Connect 4

Project report

Yuchen Luo

X00192070

This is my own work. Any material taken from other sources has been fully referenced in the text of the work. All sources used in the preparation of this work have been listed in the References.

Contents

Figures

1. Abstract

This report is about a 'Connect 4' two-player game. The subsystems: MSP430G2553 Subsystem, Led Matrix Subsystem, Seven Seg Displays Subsystem and the Switch Circuit Subsystem. The structure of the various subsystems and the required operations are systematically described in this report, together with the code that can be implemented. The components of the various subsystems and the required operations are systematically described in this report, together with the code that can be implemented. The relevant block diagram describes how the circus works. It also describes how to test the components for a more intuitive understanding. In addition to the game itself, new features are made in other game program sections. In the end, it talks about how to improve and gain. This project was completed in c and involved knowledge of Embedded Systems as well as object-oriented programming.

keywords: Software, Launchpad, Matrix, Switch, Programming, Game

2. Preface

I would like to express my very great appreciation to Michael Gill for his patient guidance, enthusiastic encouragement and useful critiques of this project. His willingness to give his time so generously has been very much appreciated.

I'd also want to thank my classmates for their technical advice and assistance

3. Introduction

- Introduction to the Connect 4 system:
  An initial introduction to c4 subsystems, including a user guide.

- MSP430 Section:

A brief explanation of the MSP430 that was utilized, as well as the attributes of the MSP430. Describe the registers that are used to control the GPIOs

- LED Matrix Section:

A description of the circuit's operation. The MSP430 and the matrix interface. Explain the bus protocol and how it functions. Describe how header files, source files, include guards, and pre-processor macros are used to create modular embedded applications. Explain how the Matrix's driver code works and how the Matrix was tested.

- Switch Circuit

The connection between the switch and the MSP430. The switch's driver/debounce code functions, how it was modularized, and how it was tested.

- Connect 4 Program

The program's operation. How this game works with getMove, placeMove and is Winner

- Other Game Programs

Special functions to record the result of each round

4. Introduction to the Connect 4 system

Here are four main components. MSP430 connect to the LED Matrix with the bus. One side of the Switch to the ground, and another side connects to the Msp430.

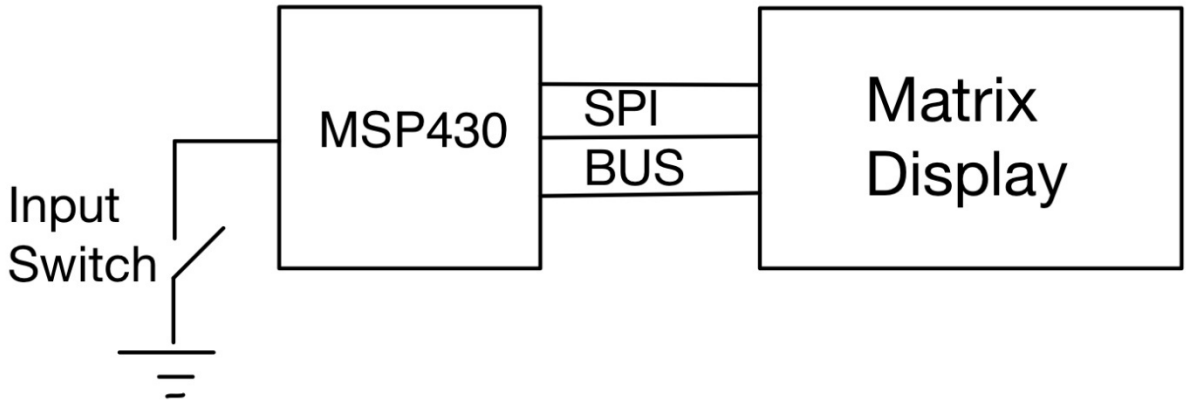

Figure 1 Block diagram of the system

User Guide: Connect 4

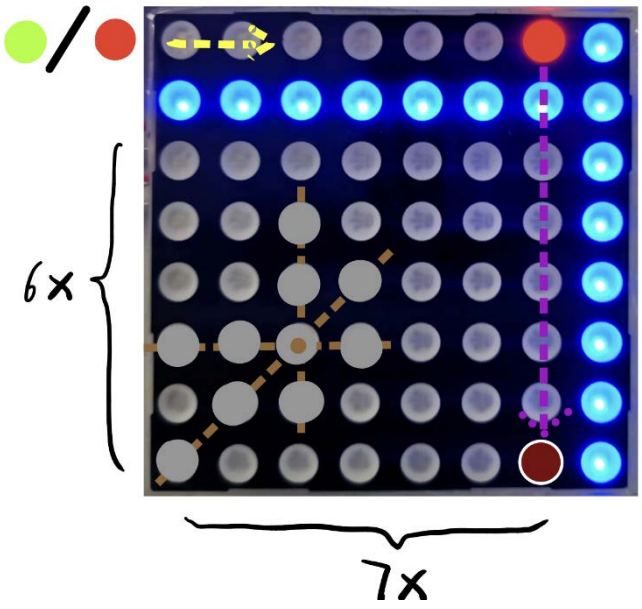

Figure 2 User guide

- This is a two-player connection board game, in which the players choose a colour and then take turns dropping coloured LEDs into a seven-column, six-row vertically suspended grid.
- There is a LED shifting from left to right within the first row again and again. When the player presses the button, which colour drops straight down, occupying the lowest available space within the column.
- The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's colours.

5. MSP430 Section

The **MSP430** is a mixed-signal microcontroller family from Texas Instruments, first introduced on 14 February 1992. Built around a 16-bit CPU, the MSP430 is designed for low cost and, specifically, low power consumption embedded applications.

