UM-SJTU Joint Institute
Microprocessor Based System Design
(VE373)

Project Report

Group 3
Smart Pressure-Sensitive Light

Minqing Sun 519370910005

Shuhan Wang 519370910087

Zhuangyun Wang 519370910142

Date: August 5, 2022

1 Introduction

In this project for course VE373, our team have designed a smart pressure-sensitive light that can realize real-time interaction with people while demonstrating our knowledge of embedded system using PIC 32.

1.1 High-level Description

A 4- times-4 thin-film pressure sensor that has 16 independent sensor units collecting pressure values at different locations is used to detect pressure. The sensor is connected to an ADC extended module that is controlled by the microprocessor PIC 32 to select which of the 16 ADC values is to be sampled. What is obtained from the pressure sensor is to be displayed on the LED light board as well as on the indicator light connecting to PIC32. An OFF button is used to turn off the entire device.

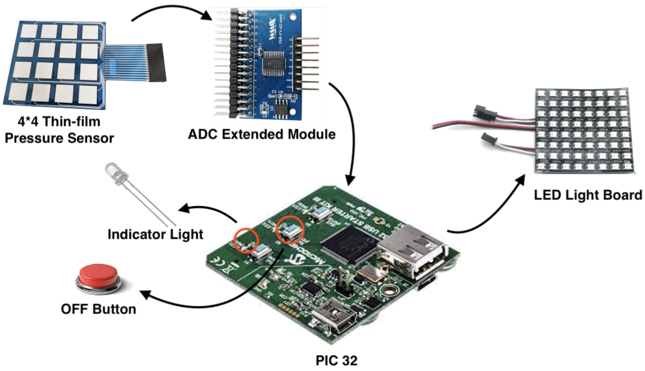

Figure 1: High-level diagram

1.2 Brief Description of Functions Realized

- With different locations, the LED light board will present different colors; there are 16 colors correspond to the sixteen sensor units.
- With different number of units pressed, the patterns shown on the board are different.
- The brightness of the indicator light shows how heavy the units are pressed.
- An off button controls the whole device.

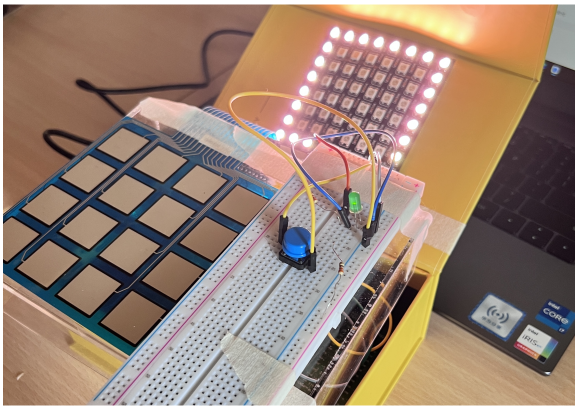

Figure 2: Designed system

2 Detailed Design

2.1 Component Level Diagram

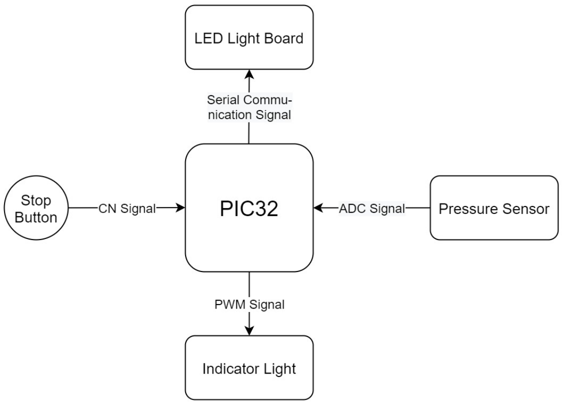

Figure 3: Component Level Diagram

2.2 Pressure Sensor - ADC

The most important input is from pressure sensor. Pressure sensor offers PIC32 with 16 independent input analog signals that can be collected by the ADC peripheral. Also, an extended module is used which encodes a 4-bit binary signal to a 16-bit one-hot signal so that only one input from one of the sub-sensors will be read at a time. Using the interrupt flag to indicate that a analog value has been successfully transferred, an array of size 16 is used to store these values for further analysis. And with pressure larger than 70, the sub-sensor is considered pressed.

