Assignment KD5065 (C-Programming) 2017/2018 Report

Module Title: C-Programming and Digital System

Tutor: Ian Elliot

Name:

Student ID:

Electrical Power Control System Design

Problem specification:

In this module, we are using the hardware platform PIC16F627 which is a flash based 8-bit CMOS microcontroller [1] to implement some applications written in C. The software platform which we are using for debugging and simulation is SourceBoost.

In this assignment we are going to implement a power control system. The system has two input sensors S1, S2 to monitor different load points in the consumer network. The measured data will be used to control the three power generators G0, G1, G2 in the system. In detail: G0 is main power generator which is always on, G1 is on when S1 over 60%, G2 is on when S2 over 70%. In addition, if both S1 and S2 are over 90% and last over 10 seconds, all the three power generators will be switched off and warning LED is on.

To design this system [2], we can refer to what we have learned recently. By drawing a state diagram of the system will make this problem clearer. We should define several states of the system and find the transfer conditions between the states. Firstly, we'd like to come up with the system design.

System Block Diagram:

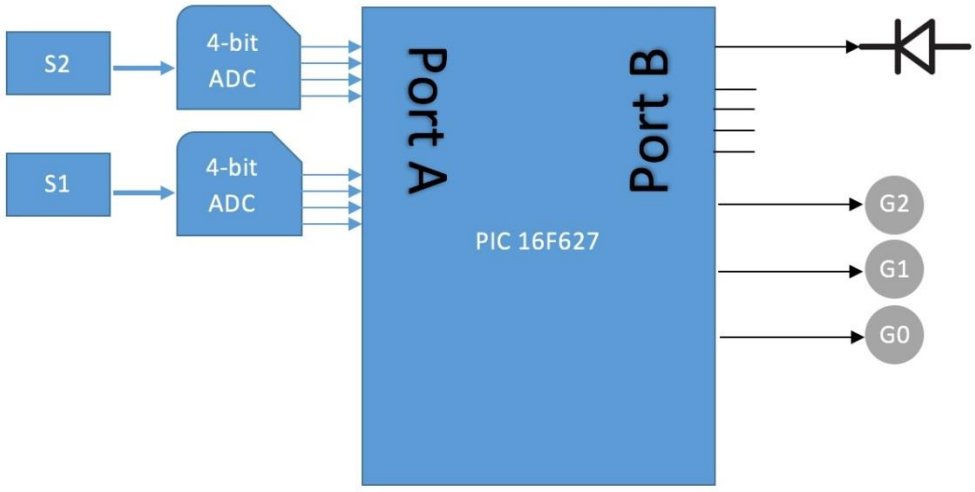

Figure 1 System Block Diagram

The design of the system is shown in the above. we implement C program in the PIC16F627 to control the system. The power control system uses port A as input, port B as output. As far as we have known, both port A and port B have 8 bits.

According to our requirements of the system design specification, we have 2 input sensors each provides 4-bit information for the power consumption shown in percentage. The data is used to control 3 power generators and 1 warning LED. So we come up with the input and output ports design as follows: We set port A as the input port, bit 7 to bit 4 represent the sensor S2's data, bit 3 to bit 0 represent the sensor S1's data. Port b is configured as output port, bit 7 is representing LED, bit 2 to bit 0 represent the G2, G1, G0 respectively. All the bits are high effective, which means LED, G2, G1, G0 are turned on only when the bit is set to high (1).

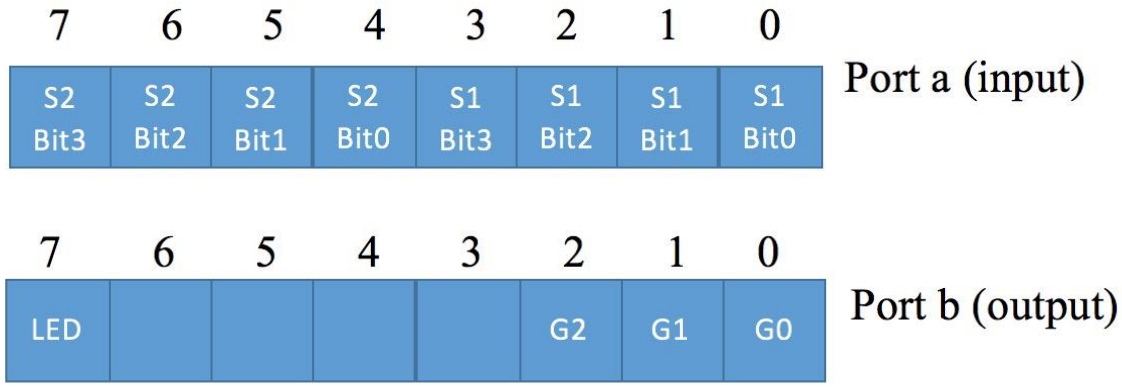

Figure 2 I/O port definition diagram

The 4-bit sensor's value represents the percentage of power consumption by the system.

4-bit value ranges from 0000 to 1111 which in decimal is 0 to 15. We are going to map the 4-bit value from 0-15 to 0%-100%. We want to distribute it averagely. Then 100% is equal to 1111, 0% is equal to 0000.

60% is equal to 15*60% =9; 70% is equal to 15*70%=10.5; 90% is equal to 15*90%=14.5.

Condition of sensor S1's value more than 60% is equal to S1>1001. S2's value more than 70% is equal to S2>1010. More than 90% is equal to S1 or S2 > 1110.