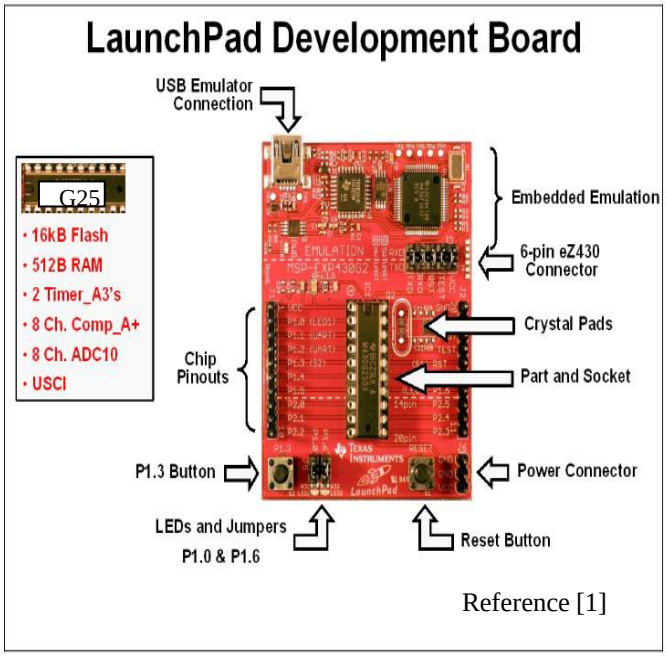

Figure 3 LaunchPad Development Board

Launchpad Board Features:[1]

- **USB interface:** This allows A PC to download a compiled program to the launchpad, run the program, serially transfer data to/from the Launchpad and debug a running program.
- **Reset switch:** To reset the board.
- **2 User Leds:** Writeable from a running program.
- **1 User Switch:** Readable from a running Program

Texas Instruments (TEXAS INSTRUMENTS, TI) MSP430TM series of microcontrollers (MCU) is a 16-bit mixed-signal processor based on Reduced Instruction Set Computing (RISC). The chip has an integrated Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC), which allows it to receive and output not only digital signals but also analogue signals, hence the name Mixed Signal Processor. The block diagram of the MSP430 family of microcontrollers is shown in Figure 2. [2]

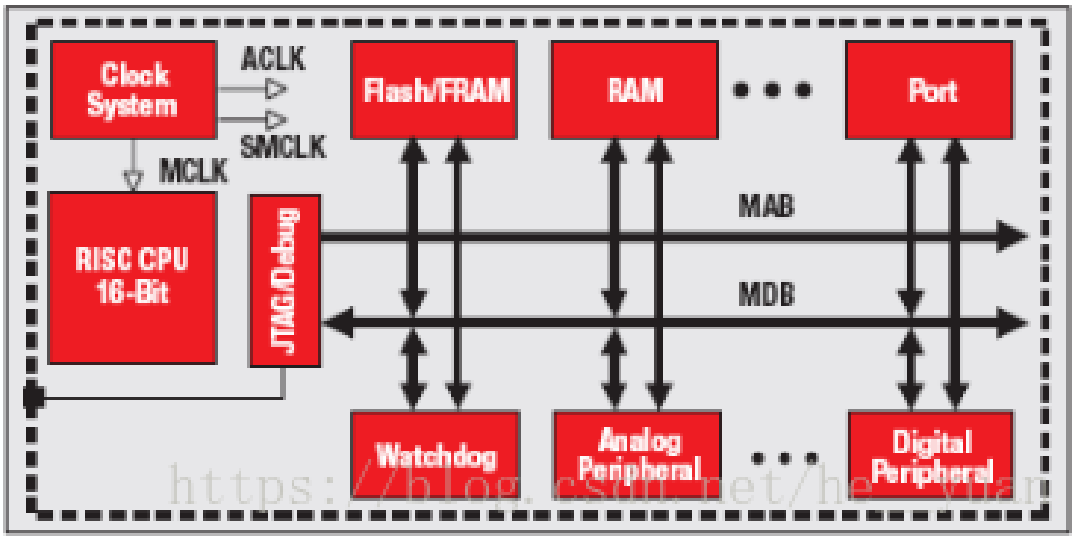

Figure 4 Block diagram of the MSP430 series of microcontrollers

GPIO Registers in MSP430 [3]

The GPIO block has many registers. We will only go through some of the digital I/O registers that are within the scope of this project.

1. PxDIR: This is the GPIO direction control register. Setting any bit to 0 in this register will configure the corresponding Pin[0 to 7] to be used as an Input while setting it to 1 will configure it as Output.

2. PxIN (Readonly): Used to Read values of the Digital I/O pins configured as Input. 0 = Input is LOW, 1 = Input is HIGH.

3. PxOUT: Used to directly write values to pins when pullup/pulldown resistors are disabled. 0 = Output is LOW, 1 = Output is HIGH. When pullup/pulldown resistors are enabled: 0 = Pin is pulled down, 1 = Pin is pulled up.

4. RxREN: For pins configured as input, PxREN is used to enable a pullup/down resistor for a given pin and PxOUT in conjunction with PxREN is used to select either Pullup or pulldown resistor. Setting a bit to 1 will enable pullup/down resistor for the corresponding pin while setting it to 0 will disable the same.

