Yu Yin Guan

Email: yguan5@ucsc.edu

Section: Wednesday 10am-12pm

Lab Project Report

Introduction:

In the Lab project, our objective is the create a working connect four game with the knowledge gain in class. This game can play with other players wirelessly through a xbee where the players are able to select the channels they desire. Player can view the dots displayed on the LED display matrix without a problem. The game should also be full functional where there is a turn system which allows the two players to take turn. A display of victory should be presented when there is 4 dots connected.

Design:

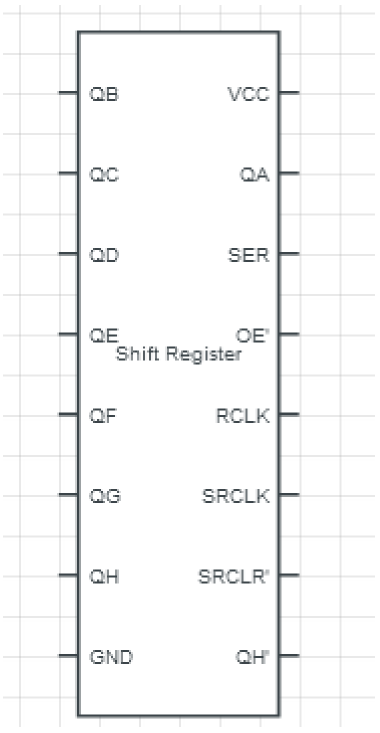
Design Description:

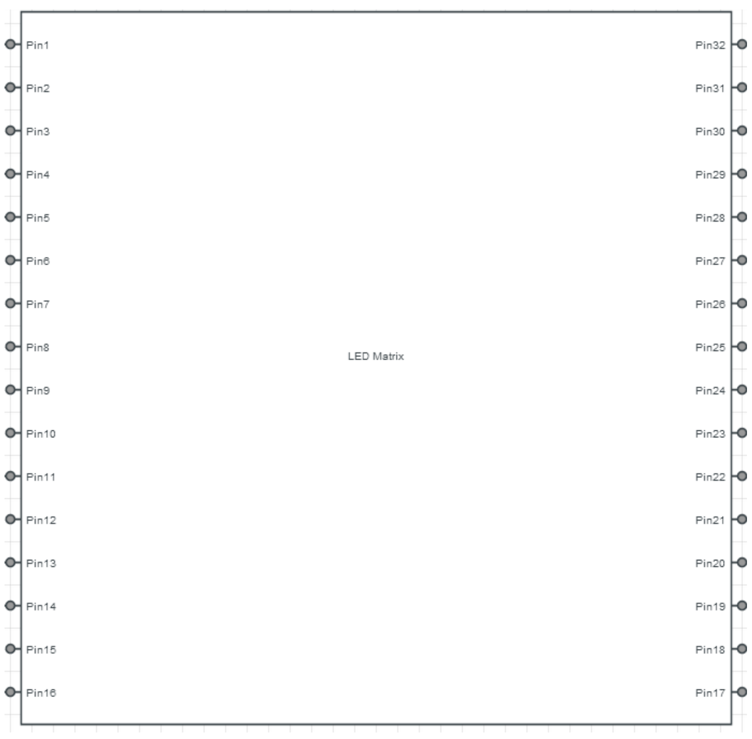
For my design, I created ring counter which counts from row 1 of the LED matrix to row 8 and repeats. Before going onto another row, I would use 2 registers and clock in the data of which LEDs are to be lit on the next row, but doesn't latch it. When the ring counter counts to the next row, I latch the data stored in the registers and start preparing for the data of the next row. Each row will be lit for two milliseconds, thus creating an matrix that refreshes and appear to have the LED light up. The details of design will be discussed on the hardware and software part of the design.

Hardware or How the project is wired:

My design consist of three shift register, sixteen 330 ohm resistors, and an LED Matrix display. The colors for the dots of the connect four game that I choose is Red and Blue where Blue as the color of my dots and Red as the color of my enemies dots.

Parts:

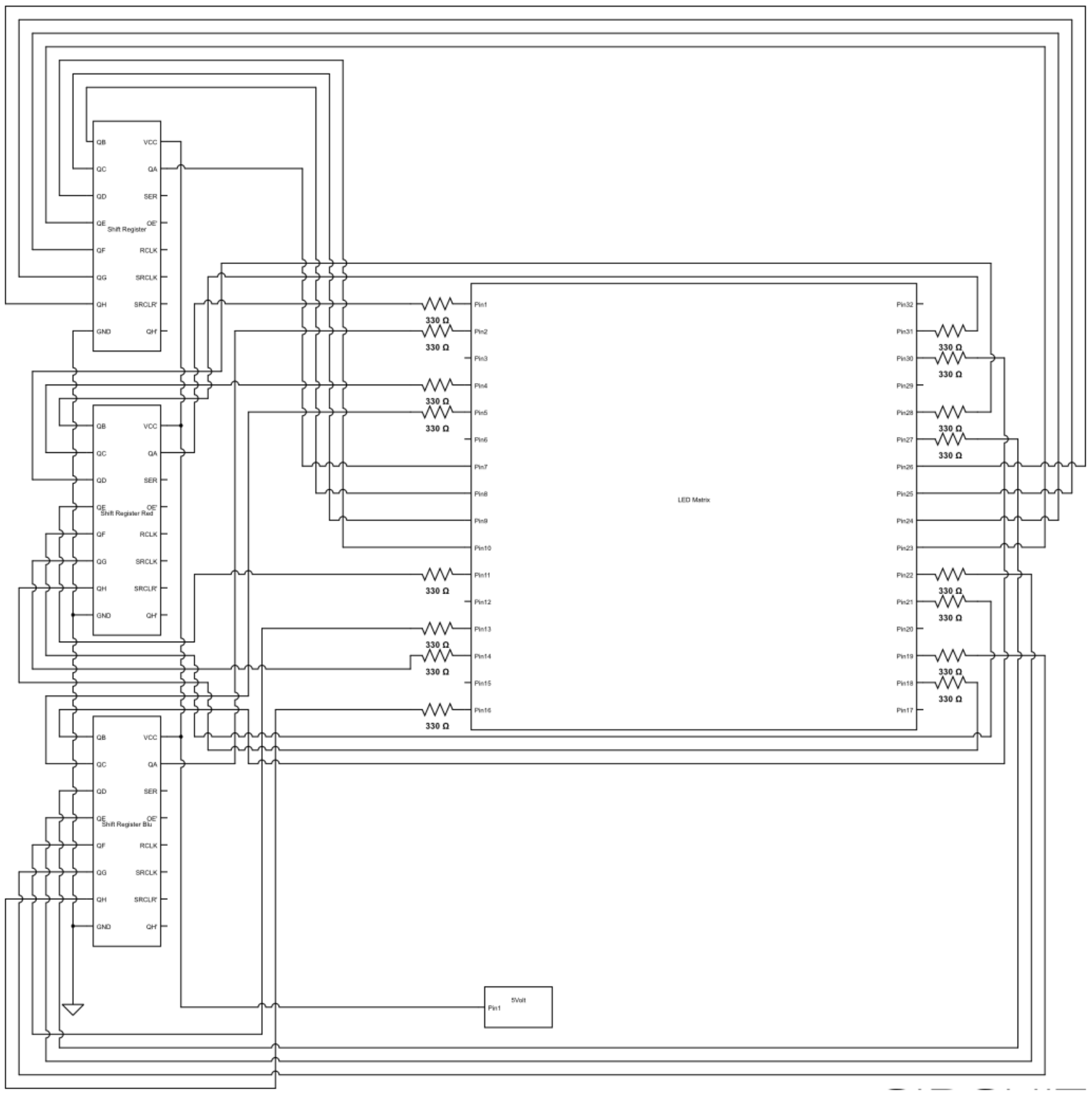




Since we are using Blue and Red LEDs on the Panel, Pins 1, 2, 4, 5, 11, 13, 14, 16, 18, 19, 21, 22, 27, 28, 30, and 31 will all be connected with and 330 resistor. The reason for this is because I wanted an even distribution of currents flowing through the LEDs as they light up and limit the amount of current through the LEDs. The other end of all those resistors will then be connected with two shift registers which I would use to control which LEDs are to be lit. The last shift register is connected with the pins 7 to 10 and 23 to 26, which are the anode of all the LEDs in the matrix. This shift register is used to be an ring counter that counts through from row 1 to row 8, lighting up the LEDs that should be on in the row, and repeats indefinately. The GND of the registers will be grounded and the VCC will be connected to a 5V power. The SER, OE', RCLK, SRCLK, SRCLR' of the register is connected to the microcontroller for control. Notice that I did not mention the use of any transistors in my design. This is because since there will be at most 8 LED lit up at a given time due to a ring counter, the matrix requires significantly less currents. Although the LEDs are not as bright, the LED display is able to light up the whole panel without a problem and it seems the shift register chips can handle this amount of current draw. Since the transistors are recommended but not required, I did not put them in my design as I see them as a redundant piece of design. It should also be mentioned that my PSOC5 board is set to 3.3V.

Final outlook:

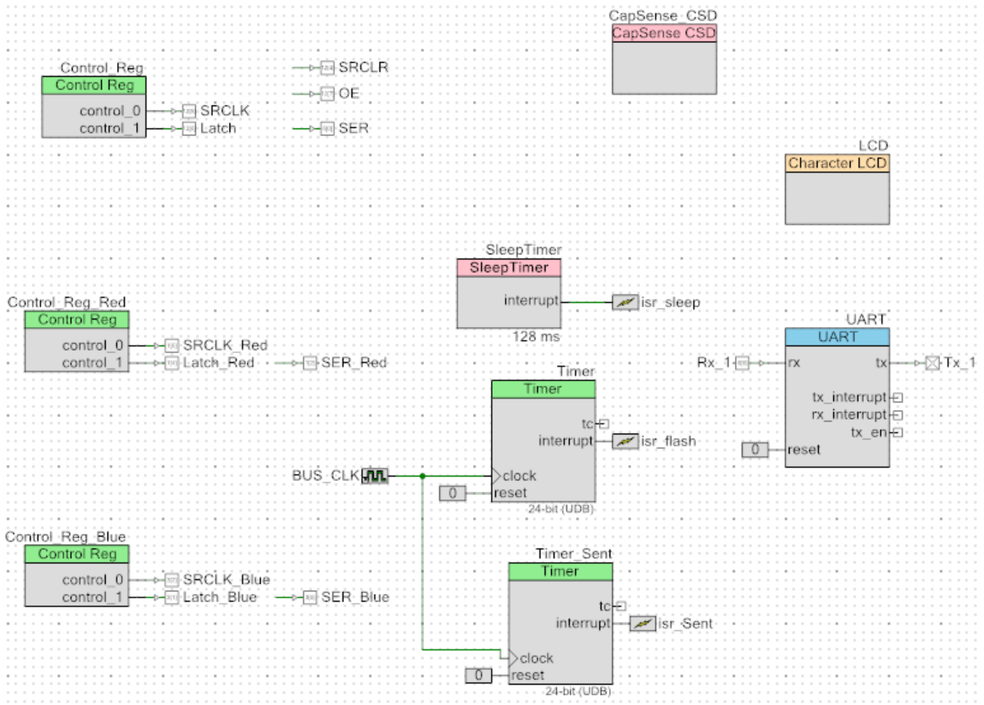
All the information in this is exactly the same as my design. Also, the unconnected part of the shift registers are reserved for the microcontroller except for QH'.