State diagram:

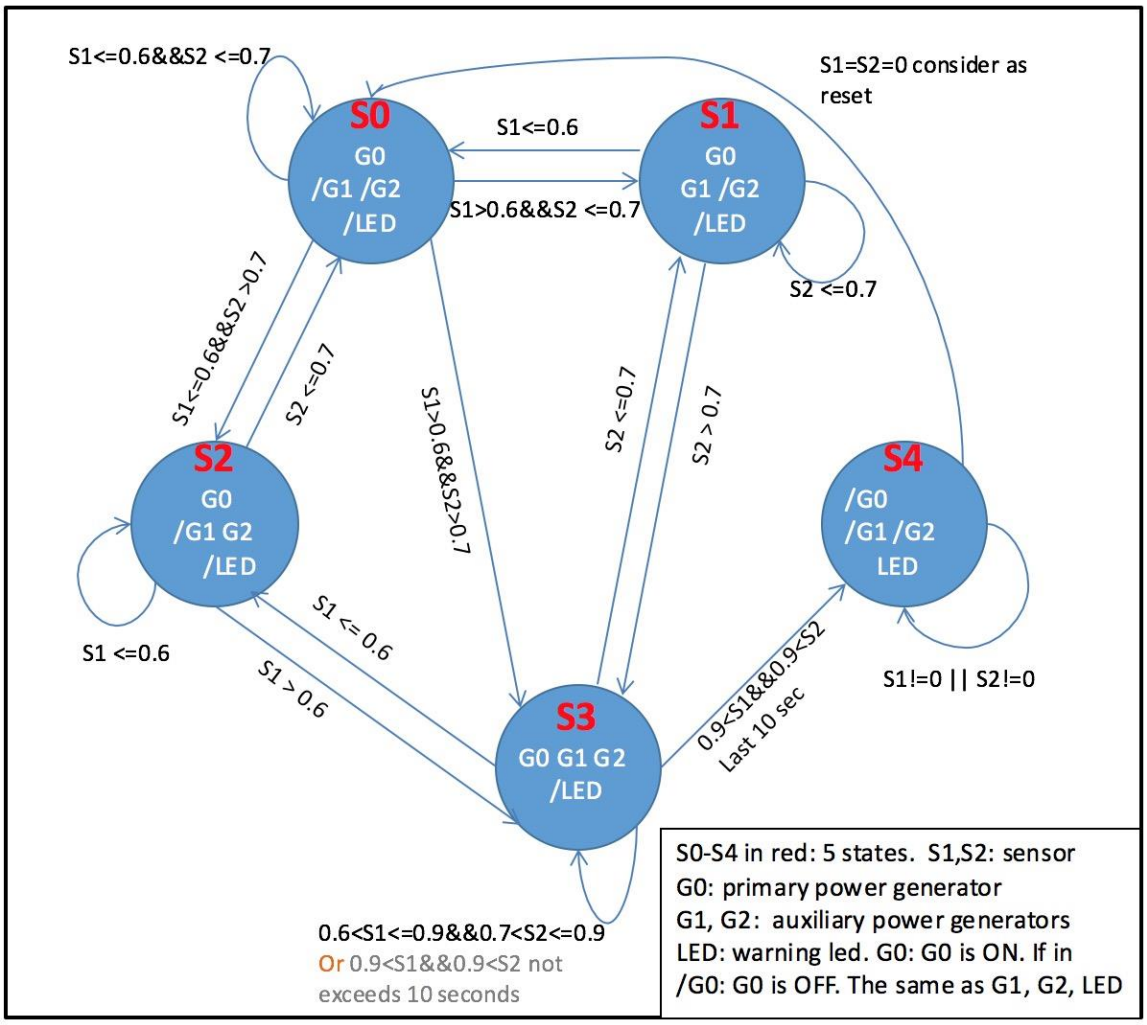

Figure 3 State Diagram

The above diagram shows the states and states transfer conditions of the system.
States briefing:

- S0: The initial state, since G0 is always on, in the initial state we turn on G0 by default and G1, G2, LED are off because sensor S1, S2 have not started to work yet.
- S1: The state when G0 is on, G1 is on, G2 is off, LED is off.
- S2: The state when G0 is on, G2 is on, G1 is off, LED is off.
- S3: The state when G0 is on, G1 is on, G2 is on, LED is off.
- S4: The warning state which shows overloading. G0, G1, G2 are off. Warning LED is on.

Transfer conditions:

- S0: Can transfer to S1, when sensor S1's value is over 60% and sensor S2's value doesn't exceed 70%. S0 can transfer to S2, when sensor S1's value doesn't exceed 60% and sensor S2's is more than 70%. S0 can transfer to S3, when sensor S1's value is over 60% and sensor S2's value is more than 70%. Stay in S0 if sensor S1's value is no more than 60% and sensor S2's value doesn't exceed 70%.
- S1: Can transfer to S0, when sensor S1's value is no more than 60%. S1 can transfer to S3, when sensor S2's value is over 70%. Stay in S1 if sensor S2's value is not over 70%.
- S2: Can transfer to S0, when sensor S2's value is no more than 70%. S2 can transfer to S3, when sensor S1's value is over 60%. Stay in S2 if sensor S1's value is not over 60%.
- S3: Can transfer to S1, when sensor S2's value is no more than 70%. S3 can transfer to S2, when sensor S1's value is no more than 60%. S3 can transfer to S4, when sensor S1's value is more than 90% and sensor S2's value is over 90% and after 10 seconds still over 90%. Stay in S3 if sensor S1's value is not over 90% but more than 60% and sensor S2's

value is not over 90% but more than 70%. Another way to stay in S3 when sensor S1's value is more than 90% and sensor S2's value is over 90% but after 10 seconds not over 90%.

- S4: Can transfer to S0, when sensor S1's value is equal to sensor S2's value equals to 0. Which we consider as a reset of the system to the initial state. Stay in S4 if sensor S1's value is not equal to 0 or sensor S2's value is not equal to 0.

Design details:

According to our design, we achieved states transfer function across all the five states, S1 to S2 needs to go through S1 to S0 to S0 to S2, or S1 to S3 to S2 which makes sense, because once the sensor S2's value is over 70% state S1 will transfer directly into S3, then to S2 if sensor S1's value is not over 60%, this is same apply to S1 to S0 to S2.

We carefully designed these state graph fully taking the real application needs into consideration. For example, when we were in warning state, we can jump to initial state by reset the power system. The reset condition is when sensor S1's value and sensor S2's value is equal to 0 which means the system has no huge consumption now.