2.3 LED Light Board - Serial Communication

The LED light board is our main output device. It has 64 pixels, every pixel can display 24-bit true color light. The light board requires a unique communication protocol that is based on strict timing sequence. The timing requirement has to be exact to $0.1\mu s$ so that correct response can be displayed.

2.3.1 Communication Protocol

The protocol dictates that every pixel on the LED board requires 24-bit binary signal to be lightened up. The 24-bit signal contains three sets of 8-bit RGB color information. And these 24-bit are sent one by one until total 64 pixels have received their instructions.

For every 24-bit signal, it is represented by a serial of wave with $1.25\mu s$ as its period and changing duty cycle. One bit is one period. If the duty cycle is larger than 50%, the bit equals to 1, vice versa.

If the board hasn't received any high voltage signal for $280\mu s$, the board will stop receiving data and display the result.

2.3.2 Function Realization

To realize the unique communication protocol, functions are built to pack and send the signals in our program.

24 bits are packed into function called send_rgb, which receives 3 sets of 8-bit RGB color information. And in related color display function, The function send_rgb is called 64 times to generate colors in different pixels. The choice of different display functions is controlled by the mode selection variable given by the main function.

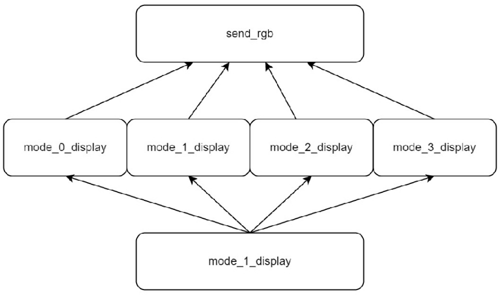

Figure 4: Function Realization of Protocol

2.4 Indicator Light - PWM

Indicator light is used to indicate the level of pressure exerted on the sensor. By scanning the input analog value array, the maximum pressure applied on the sensor is found. The maximum value is the indicator of the PWM duty cycle that interactively showing the pressure level using the brightness of the light.

2.5 Stop Button - Change Notice Interrupt

With stop button connected to one of the CN port, the disable signal is detected. Once the CN interrupt will terminate the device, clear the LED output and the indicator light.

3 Test Plan and Test Result

3.1 Different Modes of display of LED Light Board

Our project involves different modes of display of LED light board in reaction to pressing different number of sensor units. Specifically, there are four modes which corresponds to the number of sensor unit pressed equals 0, 1, 2 and 3. In each mode, the color of the display depends on the location of the sensor unit. Each unit corresponds to one color.

3.1.1 Mode 0

When no sensor unit is pressed, there will be a light effect saying hi and showing pressing instruction. Here we test if the light can show the light effect in the specified order and generate random color each time. The color is generated randomly as we don't touch the device. The picture below can show that the light can show the light effect in the specified order.

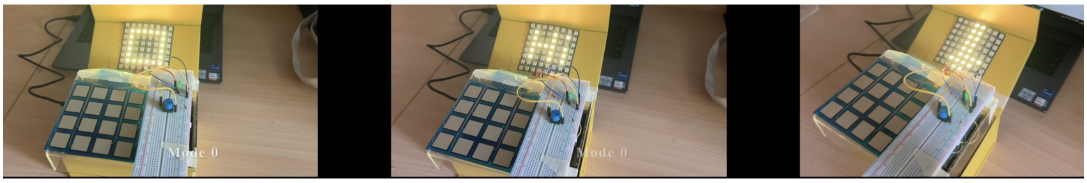

Figure 5: Mode 0 Test Result

3.1.2 Mode 1

When one sensor unit is pressed, the LED light board will show the single color corresponding to the sensor unit. Here we test if pressing one sensor unit can let the LED light board light up with one single color and if changing the pressed unit can change the displayed color. The picture below shows that when we press one sensor unit, the LED light board can light up with one single corresponding color, and when we change a pressed sensor, the color changes.