PxDIR	PxREN	PxOUT	I/O Config
0	0	X	Input with resistors disabled
0	1	0	Input with Internal Pulldown enabled
0	1	1	Input with Internal Pullup enabled
1	X	X	Output – PxREN has no effect

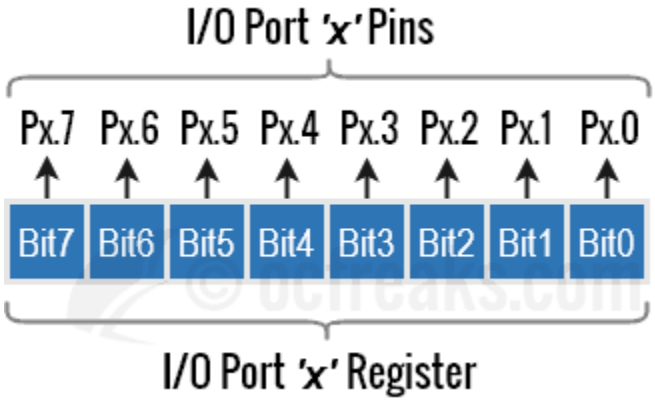

Figure 5 I/O Port Pins

6. LED Matrix Section

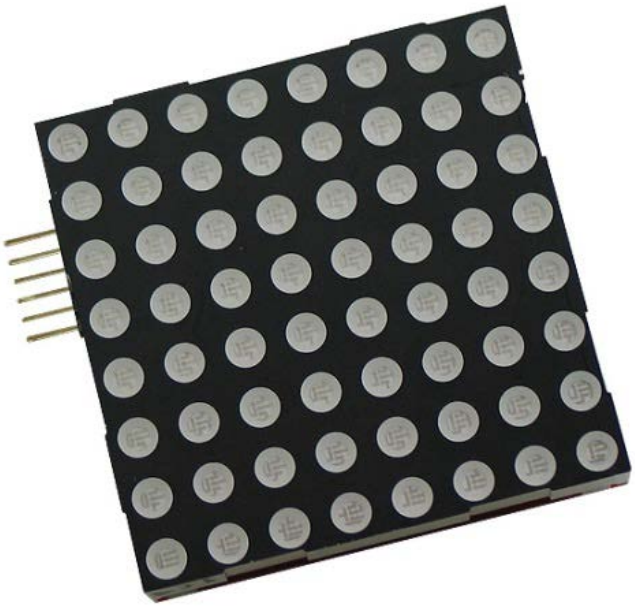

Figure 6 LED Matrix 1

MOD-LED8x8RGB is an intelligent 8×8 RGB LED module, that allows many modules to be stacked together.

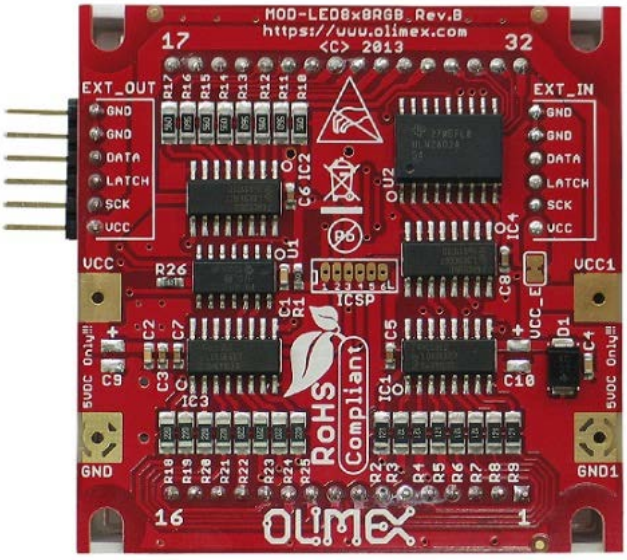

Figure 7 LED Matrix 2

MOD-LED8x8RGB works with a 5V power supply as you can see on the two sides there are big copper pads named VCC/GND and VCC1/GND1 when modules

arsnappedap together permanently (if you make big screen) is good these VCC-VCC1 and GND-GND1 to be connected between the modules so better power distribution is achieved.[4]

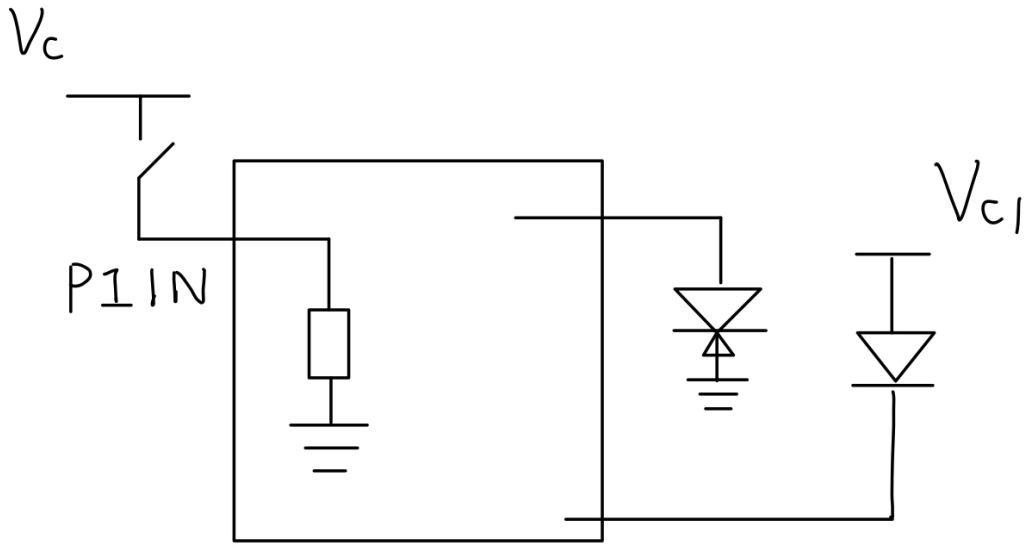

Figure 8 Block diagram of LED Matrix

Features[5]

- 3-color LEDs
- Supplied by 5V DC, either from the host board or externally
- Easy to stack multiple ones to build any LED matrix
- 4 mounting cuts for firm positioning
- UEXT to connect to device or host microcontroller
- Dimensions: 60.0 x 60.0 mm (2.36 x 2.36")