Codes with Comments (This piece of codes with **green color comments** is for demo, we changed the text setting. You are **not able** to run it directly. The appendix code can be copied and run):

Demo code. (If you want to run the code please copy the appendix code)

```
#include <system.h>
void state0(void);//initial state G0 on. G1, G2 OFF
void state1(void);//G0, G1 on. G2 OFF
void state2(void);//G0, G2 on. G1 OFF
void state3(void);//G0, G1, G2 on.
void state4(void);//Warning state, G0, G1, G2 OFF
void config(void);//PIC configuration
//global variables declaration
unsigned char state, nxtstate, inbyte, sensor_s1 , sensor_s2;
```

```
int percent_60 = 9;//4 bits: 0xF in decimal is 15. So
0.6*15=9
int percent_70 = 10;//4 bits: 0xF in decimal is 15. So
0.7*15=10.5
int percent_90 = 14;//4 bits: 0xF in decimal is 15. So
0.9*15=14.5

int main(void) {
    config();
    nxtstate=0;
    state=0;

    do{
        switch(state) {
            case 0: {state0(); state=nxtstate;  break;}
            case 1: {state1(); state=nxtstate;  break;}
            case 2: {state2(); state=nxtstate;  break;}
            case 3: {state3(); state=nxtstate;  break;}
            case 4: {state4(); state=nxtstate;  break;}
        }//end of switch. state machine transfer functions.
    }while(1);//end of do while loop
    return(0);
}

//////////////////PIC CONFIGURATION////////////////////////

void config(void) {
    cmcon = 0x07; //this needed to make the porta see digital
I/O
    trisa = 0xff; //set all the pins of port a as input
direction
    trisb = 0x00; //set all the pins of port b as output
direction
    portb = 0; //initial the value of port b to be 0
}

//////////////////DEFINING THE STATES////////////////////////

/*G0 on. G1, G2 LED OFF (Initial state) */
void state0(void){
    inbyte = porta;
    portb = 0x01;//G0 on. G1, G2 OFF
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as
sensor s2 data
    if((sensor_s1 <= percent_60) && (sensor_s2 <=
percent_70)){
        nxtstate = 0; // s1<=60% and s2<=70% stay in state 0
```

```
    }
    else if ((sensor_s1 > percent_60) && (sensor_s2 <=
percent_70)){
        nxtstate = 1; // s1>60% and s2<=70% go to state 1
    }
    else if ((sensor_s1 <= percent_60) && (sensor_s2 >
percent_70)){
        nxtstate = 2; // s1<=60% and s2>70% go to state 2
    }
    else if ((sensor_s1 > percent_60) && (sensor_s2 >
percent_70)){
        nxtstate = 3; // s1>60% and s2>70% go to state 3
    }
}//end of state0.

/*G0, G1 on. G2 LED OFF*/
void state1(void){
    inbyte = porta;
    portb = 0x03; //G0, G1 on. G2 OFF
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as
sensor s2 data
    if(sensor_s1 <= percent_60){
        nxtstate = 0; // s1<=60% go to state 0 directly
        return;
    }
// s1>60%
    if(sensor_s2 <= percent_70){
        nxtstate = 1; // s1>60% and s2<=70% stay in state 1
    }
    else {
        nxtstate = 3; // s1>60% and s2>70% go to state 3
    }
}//end of state1.

/*G0, G2 on. G1 LED OFF*/
void state2(void){
    inbyte = porta;
    portb = 0x05;//G0, G2 on. G1 OFF
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as
sensor s2 data
    if(sensor_s2 <= percent_70){
        nxtstate = 0; // s2<=70% go to state 0 directly
        return;
    }
// s2>70%
    if(sensor_s1 <= percent_60){
```

```
        nxtstate = 2; // s1<=60% s2>70% stay in state 2
    }
    else {
        nxtstate = 3; // s1>60% s2>70% go to state 3
    }
}//end of state2.

/*G0, G1, G2 on. LED OFF*/
void state3(void){
    int count = 0;
    inbyte = porta;
    portb = 0x07;//G0, G1, G2 ON
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as
sensor s2 data
    if(sensor_s1 <= percent_60) {
        nxtstate = 1; // s1<=60% go to state 1
    }
    else if (sensor_s2 <= percent_70) {
        nxtstate = 2; // s1>60% and s2<=70% go to state 2
    }
    else if ((sensor_s1 <= percent_90)&&(sensor_s2 <=
percent_90)){
        nxtstate = 3; // 60%<s1<=90% and 70%<s2<=90% stay in
state 3
    }
    else if ((sensor_s1 > percent_90)&&(sensor_s2 >
percent_90)){
        delay_s(10);//after 10 seconds check again. External
function
        if((sensor_s1 > percent_90)&&(sensor_s2 >
percent_90)){
            nxtstate = 4; // s1>90% and s2>90% after 10
seconds go to state 4
        }
        else{
            nxtstate = 3; // s1<=90% or s2<=90% after 10
seconds stay in state 4
        }
    }
}//end of state3.

/*G0, G1, G2 OFF. LED ON (overloading, warning state)*/
void state4(void){
    inbyte = porta;
    portb = 0x80;//G0, G1, G2 OFF, warning LED bit(7) high
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as
```

```
sensor s2 data
    if((sensor_s1 == 0)&&(sensor_s2 == 0)) {
        nxtstate = 0; // s1=s2=0 go to state 0
    }
    else {
        nxtstate = 4; // s1!=0 or s2!=0 stay in state 4
    }
}//end of state4.
```

Simulation results:

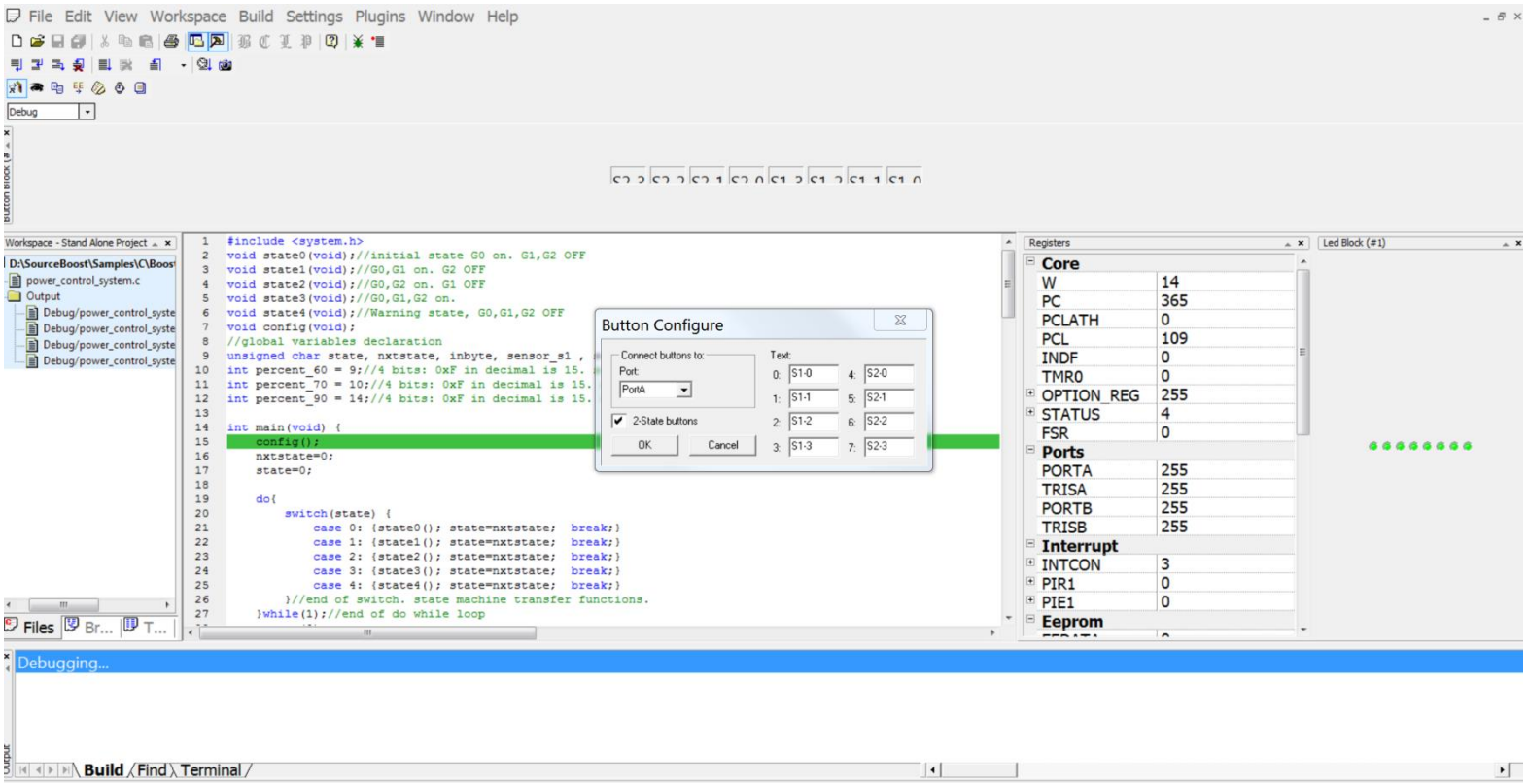

Figure 4 Button Configuration

Firstly, we configure the buttons as 2-state buttons for simulation. It is because we will press multiple buttons simultaneously. (S1>60%: we set sensor S1 to 1011 by pressing bit 3, 1, 0 of port A at the same time.)

LED BLOCK is connect to PORTB.

In state 0: initial state

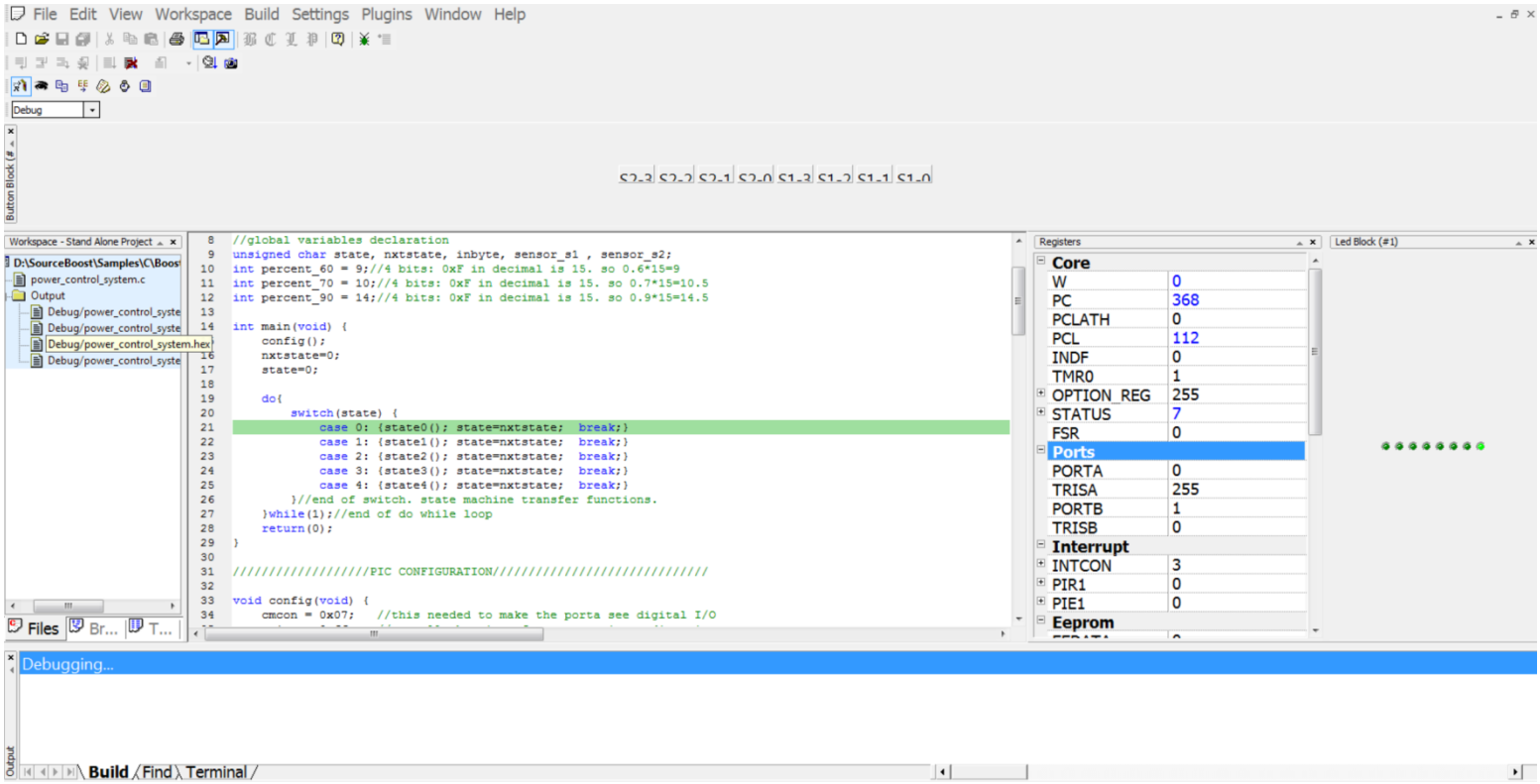

Figure 5 state 0 (initial state)