Besides the LEDs, the Xbee also requires some connections. Though I will not be showing it in a form of picture, I will describe how it is connected. There are only 4 pins on the Xbee which I used, the Din and Dout pin, and the VCC and GND pin. The GND is connected to ground while VCC is connected to one of the 3.3V power supply on the board. The Din pin is connected to a pin which is assign to an UART TX, where the Dout pin is connected to a pin which is assign to an UART RX. The Xbee is independent from the LED display matrix since it require not physic connection to it, all it requires are software for command and transition and receive of information.

<u>Software or How it is programmed and its functionalities:</u>

Top Schematic:



As shown on the pictuer, there are three control registers each for using a shift register. I control the shift register by changing values to these registers, so if I want to have SRCLK to clock once, I change it to 0 then 1. This design also allows me to control two clocks at the same time or one at a time depending on the situation. The SER pins are controlled directly by writing to them because I want to work it independently. In my code, the OE and SRCLR are always 0 and 1, the reason why there is only 2 pins for 3 shift registers is because they are all the same value and I connected them to the same pin. The three timer above all gives interrupts within a given time of 500ms. The first timer with isr_sleep, is use for the animation for dropping dots. The 2nd timer with isr_flast, is use to create a flashing animation when a person wins. The last timer with isr_Sent is a timer used to sent a packet every ½ a second. There is also a capsense and LCD for controlling the user control of the homerow and the display of location and error on LCD. The UART is used for the xbee to sent and receive data. All of these parts will be described in detail later.

More Design Discriptions:

The way I implemented my game is that I have a greeting page with a happy face. Then I choose which ever player goes first with the two capsense buttons. After a chose is made, it will either allow the player to move the home row dot with the capslide or wait until there is a signal of the enemy about the move they made. After a move is made, there will be a drop animation of the dot from home row down to their respective spots for both the players. After the drop animation, there is a check for victory for both the players. Then, it will loop forever letting two players take turn making their moves until one of them wins. When one player does win, there will be a flashing animation of those 4 connected dots for 5 second and goes back to the greeting page to start a new game. The whole time when the game is going, there will be data of the set packets sent to the enemy every ½ a second. Also the player will wait for the enemy to make a move before they can do anything. The LCD will put out who's turn it is. Also, all the data of what is to be lit on matrix is stored in a 2D array.

Algorithms:

In my main software, I created a bunch of loops which will exit itself and go into the next loop depending and given a condition. The first loop I have in my main is an infinate loop which will never end, and with it is my main code for this design. Everything else in the main is to set up the LED matrix and the data of it, or the Xbee and other software like UART. As the code get into my main loop, there is a while loop for a homepage for the beginning of the game to choose who goes first.

Algorithm for the first greeting page loop:

```
while(haven't chosen who goes first){
        update senser values

        Set up LEDs to display a happy face

        while(when capsense slider not touched){
             fresh leds using ring counter
        }

        if (button 0 is pressed){
              the player is set to go first
              reset all the array to erase off the happy face
              delay ½ a second for a transition feeling
              exit using break or while loop parameter change
        }
        if (button 1 is pressed){
              the enemy is set to go first
              reset all the array to erase off the happy face
              delay ½ a second for a transition feeling
              exit using break or loop parameter change
        }
}
```

This loop will not exit until one of the buttons are press and depending on which buttons are pressed, it will go into either the 1st player loop or 2nd player loop. Also, the algorithm above is made for capsense where all I need to do is put the CAP_SENSE into the top schematic, set the pins and update what it is reading. After it gets into the 1st player loop, it first hits the chose spot loop.

Algorithm for chosing spot:

```
while(no button is pressed){

          update sensors and button
           print current spot to LCD

           while(sensor is not touched){
                   keep refreshing ring counter
           }

           if (button pressed){
               exit
           }

             save adjust value of position

           if(saved value have no error){
               set the spot
           }
      }
```

This loop will stop only when a button on capsense is pressed, and when it does, it exits the loop with the current spot value that the player selected for the dot to drop. Then after that it hits the dropping animation set up and loop.