The driver code of the Matrix

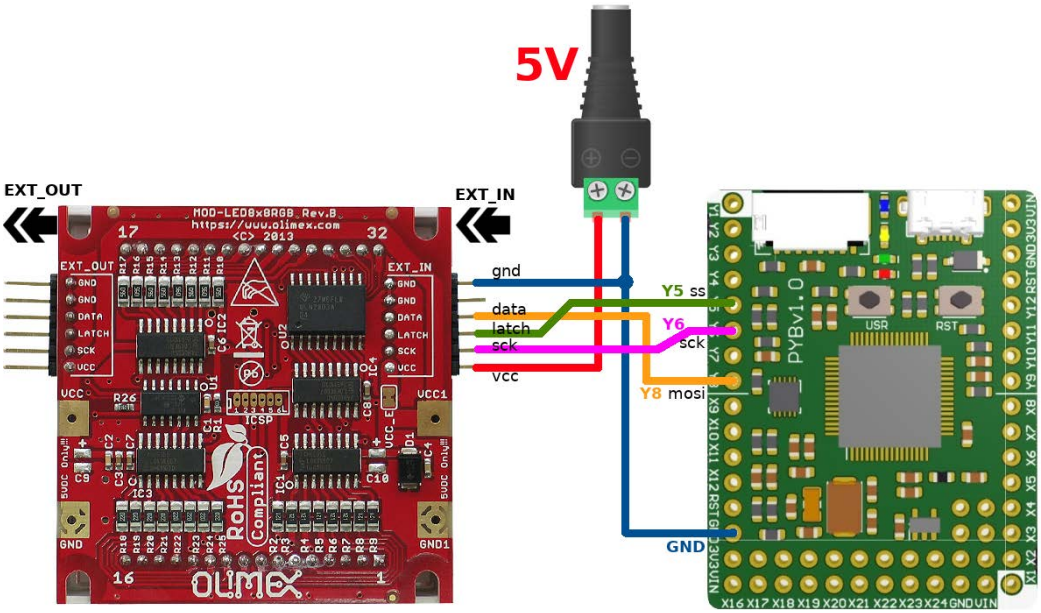

Figure 9 the Matrix connect to the Board [6]

To connect MOD-LED8x8RGB you will need ARDUINO working with 5V like OLIMEXINO-32U4 – make sure the power jumper is set to 5V as OLIMEXINO-32U4 can work both on 3.3 and 5V.

Then you should connect OLIMEXINO-32U4 to MOD-LED8x8RGB as in the above picture in this way:

- GND - GND
- 5V - Vcc
- P1.4 - Latch
- P1.5 - SCK
- P1.7 – Data

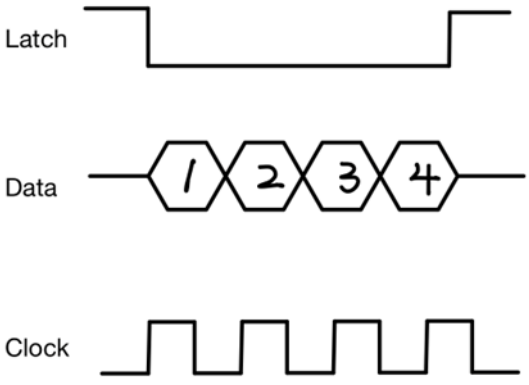

Figure 10 Signal

Header files&Sourse files

emphasizes separating the functionality of a program into independent, interchangeable **modules**, such that each contains everything necessary to execute only one aspect of the desired functionality.

- #include <msp430.h>
- #include <stdint.h>
- #include "mtxdisp.h"

```
/*
 * mtxdisp.h
 */
#ifndef MTXDISP_H
#define MTXDISP_H

#include <msp430.h>
#include <stdint.h>

#define MTXDIR P1DIR
#define MTXOUT P1OUT
#define MTXDATA BIT7
#define MTXCLK BIT5
#define MTXLATCH BIT4

void initMtx(void);
void clearMtx(void);
void lightMtx(const uint8_t mtxBuf[8][3]);

#endif /* MTXDISP_H */
```

Launchpad Software: [1]

- **IDE/C/C++ Compilers:**

o **TI Code Composer Studio (CCS) Full version**: Cost $500 approx. The department has obtained the full version of CCS under the TI university program and it is available in the labs. This is the compiler that will be used for this course

Testing

```
/*
 * mtxdisp.c
 */
#include <stdint.h>
#include "mtxdisp.h"

void initMtx(void)
{
    MTXDIR |= MTXDATA + MTXCLK + MTXLATCH; ////Makes Data,Latch and Clk pins outputs
    MTXOUT &= ~(MTXDATA + MTXCLK);//Makes Data and Clk low
    MTXOUT |= MTXLATCH;    //Make Latch high as it is active low
}

void clearMtx(void)
{//clocks in 8*3*8(192) lows to matrix (Clears matrix)

    int x;
    MTXOUT &= ~MTXLATCH;    //Latch low
    MTXOUT &= ~MTXDATA;     //Data low
```

```
    for (x=1; x<=192;x++)
    {
        MTXOUT |= MTXCLK; //Clk high
        MTXOUT &= ~MTXCLK;//Clk low
    }

    MTXOUT |= MTXLATCH; //Latch high
}

void lightMtx(const uint8_t mtxBuf[8][3])
{// displays mtxBuf values on matrix displays

    int x,colour,row;
    uint8_t val;

    MTXOUT &= ~MTXLATCH;    //Latch low

    for(row=7;row>=0;row--)    //for each row from 7 downto 0
    {
        for(colour=2;colour>=0; colour--)
        {//for each colour from 2 downto 0
            val = mtxBuf[row][colour];

            for (x=1; x<=8;x++)    //for each bit in val shift it out
            {
                if (val&BIT0) // is LSb of val high
                    MTXOUT |= MTXDATA; //Data high
                else
                    MTXOUT &= ~MTXDATA; //Data low
```

```
                MTXOUT |= MTXCLK; //Clk high
                MTXOUT &= ~MTXCLK;//Clk low
                val>>=1;
            }
        }
    }
    MTXOUT |= MTXLATCH; }//Latch high
```

7. Switch Circuit

The switch can be interfaced to MSP 430. The following program explains how to interface switch to MSP430.