In the debug mode, we don't press any button and do step over (F10), at this time input value is 0, as PORTA suggests. Only G0 will on, PORTB value is 1 which is matched with the LED result.

Note that TRISA is 255 equals 0xFF which means we configure all the pins of PORTA to the input direction. TRISB is 0 equals 0x0 which means we configure all the pins of PORTB to the output direction. Which is correct comparing with the codes of configuration part.

Right now the program is looped in the state 0, waiting for us to change the input setting.

At this time if we change the input value, but both sensor s1 and sensor s2 are not over the threshold, it will also stay in state 0. Only when we change the sensor s1 value over 1001 (9) or sensor s2 value over 1010 (10), the state will transfer.

In state 1: G1 is on

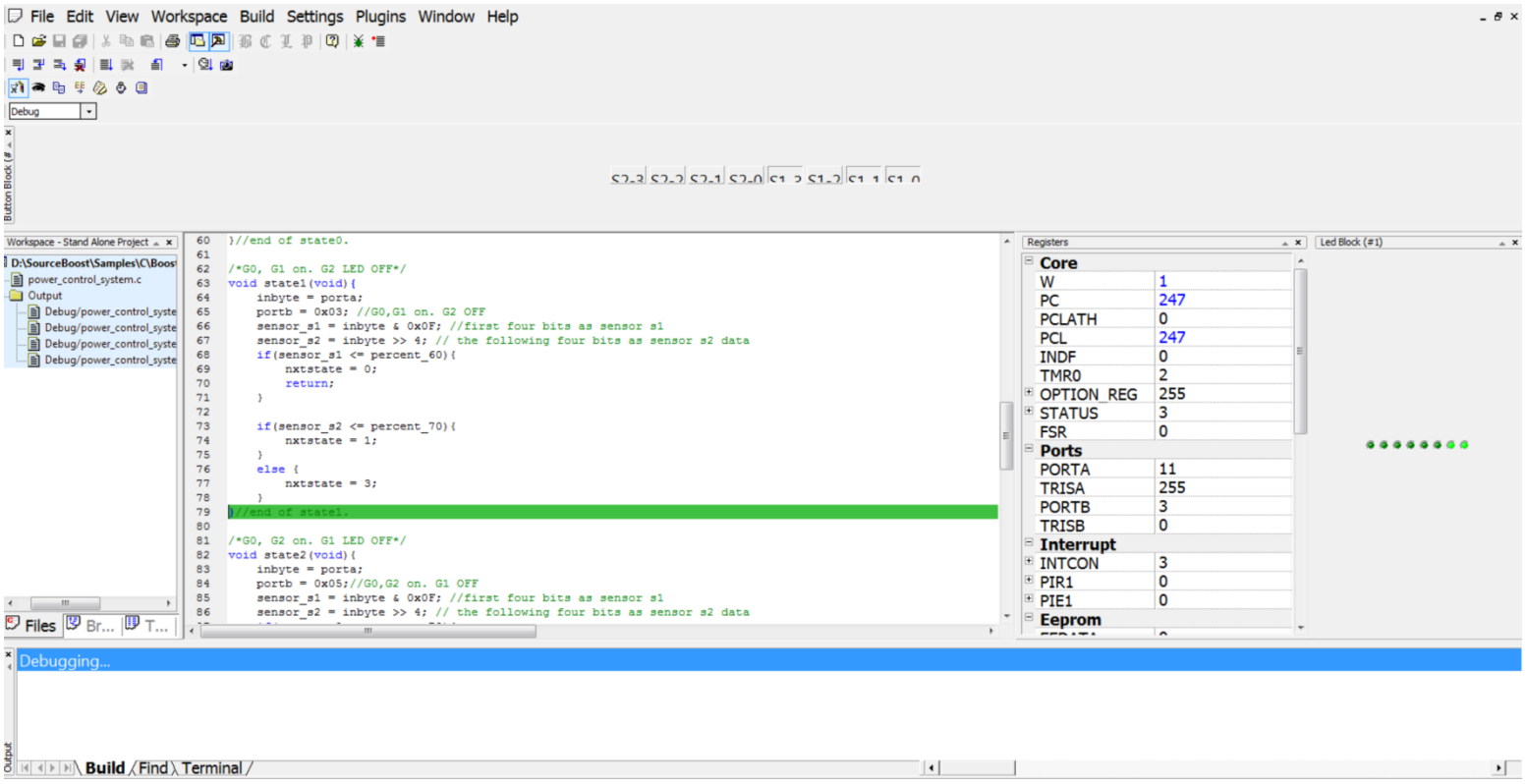

Figure 6 state 1 (only G0, G1 are on)

As the screenshot shows our debug process, we press the button to configure the sensor s1's input to 1011 which is over 60%, then we step over, the debug point will move to state 1, we press F11 (step into) to debug into this function, we will get the output as PORTB shows, it is configured as 0x03, the G1, G0 are on then.

In state 1, if we don't change the sensor s1's value below 60% it will not go to the state S0 (turn off G1 for power saving purpose). If we change the sensor s2's value over 70% it will go to state 3. This time we want to evaluate the process of state 1 to state 0 to state 2, we change the input as follows.