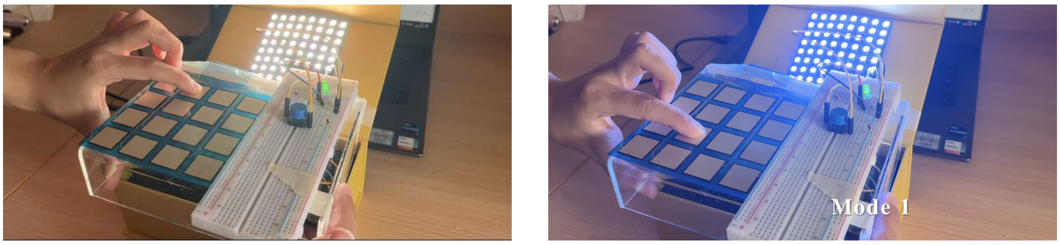

Figure 6: Mode 1 Test Result

3.1.3 Mode 2

When two sensor units are pressed, the LED light board will show the mix color of the corresponding two sensor units and the mix color will blink for two times. Here we test if pressing two sensor units can show the blinking mix color and if changing the units can change the mix color. The picture shows that a blinking mix color is successfully shown when two units are pressed.

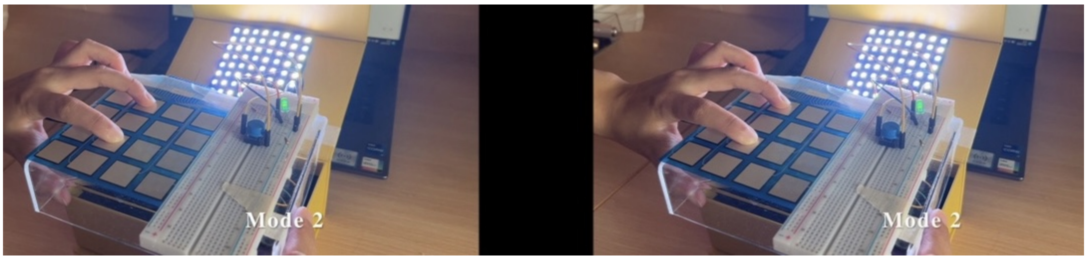

Figure 7: Mode 2 Test Result

3.1.4 Mode 3

When three sensor units are pressed, the LED light board will play a marquee of all the 16 colors. Here we test if pressing three units at one time can let the device play a marquee. The picture shows that a marquee will play smoothly when three sensor units are pressed.

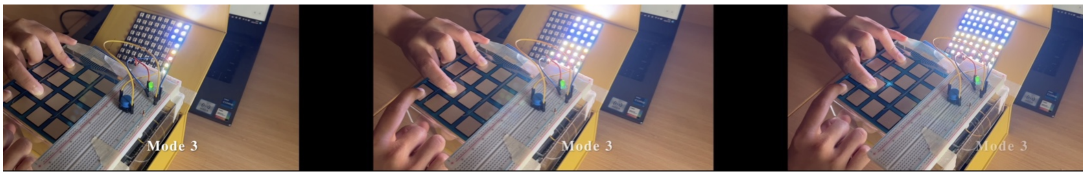

Figure 8: Mode 3 Test Result

3.2 Indicator Light

An indicator light is designed to indicate the pressure exerted on the sensor. When we exert more pressure on the sensor, the light will be brighter. Here we test if the brightness will change with small or large pressure exerted. By comparing this two situations, we can conclude that the indicator light works.

3.3 OFF Button

We have also included a button to turn off the device. When pressed, the LED light board and the indicator light will both go off. Here we test if pressing the button can turn off the lights. Through the test, we can see that the off button works well.

4 Timeline

The refined timeline of this project is shown as below. Compared with the one in proposal, time for coding is compressed due to the slow express delivery, and is adjusted to match the progress of lab sessions.