Algorithm for dropping animation:

```
checks what spot play selected to drop disc

        for(all the columns of row 8){
            if(selected by player){
                setLocation = selected location in column #
            }
              refreshes ring counter while looping
        }

        Start timer for the ½ isr for dropping to next spot

        while(have not droped to the bottom){
            refreshes ring counter while looping
        }

        stop timer so it does not interrupt when after the dropping animation
```

Algorithm for the Dropping Animation Timer Interrupt:

```
CY_ISR(sleep){

    if(haven't hit row 1 and there still nothing under the dot){
        spot under = color of current spot
        delete the color of current spot
    }else{
        set a value to exit the while loop on the algorithm above
    }
}
```

Every 124 milliseconds, it interrupts and drops the dot one spot lower until it is on top of another dot or at bottom row. Meanwhile when waiting for the ISR for dropping the dot, the ring counter just keeps refreshing the matrix. After this loop, it will set the packet with the location of the dot I just dropped so that the interrupt that happens every ½ a second will sent out the most recent chose of dots. Of course the Xbee must be setted up, which I did on main before the main loop.

Algorithm for setting Xbee:

```
    CyDelay(3000);
    UART_PutString("+++");
    CyDelay(2000);
    UART_PutString("ATAP1\r");
    UART_PutString("ATCH F\r");
    UART_PutString("ATWR\r");
    UART_PutString("ATCN\r");
```

I used these codes to have the UART sent the command of changing to AP mode and changing channel on my XBEE.

Algorithm for the ISR of senting the packet:

```
CY_ISR(st) {

    Put the array of packets through the UART to XBEE;
    Read timer status to get out of interrupt
}
```

After I updated the packet that I am senting, I will check for victory and does a flashing animation of the 4 dots connected.

Algorithm for dealing with victory:

```
      if(the player won, checked with victory function){

                start timer for interrupt flashes

                while(Less than 5 flash){
                    refresh matrix
                }
                stop the interrupt timer
                Delay 1 second for transition
                reset the arrays and datas
                set value to return to the greeting page
            }

            if not won, set value to go to the other player's loop
```

Algorithm for checking victory:

```
    Create variables used

      //horizontal win check
    for(column 1-8 at that selected row){
        if(there is color that is being checked for win){
            increment a win value
        }else{
            put win value to 0;
        }
        if(win value >= 4 after checking the whole row){
            save which values are connected
            set the win and returns a true for winning
        }
    }
      //vertical win check
    for(row 1-8 at that selected row){
        if(there is color that is being checked for win){
            increment a win value
        }else{
            put win value to 0;
        }
        if(win value>=4){
            save which values are connected
            set the win and returns a true for winning
        }
    }

    //diagonal check, this is same for the 2 direction, just with different values
    while(within matrix frame){
        if(there is color that is being checked for win, going one of the
            diagonal way from that spot){

            increment a win value
        }else{
            break;
        }
    }
    while(within matrix frame){
        if(there is color that is being checked for win, going the other way
            diagonal way from that spot){

            increment a win value
        }else{
            break;
        }
    }

    if(win value>=4){
            save which values are connected
            set the win and returns a true for winning
    }

    return false if nothing is true
}
```

<u>Algorithm for 4 flashing dots isr:</u>

```
CY_ISR(fl) {

    if(if winning spots save from check win is lit){
        turn it off
    }else{
        turn it on
        }
    read timer status to exit
}
```

After all of that are done, it will loop from beginning again and keep the greeting loop and goes to the enemy loop. The enemy loop is exectly the same as the player loop but one difference, it checks the received signal for new move by the other player. If it is a new move it goes through the dropping animation and victory checking then goes back to the other player again if not won.

<u>The overall algorithm for the main loop:</u>

```
void main(){

      Start all the shematics

      set up the isr

      Wirte OE as 1 and SRCLR as 0

      initialize array for xbee

      sent code for XBEE command

      Print choice of starting order

      Print LCD

      Start the Senting timer

      for(;;){
         Greeting loop


        Player one{
            selecting spot for dot loop

            dropping dot animation

            update packet

            check for win

            change player turn
        }else player two{
            wait for enemy move

            dropping dot animation

            check for win

            chang player turn
        }
    }
```

Testing Strategy/Methodology:

The way I have tested my design was split into multiple small sections. First when I designed and connected my circuits, I did not use my PSOC 5 board at all. All I did was connect the LEDs anode and cathode to the power to see if I got my colors correct. After I have my shift registers implemented, I created a test function, which I can change whatever I want to display on the LED screen, and tried to see if I can like up any LED at Blue and Red. Then I tried to see if I can light up the whole LED without a transistor for 10 minutes and checked if the chips can handle it. I did my math first to see if current is high enough of course. The LED looks a bit duller but it works fine. Then as I implemented drop and other software function, I did my test my creating different situations and put it on the LED to see what it does and uses the details of what it did to debug until it work to the way of my liking. The wireless connect of the XBEE was the same where I tried to receive packets and display on my LED then sent packets of location I choose and see if other people are receiving it. So my method of testing is just do it and refine as I go. There are limits and other implementations that I did not put into consideration at the time of doing the project. For one, there are no time limit for how long the player have to wait. Also, there is no condition on what happens if the whole matrix but the home row is filled.