In state 2: G2 is on

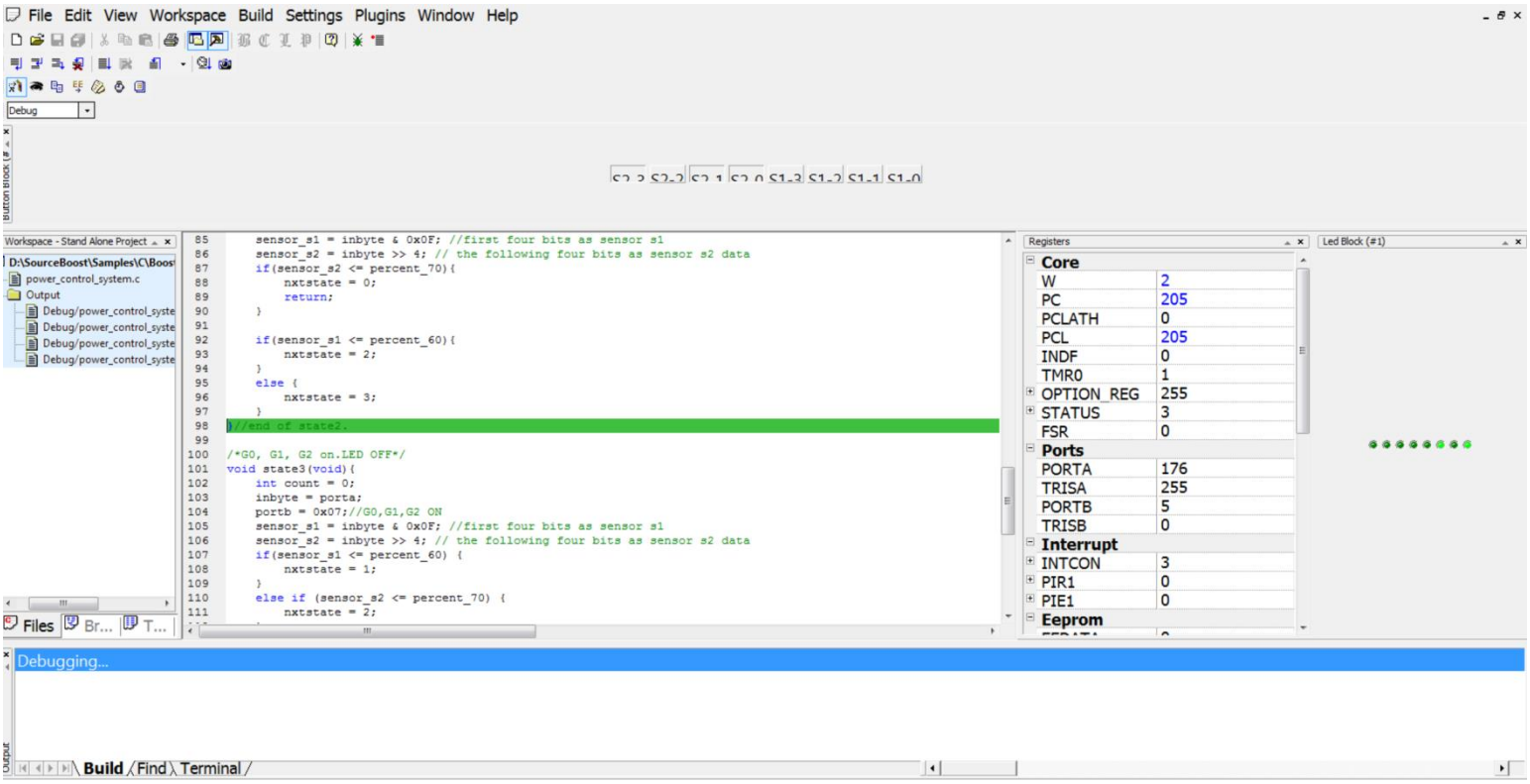

Figure 7 state 2 (only G0, G2 are on)

As the screenshot shows our debug process, we press the button to configure the sensor s2 input to 1011 which is over 70%, then we step over, the debug point will move to state 2, we press F11 (step into) to debug into this function, we will get the output as PORTB shows, it is configured as 0x05, the G2, G0 are on then.

In state 2, if we don't change the sensor s2's value below 70% it will not go to the state S0 (turn off G2 for power saving purpose). If we change the sensor s1's value over 60% it will go to state 3.

This moment we want to evaluate the state s3, so we generate the following settings for sensor s1.