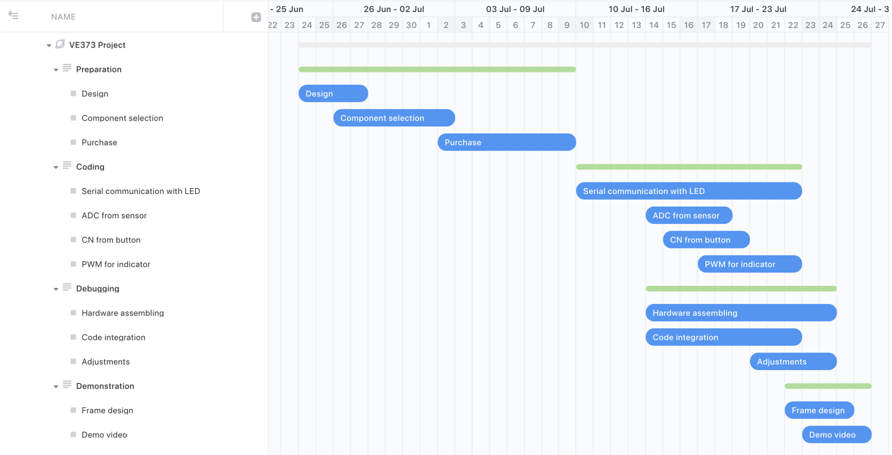

Figure 9: Timeline

5 Component List

Part	Part Number	Price (RMB)
PIC 32	PIC32MX795F512L	N/A
Pressure Sensor	IMM0092	125
ADC Extended Module	FSR 4*4 AD board	72
LED Light Board (x2)	WS2812b	40
Bread Board	N/A	0
Button	N/A	0
LED	N/A	0
Dupont Lines	N/A	0
Total Price		273