Design Challenges and Observations/Result:

Some challenges I have encounter are how to use have the data ready for the next row. I needed to figure out when and how to update the data and how I am suppose to refresh the ring counter. Other challenges are to get the XBEE working, there was simily too much loop in my code so I solved this issue with just updating the packet and sent it through isr of ½ second.

The result is a game that starts with a smilly face, which choses if I am going first by pressing the two buttons on capsense. Then the game lets you chose the spot to drop the dot and goes into a dot dropping animation. The player then has to wait for the other player's turn, the same thing happens for the other player's move, and we repeat until one win. When won, the 4 dots flashes and the game resets back to the greeting happy face page.

Q/A:

<u>Description of your software. How do the key modules of the software interact with each other? How did you make sure that an event such as the arrival of a packet or a display refresh event is serviced within an acceptable time limit?</u>

My software sents a packet to the XBEE every 0.5 second about the move I have selected. This ensures the arrival of packets by constantly senting out coppies of the packet so that the other player will get one of them eventually. The XBEE is independent from the LED modules hardwarely, but in software, the what the XBEE is senting is the array of the LED displayed on the matrix. The XBEE interacts with the LED matrix my receiving and changing the colors stored in the array. So if the moves of the enemy was sent and I received it, it will change the array for display. I regresh my ring counter when ever there is a loop in the code I wrote, because controlRing is a ring counter control which counts to the next like each time it is called. So if it is in a loop whenever I want to do something, the LED will refresh as I am doing my work. For other parts where there are no loops, it does not matter since the code is going so fast, our eyes can not see the delays.

<u>Discussion of key design decisions made. What factors drove your decisions on the hardware and software design for interfacing the 8x8 LED display? How did you determine the refresh rate of the display?</u>

I choose to work with shift registers instead of a control register because the shift register consumes way less resources such as pins and current. I choose to use an Array to keep tract of the LED lit up is because I can manupulate it freely as I do my coding. The transistors was not implemtned in my design is because the project work without it and the chips seems to be able to handle it as I been working with them for many hours. The refresh rate of display was simply determined by going to a lowest rate in which the project still work. This is because I did not want to see any flickering. The control registers on my schematic is used because I can control two clocks at the same time freely as I like. The timers I used could have been any kind of timer, but I just wanted to try out the sleep timer and thus there is one sleep timer out of the three. Other than those, all the other decision on want to use are mostly from what I was taught in class and bais perferences.

A discussion of how you assembled tested your system. Include any test programs used. You should maintain detailed notes during the design and testing of your project to help you write the documentation.

All the tests I have on my system was to create dots on the LED and to play the game. That is all I did after getting through the hardware part of the design. There wasn't much I did for testing but to write code and play to get feed back.

Conclusion:

In conclusion, the whole project is very interresting and where everything we learned in class or other labs was to use. This project took the knowledge that I have and made it so I can put them into a bigger project. How an design is made or how the design should work appear to be very important. There are many different ways to approach all these problems and to design what we have in mind, there are no limit to how we use our knowledge gain from this class.

Code Appendix (project code):

