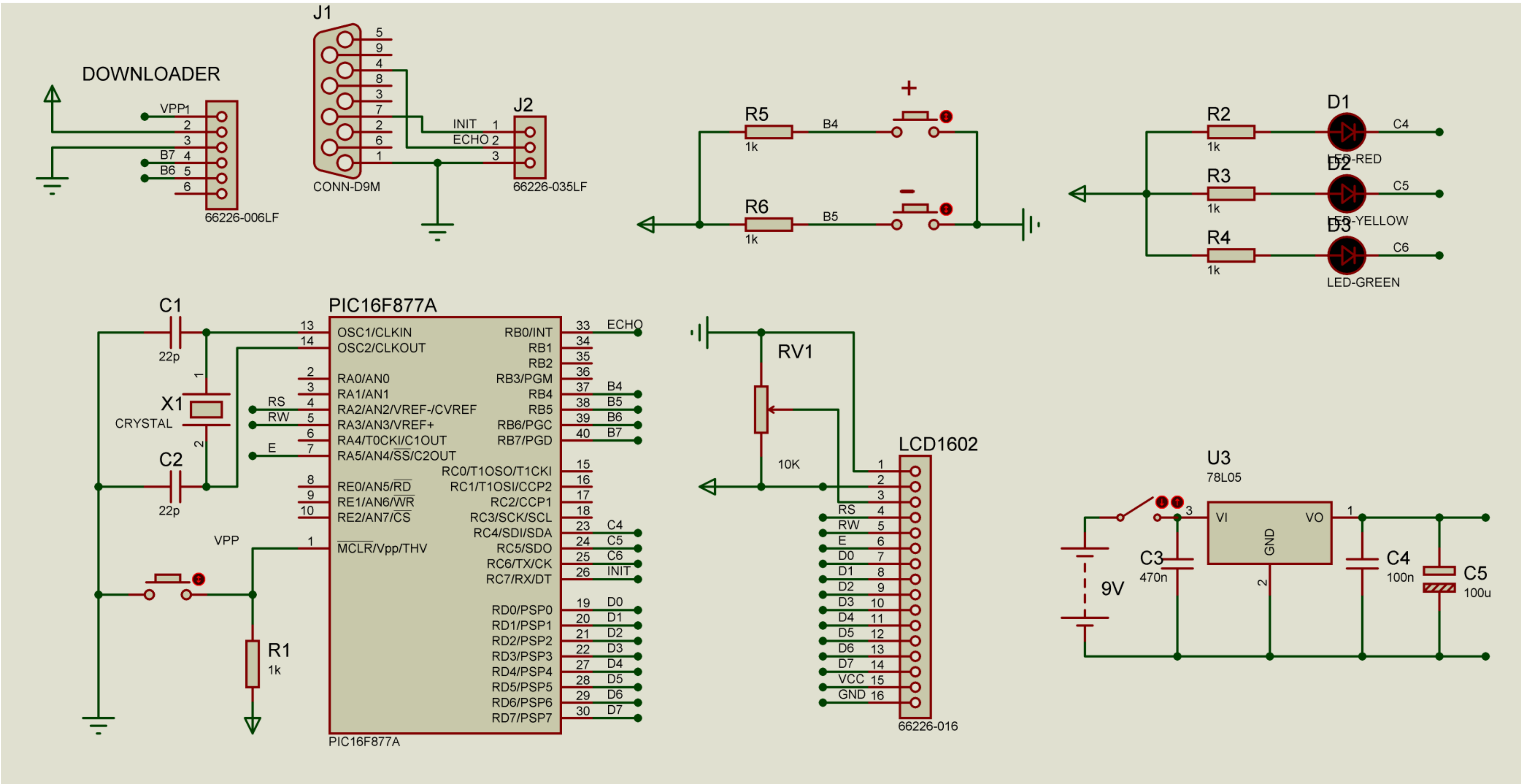
http://www.elecrow.com/download/LCD1602.pdf

[6] Polaroid 6500 ultrasonic transducer datasheet

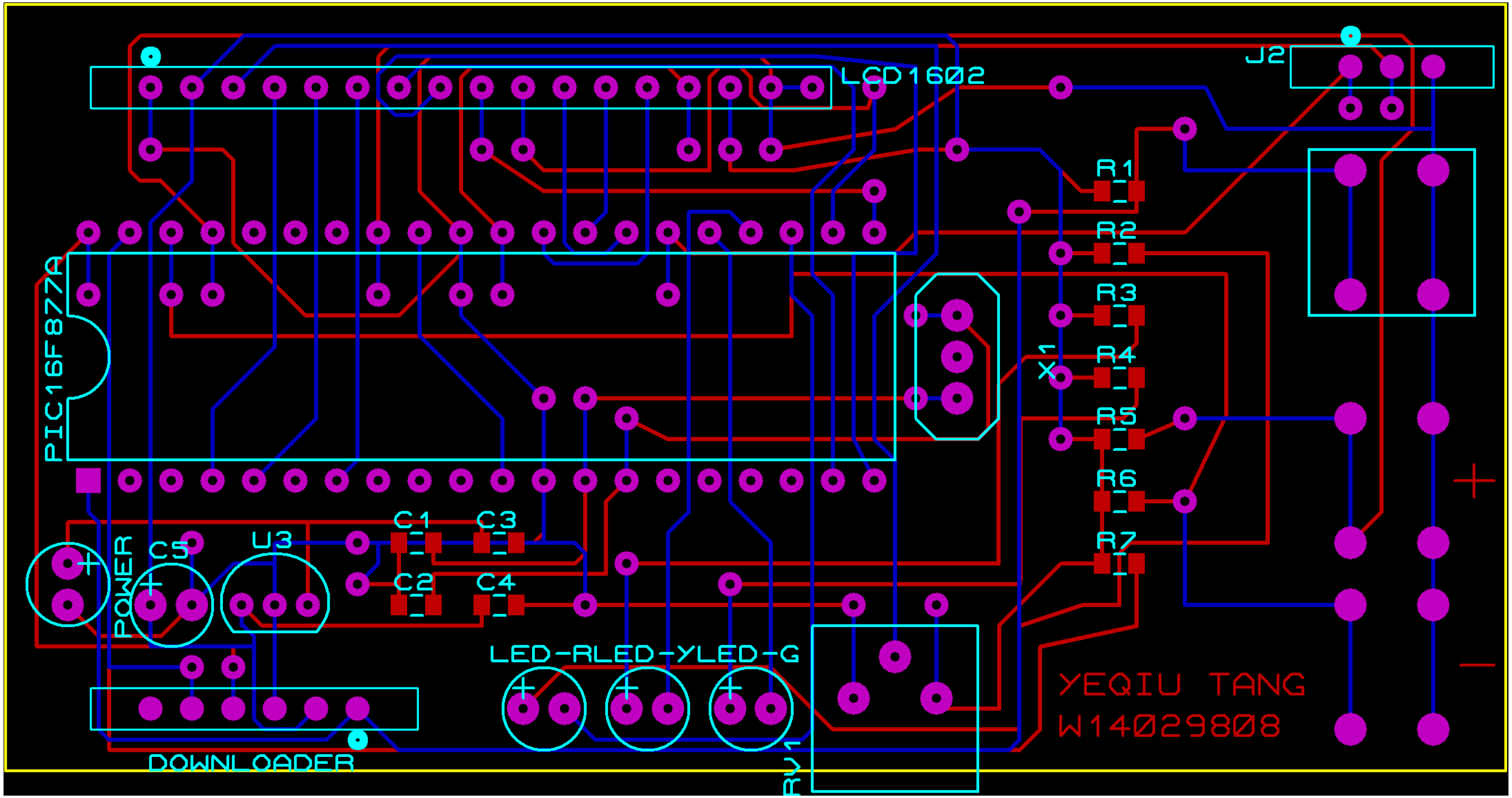
http://www.robotstorehk.com/6500.pdf

9 Appendix

Schematic Design:

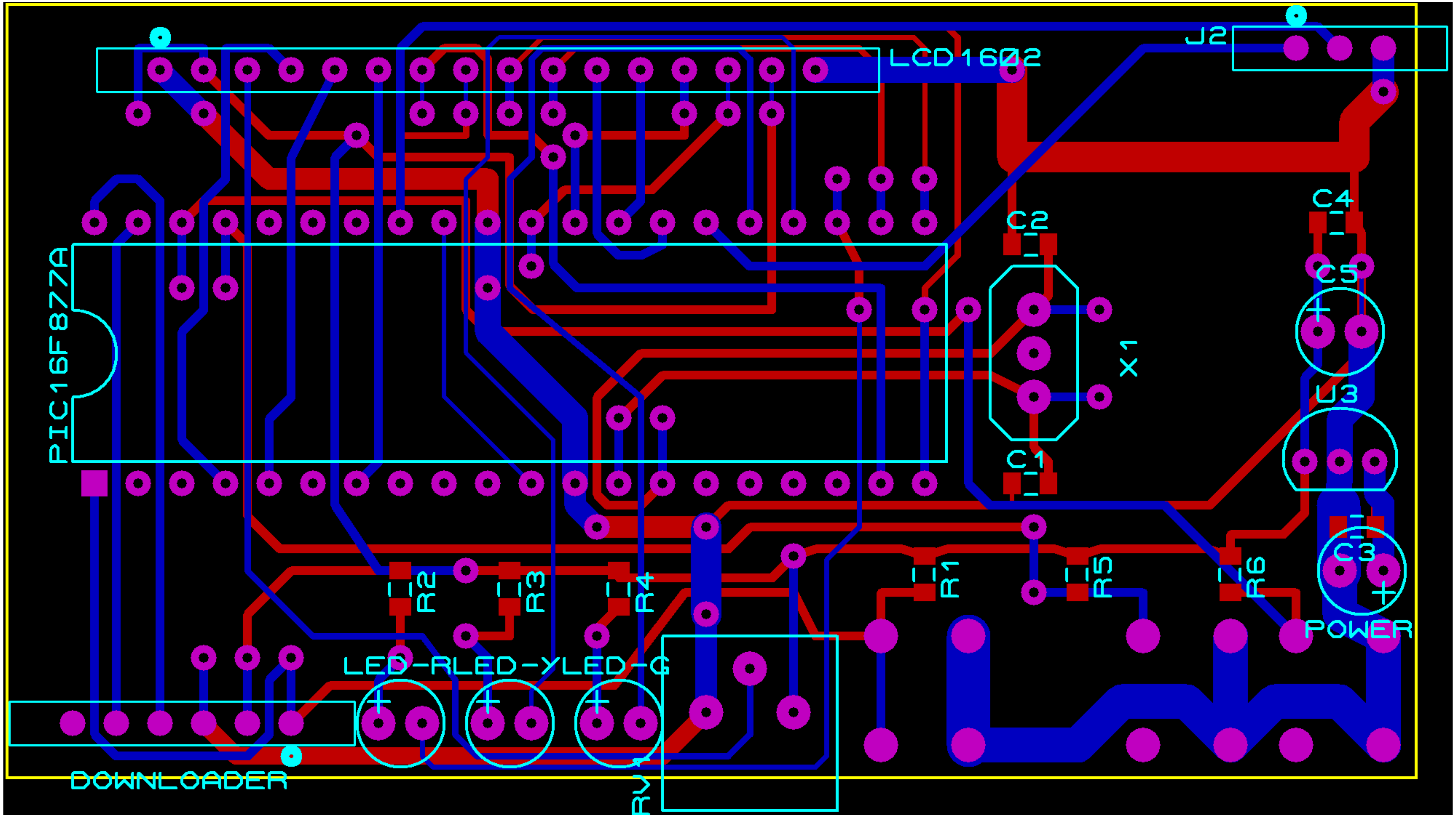


USM-1 PCB:



LCD1602
J2
PIC16F877A
R1
R2
R3
R4
R5
R6
R7
X1
C1
C3
C2
C4
C5
U3
POWER
LED-RLED-YLED-G
RV1
DOWNLOADER
YEQIU TANG
W14029808

USM-2 PCB:



LCD1602
J2
PIC16F877A
C2
C4
C5
X1
U3
C1
C3
R1
R2
R3
R4
R5
R6
POWER
LED-R
LED-Y
LED-G
DOWNLOADER
RV1

CODE:

```
INIT
    CALL Variable_INIT
    CALL LED_INIT
    CALL LCD_INIT
    CALL Timer_INIT
    CALL INT_Init
MAIN
        clrf TMR1H
        clrf TMR1L
        bcf STATUS, RP0
        bcf STATUS, RP1
        bsf T1CON, 0
        bsf PORTC, 7
Wait    btfss PORTB, 0
        goto Wait
        bcf T1CON, 0
        call loadAB
        call multiply
        call LoadBin
        call BtoBCD
        call Result_Adjust
        call BCD_Display
        call Range_Display
        call Compare
        bcf PORTC, 7
        CLRWDT
        SLEEP
        NOP
        goto MAIN
```

```
BCD_to_ASCII
    clrf ASCII_H
    clrf ASCII_L
    movfw Compressed_BCD
    andlw 0x0f
    iorlw 0x30
    movwf ASCII_L
    swapf Compressed_BCD, w
    iorlw 0x30
    movwf ASCII_H
    return
```

```
        CODE
BtoBCD
        bcf STATUS, C
        movlw .32           ;preset a 32 times loop
        movwf temp_shift2
        clrf BCD4           ;Clear all the outputs
        clrf BCD3
        clrf BCD2
        clrf BCD1
        clrf BCD0
Loop32
        rlf Bin0, f         ;rotate to the left
        rlf Bin1, f
        rlf Bin2, f
        rlf Bin3, f
        rlf BCD0, f
        rlf BCD1, f
        rlf BCD2, f
        rlf BCD3, f
        rlf BCD4, f
        decfsz temp_shift2, f
        goto ADJDEC
        retlw 0
```

```
ADJDEC
    movlw BCD0
    movwf FSR