Table 1: Component List

6 Source Code

```
#include <stdio.h>
#include <proc/p32mx795f512l.h>
#pragma config FSRSSEL = PRIORITY_0    // SRS Select (SRS Priority 0)
#pragma config FCANIO = OFF             // CAN I/O Pin Select (Alternate CAN I/O)
#pragma config FUSBIDIO = OFF           // USB USID Selection (Controlled by Port Function)
#pragma config FVBUSONIO = OFF          // USB VBUS ON Selection (Controlled by Port Function)

// DEVCFG2
#pragma config FPLLIDIV = DIV_2         // PLL Input Divider (2x Divider)
#pragma config FPLLMUL = MUL_15         // PLL Multiplier (15x Multiplier)
#pragma config UPLLIDIV = DIV_1         // USB PLL Input Divider (1x Divider)
#pragma config UPLLEN = OFF             // USB PLL Enable (Disabled and Bypassed)
#pragma config FPLLODIV = DIV_1         // System PLL Output Clock Divider (PLL Divide by 1)

// DEVCFG1
#pragma config FNOSC = PRIPLL           // Oscillatoits (Primary Osc w/PLL (XT+,HS+,EC+PLL))
#pragma config FSOSCEN = ON            // Secondary Oscillator Enable (Enabled)
#pragma config IESO = ON               // Internal/External Switch Over (Enabled)
#pragma config POSCMOD = HS             // Primary Oscillator Configuration (HS osc mode)
#pragma config OSCIOFNC = ON            // CLKO Output Signal Active on the OSCO Pin (Enabled)
#pragma config FPBDIV = DIV_2           // Peripheral Clock Divisor (Pb_Clk is Sys_Clk/2)
#pragma config FCKSM = CSECMD           // Clock Switching and Monitor Selection (Clock Switch Enable, FSCM Disabled)
#pragma config WDTPS = PS1              // Watchdog Timer Postscaler (1:1)
#pragma config FWDTEN = OFF             // Watchdog Timer Enable (WDT Disabled (SWDTEN Bit Controls))

// DEVCFG0
#pragma config DEBUG = OFF              // Background Debugger Enable (Debugger is disabled)
#pragma config ICESEL = ICS_PGx2        // ICE/ICD Comm Channel Select (ICE EMUC2/EMUD2 pins shared with PGC2/PGD2)
#pragma config PWP = OFF                // Program Flash Write Protect (Disable)
#pragma config BWP = OFF                // Boot Flash Write Protect bit (Protection Disabled)
#pragma config CP = OFF

#include <xc.h>
#include <p32xxxx.h>

typedef unsigned char uchar;
unsigned int colors_rgb[16][3] = {
    //define 16 colors for 16 sensor units
//    {255,250,250},//Snow
//    {255,235,205},//BlanchedAlmond
//    {255,250,205},//LemonChiffon
//    {255,245,238},//Seashell
//    {240,255,240},//Honeydew
//    {230,230,250},//Lavender
//    {255,228,225},//MistyRose
//    {176,224,230},//PowderBlue
//    {154,205,50},//OliveDrab
//    {255,193,37},//Gold
//    {238,216,174},//Wheat
//    {139,69,19},//Chocolate
//    {255,140,105},//Salmon
//    {255,165,0},//Orange
//    {176,48,96},//Maroon
//    {240,255,235},//Azure
    {54,76,39},//Sunflowers_green
    {60,73,8},//Sunflowers_brown
    {166,76,23},//Sunflowers_orange
    {181,132,6},//Sunflowers_yellow
    {137,174,154},//Portrait_green
    {40,101,139},//Portrait_blue
    {203,179,124},//Portrait_mage
    {174,211,202},//Portrait_light_blue
    {174,156,49},//Starry_yellow
    {131,154,183},//Starry_light_blue
    {64,104,164},//Starry_blue
    {26,38,75},//Starry_dark_blue
    {240,223,61},//Chair_yellow
    {119,125,38},//Chair_green
    {205,132,74},//Chair_orange
    {121,34,20}//Chair_red
};
int MODE;
int INDEX[2];

volatile int val[16];
volatile int count;
volatile int whetheran[16]; //array to store whether the sensor unit is pressed
volatile int wherean[16]; //array to store the position of pressed sensor units
volatile int howmany = 0; //indicate how many sensor units are pressed
volatile int enable = 0; //if button not pushed, enable remains 0

/*Delay related*/
void GenUsec(void); //Helper
void GenMsec(void); //Helper
void DelayMsec(uchar num); //Delay num*125us
void DelayUsec(uchar num); //Delay num*1.25us

/*Color and display related*/
void send_rgb(unsigned int r, unsigned int g, unsigned int b);//Generate signal on PORTB given RGB code
void bitbangpixel(unsigned int x);
unsigned long getRainbow(void);


void MCU_init(void) {
    //Output for LED board RB13
    PORTB=0x0000;
    //Input for ADC RB1
    TRISB=0x2;
    //Input for CN RD7
    TRISD = 0x80;
    PORTD = 0x0;
    //Output for extended module S0–S3: RE0–RE3 EN: RE4
    TRISE = 0x0;
    PORTE = 0x0;

    //Timer3 8MHz used for generating delays