The following contains all the codes I have on this project which I wrote mostly in functions. They are commented.

```
/* ========================================

 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
#include <device.h>
#define Blue 1
#define myColor 1
#define Red 2
#define Nothing 0


/*variables used in program*/
uint8 opponID = 0x85;
uint8 ID = 0x12;
uint32 DelayTime = 2;
uint32 GRow = 8;
uint32 arrayLED[8][8];
uint8  setButton = 0;
uint8  currentSpot = 1;
uint8  setSpot = 0;
uint16 CSpot;
int8    selecting = 7;
uint8   setLocation;
uint8   Horizontal[8];
uint8   Vertical[8];
uint8   Flashing = 0;
uint8   Winner = 0;
uint8   twist = 0;
uint8   z =0;
uint8   flag;
uint8   home = 1;
uint8   playerTurn;
uint8   checkSum;
uint8   GT = 0;
uint8 Senting[13];
uint8 Received[13];



/*interrupt to flash after win*/
CY_ISR(fl)
{
    /*if winning spot is lit, turn it off*/
    if(twist == 0){
        for(z=0; z<4; z++){
            arrayLED[Vertical[z]][Horizontal[z]] = Nothing;
        }
        twist = 1;
    }else{
        /*if winning spot is off, turn it on*/
        for(z=0; z<4; z++){
            arrayLED[Vertical[z]][Horizontal[z]] = Winner;
        }
        twist = 0;
    }
    /* # the flashes = Flashing/2*/
    Flashing++;
    Timer_ReadStatusRegister();
}
```

```
/*sleep timer interrupt for when the dics is dropping down*/
CY_ISR(sleep)
{
    /* if the next LED on the panel is not taken, keep dropping*/
    if((selecting>0) && (arrayLED[selecting-1][setLocation] == Nothing)){
        arrayLED[selecting-1][setLocation] = arrayLED[selecting][setLocation];
        arrayLED[selecting][setLocation] = Nothing;
        selecting --;
    }else{
        setSpot = 1;
    }
    SleepTimer_GetStatus();
}

/* resets the all the arrays to nothing*/
void resetArray(){
    uint8 h, t;
    for(h=0; h<8; h++){
        for(t=0; t<8; t++){
            arrayLED[h][t] = Nothing;
        }
    }
    for(h=0; h<4; h++){
        Horizontal[h] = Nothing;
        Vertical[h] = Nothing;
    }
}

/*prepares the reds on the next row to be displayed by ring counter*/
void PrepNextRowRed(){
    int8 i;
    for(i = 7; i>=0; i--){
        if(arrayLED[GRow][i]==Red){
            SER_Red_Write(0);
        }else{
            SER_Red_Write(1);
        }
        Control_Reg_Red_Write(1);
        Control_Reg_Red_Write(0);
    }
}

/*prepares the Blue on the next row to be displayed by ring counter*/
void PrepNextRowBlue(){
    int8 i;
    for(i = 7; i>=0; i--){
        if(arrayLED[GRow][i]==Blue){
            SER_Blue_Write(0);
        }else{
            SER_Blue_Write(1);
        }
        Control_Reg_Blue_Write(1);
        Control_Reg_Blue_Write(0);
    }
}

/*prints out the prepared Red and Blue LEDs*/
```

```
void DisplayDots(){
    Control_Reg_Red_Write(2);
    Control_Reg_Red_Write(0);
    Control_Reg_Blue_Write(2);
    Control_Reg_Blue_Write(0);
}

/*setting pins of given color to given row and column*/
void settingPin(int color, int row, int column){
    if(arrayLED[row-1][column-1] == Nothing){
        arrayLED[row-1][column-1] = color;
    }
}

/*renews the ring counter value from row 8 to 1*/
void checkRing(){
    if(GRow == 8){
            SER_Write(1);
            GRow = 0;
    }
}

/*ring counter which counts once each time called*/
void Control_Ring(){
    /*renew GRow and SER for ring counter for a new loop*/
    checkRing();
    /*clock in the next red and blue values*/
    PrepNextRowRed();
    PrepNextRowBlue();
    /*preparing row ring counter*/
    Control_Reg_Write(1);
    CyDelay(DelayTime/2);
    /*latches the value*/
    DisplayDots();
    /*hit the next row*/
    Control_Reg_Write(2);
    CyDelay(DelayTime/2);

    /*make ser 0 when its 1 so its high only once*/
    if(SER_Read() == 1){
        SER_Write(0);
    }
    /*count*/
    GRow++;
}

/*sets the values that are senting every halve a second*/
void setPacket(uint8 setID, uint8 setRow, uint8 setColumn){
    checkSum = 0;
    Senting[0] = 0x7E;
    Senting[1] = 0x00;
    Senting[2] = 0x09;
    Senting[3] = 0x01;
    Senting[4] = 0x00;
    Senting[5] = 0xFF;
    Senting[6] = 0xFF;
    Senting[7] = 0x01;
    Senting[8] = setID;
    Senting[9] = 0x00;
    Senting[10] = setRow;
    Senting[11] = setColumn;
    uint8 i;
    /*do the checksum*/
    for(i=3; i<12; i++){
        checkSum += Senting[i];
        Control_Ring();
    }
    Senting[12] = 0xFF - checkSum;
}

/*checks the move that the enemy is senting*/
void receivePacket(){
    uint8 i, R;
    uint8 E = 1;
    checkSum = 0;
    /*if there is still stuff in buffer*/
    while(UART_GetRxBufferSize()!=0 && E==1){
        /*check for the starting of packet*/
        R = UART_GetChar();
        if(R == 0x7E){
            Received[0] = R;
            /*continue to receive the reset of packet until the end*/
            for(i = 1; i<13; i++){
                /*if the reset of the packet is not here yet, wait for it*/
                while(UART_GetRxBufferSize()==0){
                /*while waiting, keep refreshing*/
                Control_Ring();
                }
                Received[i] = UART_GetChar();
                Control_Ring();
                }
            E = 0;
            }
            Control_Ring();
        }
    /*check if file is corrupted*/
    for(i=3; i<12; i++){
        checkSum += Received[i];
        Control_Ring();
    }
    /*check for correct parameters*/