    call ADJBCD
    movlw BCD1
    movwf FSR
    call ADJBCD
    movlw BCD2
    movwf FSR
    call ADJBCD
    movlw BCD3
    movwf FSR
    call ADJBCD
    movlw BCD4
    movwf FSR
    call ADJBCD
    goto Loop32


ADJBCD
    movlw 0x03
    addwf INDF, w
    movwf temp_BCD
    btfsc temp_BCD, 3
    movwf INDF
    movlw 0x30
    addwf INDF, w
    movwf temp_BCD
    btfsc temp_BCD,7
    movwf INDF
    retlw 0
```

```
;************************* LED Indication  *************************************
;******************************************************************************
adjustC
		movfw LoLimit1
		andlw 0x0f
		iorlw 0xA0
		movwf LLimit1
		decf LLimit2
		return
Compare
;Calculate the value of the upper and lower limits for YELLOW Band
		movfw LoLimit2
		movwf LLimit2

		movfw LoLimit1
		movwf LLimit1
		movfw LoLimit1
		andlw 0x0f0
		btfsc STATUS, Z		;If LoLimit1 is zero
		call adjustC
		movlw 0x10
		subwf LLimit1, w	;decrease LLimit1 by 0x10
		movwf LLimit1

		movfw LoLimit0		;LLimit0 = LoLimit0
		movwf LLimit0

		movfw UpLimit2		;ULimit2 = UpLimit2
		movwf ULimit2
		movfw UpLimit1		;if UpLimit1 = 0x9X
		btfss UpLimit1, 7
		goto $+5
		btfss UpLimit1, 4
		goto $+3
		incf ULimit2		;add ULimit2 with carry bit
		movlw 0x0F1
		addlw 0x10
		movwf ULimit1
		movfw UpLimit0		;ULimit0 = UpLimit0
		movwf ULimit0
;******************************************************************************
```

```
;*****************************************************************************
        ;Gren band comparison
        swapf BCD1, w           ;get the least significant digit of result
        andlw 0x0f
        movwf BCD1

        movfw LoLimit2          ;compare the most significant digit
        subwf BCD3, w
        btfsc STATUS, Z
        goto $+4
        btfss STATUS, C     ;
        goto  Compare2      ;C=0
        goto UpSide

        movfw LoLimit1  ;compare the second most significant digit