    OSCCONbits.PBDIV = 0x0;
    T3CON=0x0;
    PR3=1;
    TMR3=0;

    //Timer2 10kHz used for PWM
    T2CON=0x0;
    PR2=800;
    TMR2=0;

    // Configure Timer3 interrupts
    asm(”di”);
    INTCONSET=0x1000;
    IPC3SET = 0x0000001A;//Interrupt level 6, sub level 2
    IFS0CLR=0x00001000;//Clear interrupt flag
    IEC0SET=0x00001000;//Enable Timer3 interrupt 0000 0000 0000 0000 0001 0000 0000 0000
    asm(”ei”);
}

/* LED display related functions */
void one_color_display(int index){
    int i;
    for(i = 0; i < 64; i++){
        send_rgb(colors_rgb[index][0], colors_rgb[index][1], colors_rgb[index][2]);
    }
}

void two_color_display(int index_1, int index_2){
    unsigned int c1[3];
    unsigned int c2[3];
    int i;
    for (i=0;i<3;++i){
        c1[i] = colors_rgb[index_1][i];
        c2[i] = colors_rgb[index_2][i];
    }
    for (i = 0; i < 64; i++)
    {
        if (i%2 == 0)
        {
            send_rgb(c2[0], c2[1], c2[2]);
        }
        else send_rgb(c1[0], c1[1], c1[2]);
    }
}
void radiation_display(int index){
    unsigned int c[3];
    int loc_0[4] = {27,28,35,36};
    int loc_1[12] = {18,19,20,21,26,29,34,37,42,43,44,45};
    int loc_2[20] = {9,10,11,12,13,14,17,22,25,30,33,38,41,46,49,50,51,52,53,54};
    int loc_3[28] =
        {0,1,2,3,4,5,6,7,8,15,16,23,24,31,32,39,40,47,48,55,56,57,58,59,60,61,62,63};
    int loc_hi[16] = {9,11,12,13,25,26,27,28,29,36,43,50,51,52,53,54};
    int loc_arrow[22] = {13,17,18,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,46,45,50};
    int i;
    int k=0;
    for (i=0;i<3;++i){
        c[i] = colors_rgb[index][i];
    }
    for (i = 0; i<64; i++){
        if (i == loc_0[k]){
            send_rgb(c[0],c[1],c[2]);
            k++;
        }
        else{
            send_rgb(0,0,0);
        }
    }
    DelayMsec(3);
    DelayMsec(20);
    k = 0;
    for (i = 0; i<64; i++){
        if (i == loc_1[k]){
            send_rgb(c[0],c[1],c[2]);
            k++;
        }
        else{
            send_rgb(0,0,0);
        }
    }
    DelayMsec(3);
    DelayMsec(20);
    k=0;
    for (i = 0; i<64; i++){
        if (i == loc_2[k]){
            send_rgb(c[0],c[1],c[2]);
            k++;
        }
        else{
            send_rgb(0,0,0);
        }
    }
    DelayMsec(3);
    DelayMsec(20);
    k=0;
    for (i = 0; i<64; i++){
        if (i == loc_3[k]){
            send_rgb(c[0],c[1],c[2]);
            k++;
        }
        else{
            send_rgb(0,0,0);
        }
    }
    DelayMsec(3);
    DelayMsec(18);
    k=0;
    for (i = 0; i<64; i++){
        if (i == loc_hi[k]){
            send_rgb(c[0],c[1],c[2]);
            k++;
        }
        else{
            send_rgb(0,0,0);
        }
    }
    DelayMsec(3);
    DelayMsec(80);
    k=0;
    for (i = 0; i<64; i++){
        if (i == loc_arrow[k]){
            send_rgb(c[0],c[1],c[2]);
            k++;
        }
        else{
            send_rgb(0,0,0);
        }
    }
    DelayMsec(3);
    DelayMsec(50);
    for (i = 0; i<64; i++){
        send_rgb(0,0,0);
    }
    DelayMsec(3);
}

void mode_0_display(){
    int index = rand() % 15;
    radiation_display(index);
    DelayMsec(3);

}
void mode_1_display(int index){
    //one color occupies the whole LED board when one sensor is pressed, show the corresponding color
    PORTDSET=0x2;
    one_color_display(index);
    DelayMsec(3);
}

void mode_2_display(int index_1, int index_2){
    //two colors flicker alternately when two sensors are pressed
    PORTDSET=0x2;
    two_color_display(index_1, index_2);
    DelayMsec(3);
    DelayMsec(100);
    two_color_display(index_2, index_1);
```

```c
    DelayMsec(3);
    DelayMsec(100);
    two_color_display(index_1, index_2);
    DelayMsec(3);
    DelayMsec(100);
    two_color_display(index_2, index_1);
    DelayMsec(3);
    DelayMsec(100);
}

void mode_3_display(void){
    //Marquee when more than three sensors are pressed
    PORTDSET=0x2;
    unsigned int temp_color[3];
    int j;
    int i;
    int z;
    for(j = 0; j < 64; j++){
        for (i=0; i<64;i++){
            if(i<=j){
                for (z=0; z<3; z++){
                    temp_color[z] = colors_rgb[i%16][z];
                }
                send_rgb(temp_color[0],temp_color[1],temp_color[2]);
            }else{
                send_rgb(0,0,0);