    if((Received[12] == 0xFF -
checkSum)&&((Received[11]<=8)&&(Received[11]>0))&&((Received[10]<8)&&(Received[10]>
0))&&(opponID==Received[8])){
        GT = 1;
        Control_Ring();
    }
}

/*check the homerow value*/
void ChangeHome(uint8 x){
    uint8 t;
    for(t=0; t<8; t++){
        arrayLED[7][t] = Nothing;
    }
    settingPin(myColor, 8, x);
}
```

```
/*make change of the home row if the senser is moved and at a new spot*/
void selectSpot(spot){
    /*the values was gotten from have FFFF/8, but the range was limited to halve
        so I divided my limite by have. So my values was gotten from
experimentation*/
    if((1<=spot) && (spot <31/2.5) && (currentSpot != 1)){
        ChangeHome(1);
        currentSpot = 1;
    }
    if((31/2.5 <= spot) && (spot < 63/2.5) && (currentSpot != 2)){
        ChangeHome(2);
        currentSpot = 2;
    }
    if((63/2.5 <= spot) && (spot < 95/2.5) && (currentSpot != 3)){
        ChangeHome(3);
        currentSpot = 3;
    }
    if((95/2.5 <= spot) && (spot < 127/2.5) && (currentSpot != 4)){
        ChangeHome(4);
        currentSpot = 4;
    }
    if((127/2.5 <= spot) && (spot < 159/2.5) && (currentSpot != 5)){
        ChangeHome(5);
        currentSpot = 5;
    }
    if((159/2.5 <= spot) && (spot < 191/2.5) && (currentSpot != 6)){
        ChangeHome(6);
        currentSpot = 6;
    }
    if((191/2.5 <= spot) && (spot < 223/2.5) && (currentSpot != 7)){
        ChangeHome(7);
        currentSpot = 7;
    }
    if((223/2.5 <= spot) && (spot < 255/2.5) && (currentSpot != 8)){
        ChangeHome(8);
        currentSpot = 8;
    }
}

/*I put my values to show on the led here for testing*/
void test(){
    int8 t;
    for(t=1; t<=8;t++){
        settingPin(Blue, 2, t);
        settingPin(Red, 6, t);
    }
}

/*check if the given color have won the game*/
uint8 checkVictory(int8 color, int8 S, int8 L){
    uint8 f, t, v, h, n, d1, d2;
    v = h = f = 0;
    d2 = 1;

    /*horizontal check*/
    for(t = 0; t<8; t++){
        if(arrayLED[S][t] == color){
            Horizontal[f] = t;
            Vertical[f] = S;
            f++;
            h++;
        }else{
            h =0;
            f=0;
        }
        if(h>=4){
            Winner = color;
            return 1;
        }
    }
    /*vertical check*/
    f = 0;
    for(t = 0; t<7; t++){
        if(arrayLED[t][L] == color){
            Horizontal[f] = L;
            Vertical[f] = t;
            f++;
            v++;
        }else{
            f=0;
            v =0;
        }
        if(v>=4){
            Winner = color;
            return 1;
        }
    }

    /*diagonal / check*/
    f = 0;
    d1 = 1;
    n = 1;
    Horizontal[f] = L;
    Vertical[f] = S;
    f++;
    while(((S-n)>=0) && ((L-n)>=0)){
        if(arrayLED[S-n][L-n] == color){
            Horizontal[f] = L-n;
            Vertical[f] = S-n;
            f++;
            d1++;
        }else{
            break;
        }
        n++;
    }
    n=1;
    while(((S+n)<7) && ((L+n)<8)){
        if(arrayLED[S+n][L+n] == color){
            Horizontal[f] = L+n;
            Vertical[f] = S+n;
            f++;
            d1++;
        }else{
            break;
        }
        n++;
    }
```

```
    if(d1>=4){
        Winner = color;
        return 1;
    }
    /*diagonal \ check*/
    f = 0;
    Horizontal[f] = L;
    Vertical[f] = S;
    f++;
    d2 = 1;
    n = 1;
    while(((S+n)<7) && ((L-n)>=0)){
        if(arrayLED[S+n][L-n] == color){
            Horizontal[f] = L-n;
            Vertical[f] = S+n;
            f++;
            d2++;
        }else{
            break;
        }
        n++;
    }
    n=1;
    while(((S-n)>=0) && ((L+n)<8)){
        if(arrayLED[S-n][L+n] == color){
            Horizontal[f] = L+n;
            Vertical[f] = S-n;
            f++;
            d2++;
        }else{
            break;
        }
        n++;
    }
    if(d2>=4){
        Winner = color;
        return 1;
    }
    return 0;
}

/*set up happy face on greeting page*/
void happyFace(){
    settingPin(Red, 5, 2);
    settingPin(Red, 5, 3);
    settingPin(Red, 5, 6);
    settingPin(Red, 5, 7);
    settingPin(Red, 6, 2);
    settingPin(Red, 6, 3);
    settingPin(Red, 6, 6);
    settingPin(Red, 6, 7);
    settingPin(Red, 7, 2);
    settingPin(Red, 7, 3);
    settingPin(Red, 7, 6);
    settingPin(Red, 7, 7);
    settingPin(Blue, 3, 2);
    settingPin(Blue, 3, 7);
    settingPin(Blue, 2, 6);
    settingPin(Blue, 2, 5);
    settingPin(Blue, 2, 4);
    settingPin(Blue, 2, 3);
}

/*interrupt to sent a already set packet to xbee*/
CY_ISR(st)
{
    UART_PutArray(Senting, 13);
    Timer_Sent_ReadStatusRegister();
}

void main()
{
    /*initialization/startup code here*/
    uint8 y;
    resetArray();
    CapSense_CSD_Start();
    LCD_Start();
    UART_Start();

    /*set up the isr*/
    CYGlobalIntDisable;
    isr_sleep_Start();
    isr_sleep_SetVector(sleep);
    isr_flash_Start();
    isr_flash_SetVector(fl);
    isr_Sent_Start();
    isr_Sent_SetVector(st);
    CYGlobalIntEnable;

    /* Initialize baselines */
    CapSense_CSD_InitializeAllBaselines();
    /*set the clear and output signal for transistor*/
    OE_Write(0);
    SRCLR_Write(1);

    /*initialize array for xbee*/
    for(y=0; y<13; y++){
        Received[y]=0;
        Senting[y] = 0;
    }

    /*set up xbee*/
    CyDelay(3000);
    UART_PutString("+++");
    CyDelay(2000);
    UART_PutString("ATAP1\r");