        subwf BCD2, w
        btfsc STATUS, Z
        goto $+4
        btfss STATUS, C
        goto  Compare2
        goto UpSide

        movfw LoLimit0  ;compare the least significant digit
        subwf BCD1, w
        btfss STATUS, C
        goto  Compare2
```

```
USide
        movfw BCD3          ;compare the most significant digit
        subwf ULimit2, w
        btfsc STATUS, Z
        goto $+4
        btfss STATUS, C
        goto  RED           ;C=0
        goto YELLOW

        movfw BCD2          ;compare the second most significant digit
        subwf ULimit1, w
        btfsc STATUS, Z
        goto $+4
        btfss STATUS, C
        goto  RED
        goto YELLOW

        movfw BCD1          ;compare the least significnat digit
        subwf ULimit0, w
        btfss STATUS, C
        goto  RED
        goto  YELLOW
```

```
;*****************************************************************************
        ;Yellow band Comparison
Compare2
        movfw LLimit2   ;compare the most significant digit
        subwf BCD3, w
        btfsc STATUS, Z
        goto $+4
        btfss STATUS, C     ;
        goto  RED
        goto USide

        movfw LLimit1     ;compare the second most significant digit
        subwf BCD2, w
        btfsc STATUS, Z
        goto $+4
        btfss STATUS, C
        goto  RED
        goto USide

        movfw LLimit0      ;compare the least significant digit
        subwf BCD1, w
        btfss STATUS, C
        goto  RED
```

```
UpSide
        movfw BCD3      ;compare the most significant digit
        subwf UpLimit2, w
        btfsc STATUS, Z
        goto  $+4
        btfss STATUS, C
        goto  Compare2      ;C=0
        goto GREEN

        movfw BCD2      ;compare the second most significant digit
        subwf UpLimit1, w
        btfsc STATUS, Z
        goto $+4
        btfss STATUS, C