In state 3: G1 and G2 are on

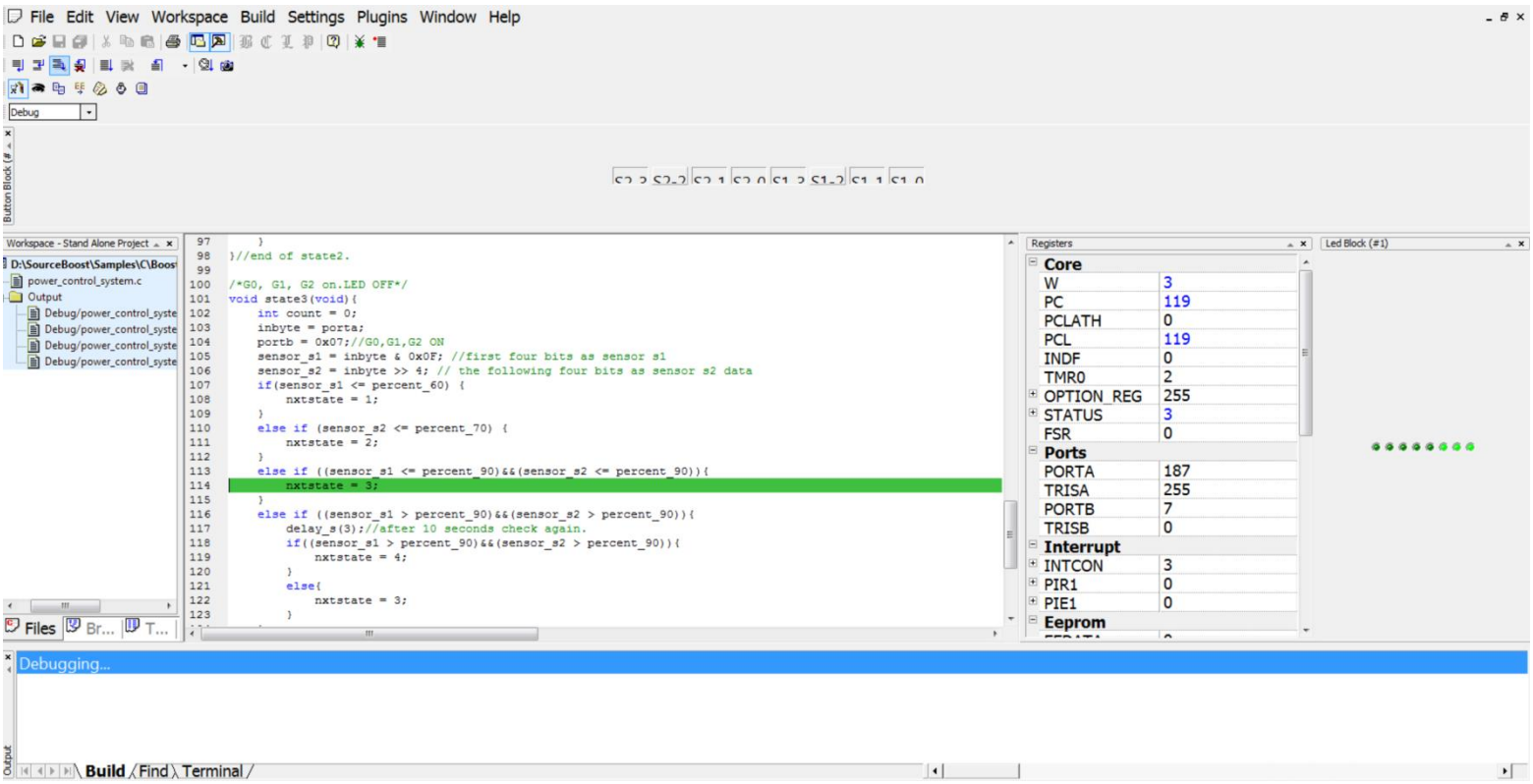

Figure 8 state 3 (G0, G1, G2 are on)

We are in state 3 now, as the screenshot shows our debug process, we press the button to configure the sensor s2's input to 1011 which is over 70%, and sensor s1's input to 1011 which is over 60%, then we step over, the debug point will move to state 3, we press F11 (step into) to debug into this function, we will get the output as PORTB shows, it is configured as 0x07, the G1, G2, G0 are on then.

In state 3, if we change the sensor s2's value below 70% it will go to the state s1 for power saving purpose. If we change the sensor s1's value below 60% it will go to state s2 for power saving purpose. If we change both sensor s1's and sensor s2's value over 90% and after a period of 10 seconds the values are still over 90%, it will go to state 4, which is the alarming state, all the power generators will be turned off, and warning LED is on. Else it will stay in state3.

This moment we want to evaluate the state s4, so we generate the following settings for sensor s1 and sensor s2.

In state 4: G0, G1, G2 are off and LED is on

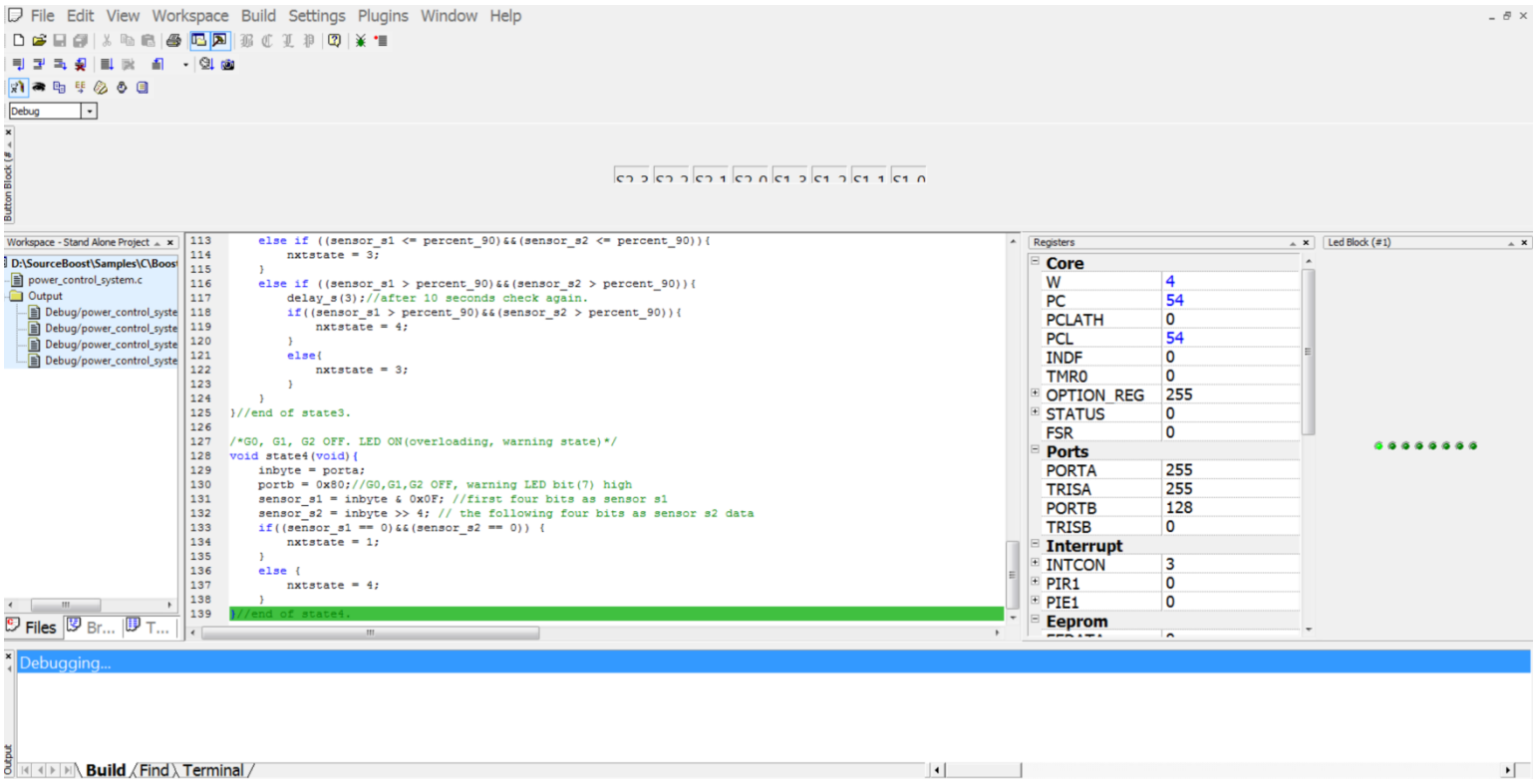

Figure 9 state 4 (Warning state, G0, G1, G2 are off LED is on)