There are LEDs connected at P1.6 and P1.0, the switch SW1 is connected in P1.3. Initially, when the switch is not pressed, the LED connected at P1.0 should turn off the and LED connected at P1.6 should turn on. LEDs should toggle after each changing position of the switch. [7]

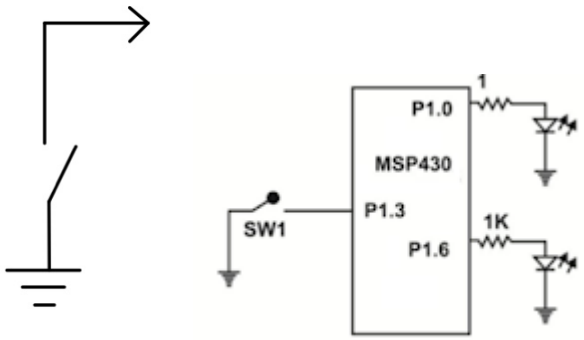

Figure 11 Block diagram of Switch

IDE/C/C++ Compilers:[1]

TI Code Composer Studio (CCS) Full version: Cost $500 approx. The department has obtained the full version of CCS under the TI university program and it is available in the labs. This is the compiler that will be used for this course

Button Bounce[8]

However, when the button is pressed, pin P1_3 does not change directly from high to low. In reality, we get a behaviour called bounce which can be seen in Diagram 5.5. This will happen when the switch contacts are made or broken.

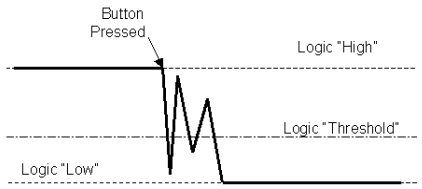

Figure 12 Button Bounce [1]

to poll the switch pin until it changes, then wait for 20ms and then rreadsit again to see if the change is valid.

a function called readSwDB() based on the debounce code above that returns:

If the switch has been pressed since the last time the function was called.

-1 If the switch has been released since the last time the function was called.

If there has been no change since the last time the function was called.

In this tog toggle, the RED LED when the switch is pressed and toggle the GREEN LED when the switch is released.

```
        initialiseSwitch(); //set up switch for reading with pullup resistors

    while(1)
    {
        val=readSwitchPin(); //read switch
        if( val != lastRead ) //check if it has changed
        {

        delay(20); //wait for 20ms
        val=readSwitchPin(); //read switch
```

Testing

int main(void)

```
{
 uint8_t BGR[8][3]={
                                    {0x00,0x00,0x00},
                                    {0xFF,0x00,0x00},
                                    {0x01,0x00,0x00},
                                    {0x01,0x00,0x00},
                                    {0x01,0x00,0x00},
                                    {0x01,0x00,0x00},
                                    {0x01,0x00,0x00},
                                    {0x01,0x00,0x00}
                                };

      volatile long x;
      uint8_t ch = BIT7, val;

WDTCTL = WDTPW | WDTHOLD;              // Stop watchdog timer

initialiseSwitch(); //set up switch for reading with pullup resistors
      initMtx();

      clearMtx();

      while (1)
      {
                  BGR[0][2]=ch;
                  lightMtx(BGR);
                  for(x=0;x<4000;x++)
                  {
                        val=readSwitchDB(); //read switch Debounced
```

```
                    if( val == DOWN ) //check if switch down
                    {
                        //added ch position to bottom row
                        BGR[7][2] |= ch;    //BGR[7][2]= BGR[7][2] | ch;
                    }
                }
                ch =ch>>1;              //ch>>=1;
                if (ch==BIT0)
                {
                    ch=BIT7;
                }
            }

            return 0;
        }
/*
 * switch.c
 */
#include "switch.h"
#include "delay.h"
#include <msp430.h>
unsigned char readSwitchPin(void);

void initialiseSwitch(void)
{
    //put comments here
    P1DIR &= ~BIT3; // Clear bit 3 making P1_3 an input i.e ~BIT3 (11110111
BIN)
    P1OUT |= BIT3; // Set bit 3 of P1OUT to select pullup i.e BIT3(00001000BIN)
    P1REN |= BIT3; // Enable the pullup resister i.e BIT3 (00001000 BIN)
```

```
}

unsigned char readSwitchPin(void)
{
    //put comments here
    if( P1IN & BIT3 ) //BIT3 (00001000 BIN)
        return UP; //pin was high (Switch un-pressed)
    else
        return DOWN; //pin was low (Switch pressed)
}

unsigned char readSwitchDB(void)
{
    // put comments here
    static unsigned char lastRead=UP; // assume switch is un-pressed at startup
    unsigned char val;
    val=readSwitchPin(); //read switch

    if( val != lastRead ) //check if it has changed
    {
        delay(200); //wait for 20ms
        val=readSwitchPin(); //read switch

        if(val != lastRead ) //check switch again to make sure it has
        { // really changed
            lastRead = val; //update lastRead
            return val;
        }
    }
    return SAME;}
```

8. Connect 4 Program

To achieve the final effect, we need the following three functions in the main:

- getMove: To get Moves
- placeMove: To Place moves in the proper position
- isWinner: To judge whether someone won

```
do{
    if(player==RED)//alternate between red and green of players
        player= GREEN;
    else
        player = RED;

    move = getMove(BGR,player);//makes a move
    row= placeMove(BGR,player,move);//places a move

    numMoves++;//increment of the moves
    lightMtx(BGR);//lights the LED
    isWinner = playerWon(BGR,player,move,row);
  //Judges whether someone won

}while(numMoves<42&&!isWinner);//loops until draw or someone won
```

getMove:

- Loop a LED to shift from left to right in the row zero, delay for a while in each position
- Judge whether a valid move:

  Check row[2] whether occupied, making sure the program can be processed

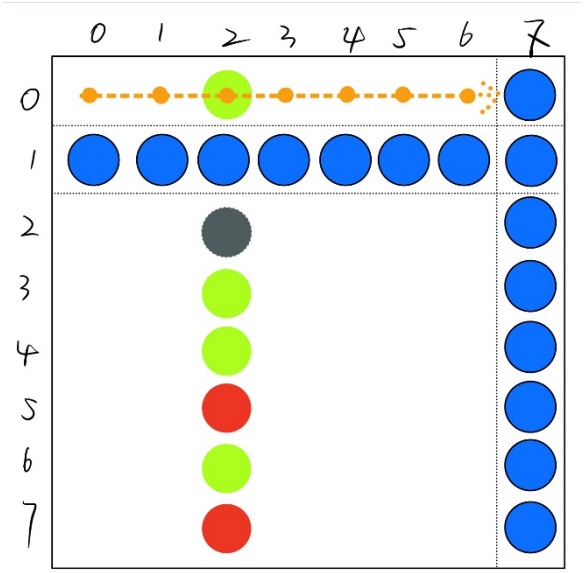

Figure 13 getMove

```
uint8_t getMove( uint8_t BGR[8][3],int player)
{
    unsigned char val,litLEDs;//val is the status of switch
    volatile long x;
    uint8_t ch=0x02;//start at rightmost position

    do{
        if(ch>0x02)//loop in the fist row from left to right
            ch = ch >> 1;//swift to the right
        else
            ch=0x80;//back to the left

        BGR[0][player]=ch;
        lightMtx(BGR);//light the LED

        for(x=0;x<20000;x++)
            ;//delay
```

```
        val=readSwitchDB();//checks the switch status
        litLEDs = BGR[2][GREEN]|BGR[2][RED];//check the row 2 position is
not occupied

    }while(val!=DOWN||(litLEDs&ch)!=0);//loop while the switch not down or this
position corresponding the position on row 2 is not occupied

    BGR[0][player]=0x00;//clear all on row 0
    return ch;
}
```

Place move:

- Make sure the red LED drop into the right position:
  Get the bottom position
- Find the empty position from the bottom up

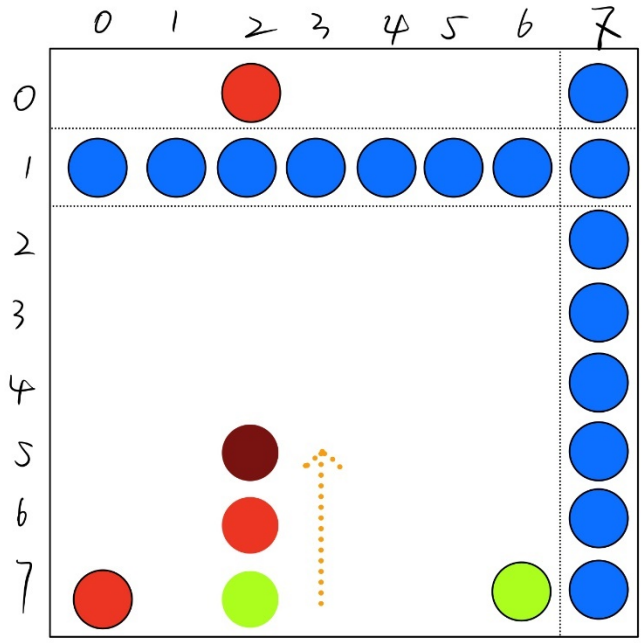

Figure 14 PlaceMove

```
int placeMove(uint8_t
BGR[8][3],int player,uint8_t
Move)
{
    uint8_t litLeds;
    int row = 7;
    litLeds = BGR[7][RED]|BGR[7][GREEN];//gets the bottom position
```

```
        while((litLeds&Move)!=0)//loop until find the empty position
        {
            row--;//check upper one
            litLeds = BGR[row][RED]|BGR[row][GREEN];//get the lowest empty
position
        }

        BGR[row][player]|=Move;//add move to the right position
        return row;
    }
```

isWinner:

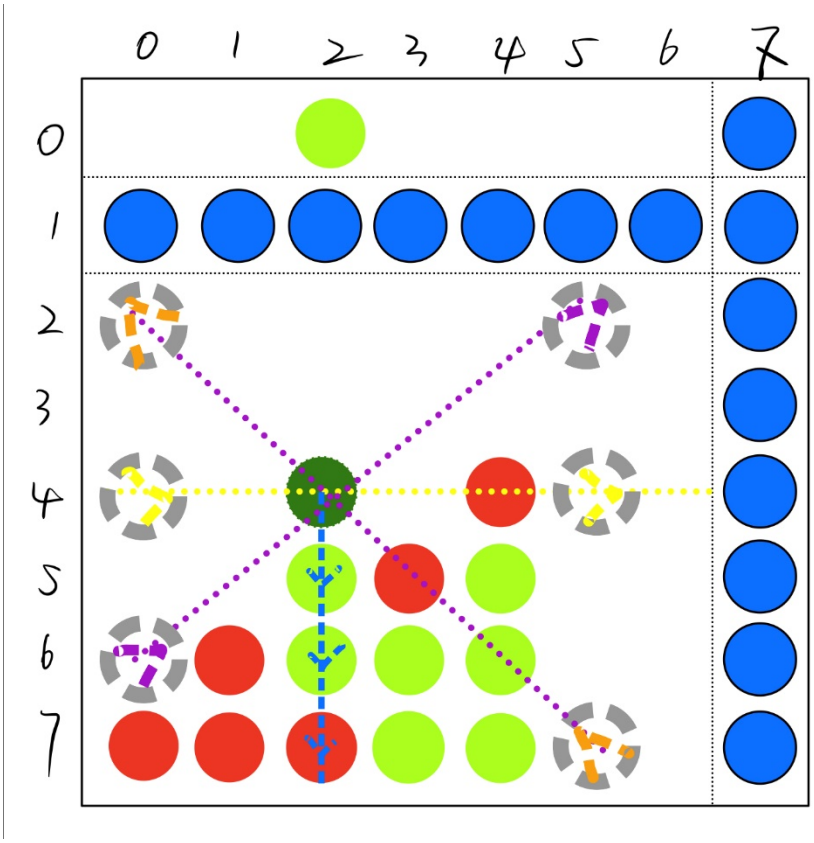

Figure 15 isWinner

- Horizontal:

  From Left to right check whether connected 4

- Vertical:

  Check whether has enough room below, Check four connected positions below from the top down

- Diagonal:

  Find the bottom left position, Check four connected positions to the top right.

  Find the bottom right position, Check four connected positions to the top left

```
//horizontal
int playerWon( uint8_t BGR[8][3],int player,uint8_t move,int row)
{
    unsigned char b,r;
    unsigned int count=0;//counter

    for(b = 0x80;b>0x01;b=b>>1) //for each bit in b from left to right
    {
        if(BGR[row][player]&b)//chech whether lit in this player colour
        {
            count++;//counter++
            if(count>=4)
                return 1;//won
        }
        else
            count=0;//continues
    }
```

```
//Vertical Test
if (row<=4)    //check whether has enough room below
{
   if ((BGR[row+1][player]&move)&&          //check row+1
         (BGR[row+2][player]&move)&&          //check row+2
         (BGR[row+3][player]&move))          //check row+3
         return 1;
}

//Diagonal Test bottom left to top right
//find bottom left start pos
b = move;
r = row;
while (b<0x80 && r<7)//build the borders
{
    b<<=1;//reaching to bottom left
    r++;
}

//count 4
count=0;//count from 0
while(b>0x01 && r>1)//bulid the borders
{
    if(BGR[r][player]&b) //check whether a light LED in this player colour
    {
        count++;
        if (count==4)//won
            return 1;
    }
    else
```

```
            count=0;//count from 0 over again

        b>>=1;//reaching to top right
        r--;
    }

    //Diagonal Test bottom right to top left
    //find bottom right start pos
    b = move;
    r = row;
    while (b>0x02 && r<7)//bulid the borders
    {
        b>>=1;//reaching to bottom right
        r++;
    }
    //count 4
    count=0;
    while(b!=0x00 && r>1)//bulid the borders
    {
        if(BGR[r][player]&b) //check whether a light LED in this player colour
        {
            count++;
            if (count==4)
                return 1;//won
        }
        else
            count=0;//count from 0 over again

        b<<=1;//reaching to top left
        r--;}
```

9. Other Game Program

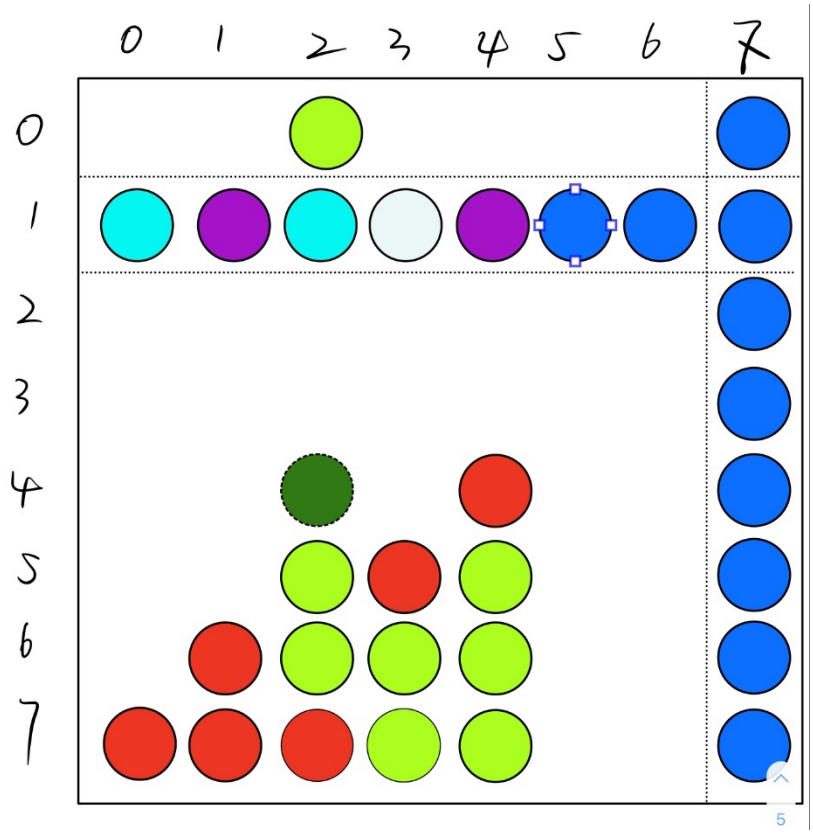

Figure 16 Features

Record the winner of each game and overlay the winning colour on the row[1] of the next round. The hearts of the winning colours are displayed until the end of the eight rounds.

Blue + red = cyan. Blue + red = purple. In a draw, blue + red + green = white.