            }
        }
        DelayMsec(3);
    }
}

void mode_display(int mode){
    if(mode==0){
        mode_0_display();
    }
    else if (mode == 1)
    {
        mode_1_display(INDEX[0]);
    }
    else if (mode == 2)
    {
        mode_2_display(INDEX[0],INDEX[1]);
    }
    else if (mode == 3)
    {
        mode_3_display();
    }
}

void bitbangpixel(unsigned int x) {
    //output color series for 24-bit colors
    char i = 24;
    do {
        if ((x >> --i) & 1) {
            PORTB=0x2000;
            Nop();
            PORTB=0x0;
        } else {
            PORTB=0x2000;
            PORTB=0x0;
            Nop();
        }
    } while (i > 0);
}

void send_rgb(unsigned int r, unsigned int g, unsigned int b) {//grb
    //given rgb values output the color of one pixel
    unsigned int color = 0;
    g = g << 16;
    r = r << 8;
    color = g | r | b;
    bitbangpixel(color);
}

/* ADC related functions */
void ADC_interrupt_config(void){
    //Clear ADC interrupt flag
    IFS1CLR = 0x2;
    //Select ADC interrupt priority
    IPC6 = 0x1400000; //5,0
    //Enable ADC interrupt
    IEC1SET = 0x2;
}

void ADC_config(void){
    //Configure analog port pins
    AD1PCFG = 0;
    //Select analog inputs
    AD1CHS = 0x00010000;//AN1 as input
    //Select format of the ADC result
    //AD1CON1SET = 0x000;//Form: 000 integer 16-bit
    //Select conversion trigger source
    AD1CON1SET = 0xE0;//SSRC: 111 auto convert
    AD1CON1SET= 0x4;//auto sampling after conversion
    //Select voltage reference
    AD1CON2 = 0;//VCFG: 000
    //Select Scan/regular mode
    //AD1CON2SET = 0x400;//CSCNA: 1
    //AD1CON2SET = 0x0;//do not scan
    //Select number of conversions per interrupt
    //SMPI = 0
    //Select buffer fill mode
    //BUFM = 0
    //Select MUX
    //ALTS = 0
    //Select ADC clock source
    AD1CON3 = 0;//ADRC = 0
    //Select acquisition time count
    AD1CON3SET = 0x1913;//SAMC = 1
    //Select ADC clock prescaler
    //ADCS = 0
    //Turn on ADC module
    AD1CON1SET = 0x8000;//ON = 1
    ADC_interrupt_config();
    //Start Sampling Sequence
    AD1CON1SET = 0x0002;//ASAM = 1;
    AD1CON1bits.ON=1;
}

void ADC_enable(void){
    AD1CON1SET = 0x0002;//ASAM = 1;
    AD1CON1bits.ON=1;
}

/* Change notice related functions */
void CN_config(void){
    asm (”di”); //disable all interrupts
    /* Configure CN module */
    CNCONbits.ON = 1; //Enable CN module
    CNEN = 0x00010000; //RD7 corresponds to CN16
    // Read port to set reference
    //readD = PORTDbits.RD6;
    // Configure CN interrupt
    IPC6SET = 0x1C0000; //Set priority level as 5 and subpriority level 0
    IFS1CLR = 0x0001; //Clear interrupt flag
    IEC1SET = 0x0001; //Enable CN interrupts
    asm (”ei”); //enable all interrupts
}

/* Delay generation related functions */
void GenUsec(void) {
    T3CONSET=0x8000;
    TMR3=0;
    while(T3CONbits.ON){
    }
}
void GenMsec(void) {
    int i;
    for (i=0; i<1000; i++) {
        GenUsec();
    }
}
void DelayMsec(uchar num) {
    uchar i;
    for (i=0; i<num; i++) {
        GenMsec();
    }
}
void DelayUsec(uchar num) {
    uchar i;
    for (i=0; i<num; i++) {
        GenUsec();
    }
}

/* PWM related functions */
void Timer2_interrupt_config(void){
    asm(”di”);
    INTCONSET=0x1000;
    IPC2SET = 0x00000016;//Interrupt level 6, sub level 2
    IFS0SET=0x00000100;//Clear interrupt flag
    IEC0SET=0x00000100;//Enable Timer2
    asm(”ei”);
}

void PWM_config(void){
    OC1CON = 0x0000; //Turn off the OC1 when performing the setup
    OC1R = 1; //Initialize primary compare register
    OC1RS = 1; //Initialize secondary compare register
    OC1CON = 0x0006; //Configure for PWM mode without Fault pin enabled
    //Timer2_interrupt_config();
    T2CONSET = 0x8000; //start the timer2
    OC1CONSET = 0x8000; //start OC1
}

void PWM_change(int a){
    if(a < 400){
        OC1RS = a*2;
    }
    else{
        OC1RS = 800;
    }
}

float Get_maxan(){
    //get the maximum of all sensor units
    int max = 0;
    int i;
    for (i = 0; i < 16; i++){
        if (val[i] > max){
            max = val[i];
        }
    }
    return max;
}

/* ISRs */
#pragma interrupt Timer2_ISR ipl5 vector 8
void Timer2_ISR(void){
    T2CONCLR=0x8000;
    TMR2=0;
    IFS0bits.T2IF=0; //CLR=0x100;
    T2CONSET=0x8000;
}