    UART_PutString("ATCH F\r");
    UART_PutString("ATWR\r");
    UART_PutString("ATCN\r");

    /*Print choice of starting order*/
    LCD_Position(0,0);
    LCD_PrintString("Left: I am 1st");
    LCD_Position(1,0);
    LCD_PrintString("Right: I am 2nd");

            Timer_Sent_Start();
    /*program*/
```

```
for(;;)
{
    /*greeting page*/
    while(home == 1){
    /*update senser values*/
        CapSense_CSD_UpdateEnabledBaselines();
      CapSense_CSD_ScanEnabledWidgets();
        happyFace();
        /*when capsense not touched, just fresh leds*/
        while(CapSense_CSD_IsBusy() != 0){
            Control_Ring();
        }
        /*if button 0 is pressed, the player one is set to go first*/
        if (CapSense_CSD_CheckIsWidgetActive(CapSense_CSD_BUTTON0__BTN)){
            playerTurn = 1;
            home = 0;
            resetArray();
            CyDelay(500);
        }
        /*if button 1 is pressed, the player two is set to go first*/
        if (CapSense_CSD_CheckIsWidgetActive(CapSense_CSD_BUTTON1__BTN)){
            playerTurn = 2;
            home = 0;
            resetArray();
            CyDelay(500);
        }
        Control_Ring();
    }
    /*set the senser back to 0*/
    CapSense_CSD_InitializeAllBaselines();

    /*if it is player 1's turn*/
    if(playerTurn == 1){
        /*set the inital spot for homerow*/
        ChangeHome(currentSpot);
        Control_Ring();

        /*set up printing of position on LCD*/
        LCD_ClearDisplay();
        LCD_Position(0,0);
        LCD_PrintString("Position:");

        /*choosing where to place disc*/
        while(setButton==0){
            /*update data*/
            CapSense_CSD_UpdateEnabledBaselines();
          CapSense_CSD_ScanEnabledWidgets();

            /*print current spot to LCD*/
            LCD_Position(0,9);
            LCD_PrintInt8(currentSpot);
            /* Wait for scanning to complete */
                while(CapSense_CSD_IsBusy() != 0){
            Control_Ring();
            }

            /*check if button pushed*/
            if (CapSense_CSD_CheckIsWidgetActive(CapSense_CSD_BUTTON0__BTN)){
                setButton = 1;
                }
                if (CapSense_CSD_CheckIsWidgetActive(CapSense_CSD_BUTTON1__BTN)){
                    setButton = 1;
                }

                /* adjust position */
                CSpot =
CapSense_CSD_GetCentroidPos(CapSense_CSD_LINEARSLIDER0__LS);

                /*if not error change spot*/
                if(CSpot!= 0xFFFF){
                    selectSpot(CSpot);
                }

                Control_Ring();
            }

            CapSense_CSD_InitializeAllBaselines();

            /*checks what spot play selected to drop disc*/
            for(y=0; y<8; y++){
                if(arrayLED[7][y] != Nothing){
                    setLocation = y;
                }
                Control_Ring();
            }


            /*start timer for droping the disc*/
            SleepTimer_Start();
            SleepTimer_Init();

            /*wait dropping the disc*/
            while(setSpot == 0){
                Control_Ring();
            }

            /*stop timer so it does not interrupt when dropping*/
            SleepTimer_Stop();


            setPacket(ID, selecting, setLocation);//Set packet
****************************************/

            /*check if Blue won after their move*/
            if(checkVictory(Blue, selecting, setLocation)){
                /*start timer for interrupt flashes*/
                Timer_Start();
                Timer_Init();
                /*flash for 5 times*/
                while(Flashing <10){
                    Control_Ring();
                }
                /*stop the timer to stop flashes*/
                Timer_Stop();
                CyDelay(1000);
                resetArray();
                home = 1;
            }
            playerTurn =2;
        }else{
            LCD_ClearDisplay();
            LCD_Position(0,0);
            LCD_PrintString("Turn: Red");
            LCD_Position(1,0);
            LCD_PrintString("Waiting...");
            GT = 0;
            while(GT == 0){
                receivePacket();
                if((arrayLED[Received[10]-1][Received[11]-1] != Nothing)){
                    GT = 0;
                }
                Control_Ring();
            }
            setLocation = Received[11]-1;
            settingPin(Red, 8, Received[11]);
            SleepTimer_Start();
            SleepTimer_Init();

            /*wait dropping the disc*/
            while(setSpot == 0){
                Control_Ring();
            }

            /*stop timer so it does not interrupt when dropping*/
            SleepTimer_Stop();

            if(checkVictory(Red, selecting, setLocation)){
                /*start timer for interrupt flashes*/
                Timer_Start();
                Timer_Init();
                while(Flashing <10){
                    Control_Ring();
                }
                /*stop the timer to stop flashes*/
                Timer_Stop();
                CyDelay(1000);
                resetArray();
                home = 1;
            }
            playerTurn = 1;
        }

        /*resets value for next turn*/
        z =0;
        Winner = 0;
        twist = 0;
        Flashing =0;
        selecting = 7;
        setSpot = 0;
        setButton = 0;
    }
}

/* [] END OF FILE */
```