        goto  Compare2
        goto GREEN

        movfw BCD1      ;compare the last signicant digit
        subwf UpLimit0, w
        btfss STATUS, C
        goto  Compare2
        goto  GREEN
;Operation for LED
GREEN   movlw 70H
        iorwf PORTC, f
        bcf PORTC, 6
        return
YELLOW  movlw 70H
        iorwf PORTC, f
        bcf PORTC, 5
        return
RED     movlw 70H
        iorwf PORTC, f
        bcf PORTC, 4
        return
adjust0     ;if LoLimit1 is zero
        decf LoLimit2, f        ;decrease LoLimit2 by 1
        movlw 9AH               ;let LoLimit1=0x9A
        movwf LoLimit1
        return
adjust1     ;if UpLimit1 is zero
        decf UpLimit2, f        ;decrease UpLimit2 by 1
        movlw 9AH               ;let UpLimit1=0x9A
        movwf UpLimit1
        return
```

```
decrease
        movf LoLimit1
        btfsc STATUS, Z     ;Test if LoLimit1 is zero
        call adjust0
        decf LoLimit1, f
        btfss LoLimit1, 3    ;if LoLimit1 is 0xXf
        goto $+9
        btfss LoLimit1, 2
        goto $+7
        decf LoLimit1, f     ;substracted by extra 6
        decf LoLimit1, f
        decf LoLimit1, f
        decf LoLimit1, f
        decf LoLimit1, f
        decf LoLimit1, f

        movf UpLimit1
        btfsc STATUS, Z     ;Test if UpLimit1 is zero
        call adjust1
        decf UpLimit1, f
        btfss UpLimit1, 3    ;if UpLimit1 is 0xXf
        goto $+9
        btfss UpLimit1, 2
        goto $+7
        decf UpLimit1, f     ;substracted by extra 6
        decf UpLimit1, f
        decf UpLimit1, f
        decf UpLimit1, f
        decf UpLimit1, f
        decf UpLimit1, f
        goto retrieve
```

```
;*******************************        ISR     *******************************
;*******************************************************************************
ISR
        movwf w_temp
        swapf STATUS, w
        clrf STATUS