```

```
#pragma interrupt CN_ISR ipl7 vector 26
void CN_ISR(void){
    IEC1CLR = 0x0001; //disable interrupt
    enable=1;
    int readD = PORTDbits.RD7; //clear mismatch conditions
    //DelayMsec(200);
    IFS1CLR = 0x001; //clear interrupt flag
    int j;
    for(j = 0; j < 64; j++){
        send_rgb(0,0,0);
    }
    DelayMsec(30);
    PORTDCLR = 0x2;
    PWM_change(0);
}

#pragma interrupt your_Timer_ISR ipl6 vector 12
void your_Timer_ISR(void) {
    T3CONCLR=0x8000; //stop timer
    TMR3=0;
    IFS0CLR=0x1000; //clear interrupt flag
}

int main(void){
    MCU_init();
    ADC_config();
    ADC_interrupt_config();
    PWM_config();
    CN_config();
    while (1){
        while(!enable){
            //while the button is not pushed
            count = 0; //count for 16 sensor units
            ADC_enable();
            PORTE = count; //output the sensor number to extended module
            while(count!=16){
                while(!IFS1bits.AD1IF);
                int value;
                value = ADC1BUF0; //read to clear buffer
                val[count] = value;
                count ++;
                PORTECLR = 0xf;
                PORTESET = count;
                IFS1bits.AD1IF=0;
                AD1CON1SET = 0x0002;//ASAM = 1;
                AD1CON1bits.ON=1;
            }
            int i = 0;
            for (i = 0; i < 16; i++){
                if(val[i] > 70){
                    //70 is the thresthold value to determine whether a sensor unit is pressed
                    whetheran[i] = 1;
                }else{
                    whetheran[i] = 0;
                }
            }
            int z = 0;
            int j = 0;
            howmany = 0;
            for (j = 0; j < 16; j++){
                //get how many sensor units are pressed and the location for the sensor pressed
                howmany = howmany + whetheran[j];
                if(whetheran[j] == 1){
                    wherean[z] = j;
                    z++;
                }
            }
            if(howmany == 0){
                MODE=0; //start the next sample
            }else if(howmany == 1){
                MODE = 1;
                INDEX[0] = wherean[0];
            }else if(howmany == 2){
                MODE = 2;
                INDEX[0] = wherean[0];
                INDEX[1] = wherean[1];
            }else if(howmany == 3){
                MODE = 3;
            }else{
                PWM_change(Get_maxan());
                continue;
            }
            //adjust PWM to indicate the pressure by a separate LED
            mode_display(MODE); //display pattern on LED board
            PWM_change(Get_maxan());
            int k;
            for (k = 0; k < 16; k++){
                //clear the array that stores pressure values
                val[k] = 0;
            }
            MODE = 0;
            PORTDCLR = 0x2;
        }
        //if button is pushed
        PORTDCLR = 0x2;
        PWM_change(0);
        DelayMsec(3);
        int j;
        for(j = 0; j < 64; j++){
            send_rgb(0,0,0);
        }
        DelayMsec(3);
        break;
    }
    int j;
    for(j = 0; j < 64; j++){
            send_rgb(0,0,0);
        }
    DelayMsec(3);
    PWM_change(0);
    PORTDCLR = 0x2;
}
```