As the screenshot shows our debug process, we press the button to configure the sensor s2 input to 1111 which is over 90%, and sensor s1 input to 1111 which is also over 90%, then we step over, the debug point will move to state 4, we press F11 (step into) to debug into this function, we will get the output as PORTB shows, it is configured as 0x80 (128), the G1, G2, G0 are off warning LED is on.

In state 4, if we change the sensor s1's value and sensor s2's value both as 0, it will go to the state S0 for reset purpose.

At this point we have simulated all the states of our power control system.

Conclusion:

Through this assignment, we have learned how to build a c model to simulate a state transfer system. We have much deeper understanding about how we can use microcontroller to solve real world issues.

Reference:

[1] Microchip, PIC16F62X Data Sheet

[2] Balaji, M., et al. "Adaptable and Reliable Industrial Security System using PIC Controller." *May-2017, International Journal for Innovative Research in Science & Technology* 3 (2017): 56-60.

Appendix

Code for power system simulation

```
#include <system.h>
void state0(void);//initial state G0 on. G1,G2 OFF
void state1(void);//G0,G1 on. G2 OFF
void state2(void);//G0,G2 on. G1 OFF
void state3(void);//G0,G1,G2 on.
void state4(void);//Warning state, G0,G1,G2 OFF
void config(void);
//global variables declaration
unsigned char state, nxtstate, inbyte, sensor_s1 , sensor_s2;
int percent_60 = 9;//4 bits: 0xF in decimal is 15. so 0.6*15=9
int percent_70 = 10;//4 bits: 0xF in decimal is 15. so 0.7*15=10.5
int percent_90 = 14;//4 bits: 0xF in decimal is 15. so 0.9*15=14.5

int main(void) {
    config();
    nxtstate=0;
    state=0;

    do{
        switch(state) {
            case 0: {state0(); state=nxtstate;  break;}
            case 1: {state1(); state=nxtstate;  break;}
            case 2: {state2(); state=nxtstate;  break;}
            case 3: {state3(); state=nxtstate;  break;}
            case 4: {state4(); state=nxtstate;  break;}
        }//end of switch. state machine transfer functions.
    }while(1);//end of do while loop
    return(0);
}

//////////////////PIC CONFIGURATION///////////////////////////////

void config(void) {
    cmcon = 0x07;   //this needed to make the porta see digital I/O
    trisa = 0xff;      //set all the pins of port a as input direction
    trisb = 0x00;   //set all the pins of port b as output direction
    portb = 0;      //initial the value of port b to be 0
}

//////////////////DEFINING THE STATES///////////////////////////////

/*G0 on. G1, G2 LED OFF(Initial state) */
void state0(void){
    inbyte = porta;
    portb = 0x01;//G0 on. G1,G2 OFF
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as sensor s2 data
    if((sensor_s1 <= percent_60) && (sensor_s2 <= percent_70)){
        nxtstate = 0;
    }
    else if ((sensor_s1 > percent_60) && (sensor_s2 <= percent_70)){
```

```
        nxtstate = 1;
    }
    else if ((sensor_s1 <= percent_60) && (sensor_s2 > percent_70)){
        nxtstate = 2;
    }
    else if ((sensor_s1 > percent_60) && (sensor_s2 > percent_70)){
        nxtstate = 3;
    }
}//end of state0.

/*G0, G1 on. G2 LED OFF*/
void state1(void){
    inbyte = porta;
    portb = 0x03; //G0,G1 on. G2 OFF
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as sensor s2 data
    if(sensor_s1 <= percent_60){
        nxtstate = 0;
        return;
    }

    if(sensor_s2 <= percent_70){
        nxtstate = 1;
    }
    else {
        nxtstate = 3;
    }
}//end of state1.

/*G0, G2 on. G1 LED OFF*/
void state2(void){
    inbyte = porta;
    portb = 0x05;//G0,G2 on. G1 OFF
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as sensor s2 data
    if(sensor_s2 <= percent_70){
        nxtstate = 0;
        return;
    }

    if(sensor_s1 <= percent_60){
        nxtstate = 2;
    }
    else {
        nxtstate = 3;
    }
}//end of state2.

/*G0, G1, G2 on.LED OFF*/
void state3(void){
    int count = 0;
    inbyte = porta;
    portb = 0x07;//G0,G1,G2 ON
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as sensor s2 data
    if(sensor_s1 <= percent_60) {
        nxtstate = 1;
    }
    else if (sensor_s2 <= percent_70) {
        nxtstate = 2;
    }
    else if ((sensor_s1 <= percent_90)&&(sensor_s2 <= percent_90)){
        nxtstate = 3;
    }
    else if ((sensor_s1 > percent_90)&&(sensor_s2 > percent_90)){
        delay_s(3);//after 10 seconds check again.
        if((sensor_s1 > percent_90)&&(sensor_s2 > percent_90)){
            nxtstate = 4;
        }
        else{
            nxtstate = 3;
        }
```

```
    }
}//end of state3.
/*G0, G1, G2 OFF. LED ON(overloading, warning state)*/
void state4(void){
    inbyte = porta;
    portb = 0x80;//G0,G1,G2 OFF, warning LED bit(7) high
    sensor_s1 = inbyte & 0x0F; //first four bits as sensor s1
    sensor_s2 = inbyte >> 4; // the following four bits as sensor s2 data
    if((sensor_s1 == 0)&&(sensor_s2 == 0)) {
        nxtstate = 0;
    }
    else {
        nxtstate = 4;
    }
}//end of state4.
```