        movwf status_temp
        movf PCLATH, w
        movwf pclath_temp
        clrf PCLATH
        ;****************  start   *****************
        call DELAY_20ms
        btfss PORTB, 5      ;if RB5 is pressed
        goto decrease
        call DELAY_20ms
        btfss PORTB, 4      ;if RB4 is pressed
        goto increase
        goto retrieve
```

```
increase
        incf LoLimit1, f
        btfss LoLimit1, 3       ;test if LoLimit1 is 0xXA
        goto $+5
        btfss LoLimit1, 1
        goto $+3
        movlw 6H                ;increased by extra 6
        addwf LoLimit1,f

        btfss LoLimit1, 7        ;test if LoLimit1 is 0xAX
        goto $+5
        btfss LoLimit1, 5
        goto $+3
        clrf LoLimit1
        incf LoLimit2              ;add carry bit to LoLimit2

        incf UpLimit1, f
        btfss UpLimit1, 3          ;test if UpLimit1 is 0xXA
        goto $+5
        btfss UpLimit1, 1
        goto $+3
        movlw 6H                   ;offset
        addwf UpLimit1, f

        btfss UpLimit1, 7          ;test if LoLimit1 is 0xAX
        goto $+5
        btfss UpLimit1, 5
        goto $+3
        clrf UpLimit1
        incf UpLimit2
        ;****************   end   ******************
retrieve
        bcf INTCON, 0
        movf pclath_temp, w
        movwf PCLATH
        swapf status_temp, w
        movwf STATUS
        movf w_temp
        retfie
```

```
;*****************************************************************************
; Load time value to ValueA and Scaling factor values to ValueB
loadAB
        movf    TMR1H, w
        movwf   ValueAH
        movf    TMR1L, w
        movwf   ValueAL

        movf    ScaleH, w
        movwf   ValueBH
        movf    ScaleL, w
        movwf   ValueBL
        return

;*****************************************************************************
;32-bit outcome with higher significant value in ValueBH:ValueBL and lower
;significant value in ValueCH:ValueCL
multiply
        call    setup               ;rotate ValueD
mloop   rrf     ValueDH, f
        rrf     ValueDL, f
        btfsc   STATUS, C           ;if carry bit is set, offset will be done
        call    add
        rrf     ValueBH, f
        rrf     ValueBL, f
        rrf     ValueCH, f
        rrf     ValueCL, f
        decfsz  temp_shift, f       ;loop untile all bits are finished
        goto    mloop
        retlw   0

;-----------------------------------------
setup   movlw   .16                 ;check for 16 times
        movwf   temp_shift
        movf    ValueBH, w
        movwf   ValueDH
        movf    ValueBL, w
        movwf   ValueDL
        clrf    ValueBH
        clrf    ValueBL
        retlw   0
```

```
            bsf OPTION_REG, 3 ;PSA prescaler to WDT
            bsf OPTION_REG, 2 ; rate 64
            bsf OPTION_REG, 1 ;
            bcf OPTION_REG, 0
;-----------------------------------
;addtion value ValueB + ValueA ->ValueB
add     bcf     Flags, 0
        movf    ValueAL, w          ;add LSB
        addwf   ValueBL, f          ;carry bit offset
        btfss   STATUS, C
        goto    add1
        incf    ValueBH
        btfsc   STATUS, Z
        bsf     Flags, 0
add1
        movf    ValueAH, w          ;ValueB= ValueA + ValueB
        addwf   ValueBH, f          ;add MSB
        btfsc   Flags, 0
        bsf     STATUS, C
        retlw   0
;************************************************
        return
        end
```

```
;*******************Write Comand***************
SEND_DATA
  MOVWF PORTD
  BSF PORTA,RS
  BCF PORTA,RW
  BCF PORTA,E
  CALL DELAY_20ms
  BSF PORTA,E
  RETURN

CLEAR
  MOVLW 01H
  MOVWF PORTD
  CALL ENABLE
  RETURN

Locate1
  movlw 80H
  movwf PORTD
  call ENABLE
  return
Locate2
  MOVLW 0C0H
  MOVWF PORTD
  CALL ENABLE
  return
```