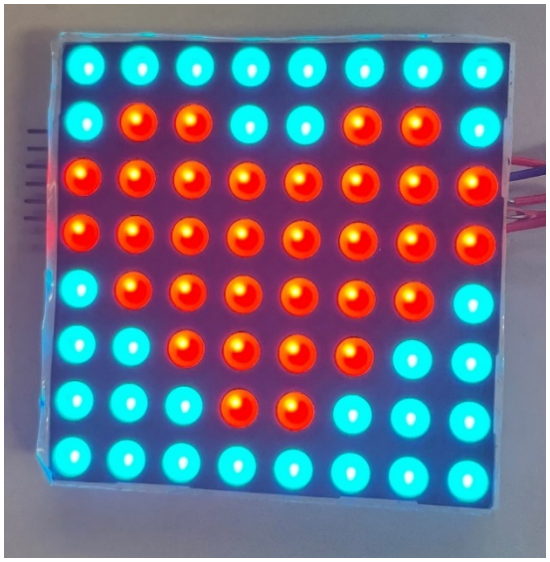

Figure 17 Matrix of Result

```
do{
    …
}while(numMoves<42&&!isWinner);//loops until draw or someone won

        if(isWinner)//if some player win
            store[player] |= 0x80>>i;//which one won, moving the LED with
which one's colour
        else
        {
            store[RED] |= 0x80>>i;//it is draw, move a white LED combined with
green, red, blue
            store[GREEN] |= 0x80>>i;
        }

        displayWinner(isWinner,player);//display the figure of the winner
```

10. Environment and Ethics

The highlight of it is its ultra-low power consumption. Ultra-low-power means that this board has a low-power mode, after entering the low-power mode it can standby for a long time with very little consumption, waking up the CPU for work at regular intervals, which is necessary to save energy when there are energy sources.

Environment requirements:

The efficient design of circuits thus minimizing components;
I initiate the design of PCB boards to be as small as possible.
In the design and production of projects keep power use to a minimum, by switching off unused equipment when appropriate;
All electronic/electrical waste be disposed of correctly.

Ethical requirements:

Maintain a clean work environment by designing software that is efficient and avoids wasting computing resources. Understand how to utilize equipment correctly; WEEE standard
Any work should be free of plagiarism, and all help should be properly acknowledged.
The Comreg standard is used in telecommunications.
Copyright and software licensing should be honoured.

11. Conclusion&Discussion

This project has given me a systematic grasp of the process of learning to do a project. I was confused about the logical relationship between the various sections at the beginning, so I should have a better grasp of the project as a whole in advance for the next study.

12. References

[1] TI Launchpad workshop
http://processors.wiki.ti.com/index.php/Getting_Started_with_the_MSP430G2553_Value- Line_LaunchPad_Workshop

[2] MSP430x2xx Family User's Guide

[3]MSP430 GPIO Programming Tutorial
http://www.ocfreaks.com/msp430-gpio-programming-tutorial/

MOD-LED8x8RGB
[4]https://olimex.wordpress.com/2013/06/21/new-product-mod-led8x8rgb-stackable-rgb-led-matrix/

[5]https://www.olimex.com/Products/Modules/LED/MOD-LED8x8RGB/open-source-hardware

[6]https://github.com/mchobby/esp8266-upy/blob/master/modled8x8/readme_ENG.md

[7] http://learncontrollers.blogspot.com/

[8] Myke Predkos' PICMicro Button Debounce
Article http://www.rentron.com/Myke6.htm

13. Appendix

```
/*
 * main.c
 */
#include <msp430.h>
#include <stdint.h>
#include "mtxdisp.h"
#include "placeMove.h"
#include "switch.h"
#include "playerWon.h"
#include "displayWinner.h"

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD;// Stop watchdog timer
	initialiseSwitch();//set up switch for reading
	initMtx();//init switches and matrix etc

	uint8_t store[3]={0xff,0x00,0x00};
	int i;
	for(i=0;i<8;i++)//counter for counting 8 rounds of a game
	{
		uint8_t BGR[8][3]={	{0x01,0x00,0x00},//Original LEDs
					{0xFF,0x00,0x00},
					{0x01,0x00,0x00},
					{0x01,0x00,0x00},
					{0x01,0x00,0x00},
					{0x01,0x00,0x00},
```

```
                                {0x01,0x00,0x00},
                                {0x01,0x00,0x00}};

        clearMtx();
        lightMtx(BGR);

        BGR[1][BLUE] = store[BLUE];
        BGR[1][GREEN] = store[GREEN];
        BGR[1][RED] = store[RED];

        int player = RED;
        int numMoves = 0;//counter for counting moves number
        int row;//row of the matrix
        uint8_t move,isWinner;

        do{
            if(player==RED)//alternate between red and green of players
                player= GREEN;
            else
                player = RED;

            move = getMove(BGR,player);//makes a move
            row= placeMove(BGR,player,move);//places a move

            numMoves++;//increment of the moves
            lightMtx(BGR);//lights the LED
            isWinner = playerWon(BGR,player,move,row);//Judges whether
someone won

        }while(numMoves<42&&!isWinner);//loops until draw or someone won
```

```
            if(isWinner)//if some player win
                store[player] |= 0x80>>i;//which one won, moving the LED with
which one's colour
            else
            {
                store[RED] |= 0x80>>i;//it is draw, move a white LED combined with
green, red, blue
                store[GREEN] |= 0x80>>i;
            }

            displayWinner(isWinner,player);//display the figure of the winner

        }
        return 0;
    }

/*
 * mtxdisp.c
 */
#include <stdint.h>
#include "mtxdisp.h"

void initMtx(void)
{
    MTXDIR |= MTXDATA + MTXCLK + MTXLATCH; ////Makes Data,Latch and
Clk pins outputs
    MTXOUT &= ~(MTXDATA + MTXCLK);//Makes Data and Clk low
    MTXOUT |= MTXLATCH;    //Make Latch high as it is active low
}
```

```
void clearMtx(void)
{//clocks in 8*3*8(192) lows to matrix (Clears matrix)

    int x;
    MTXOUT &= ~MTXLATCH;    //Latch low
    MTXOUT &= ~MTXDATA;     //Data low

    for (x=1; x<=192;x++)
    {
        MTXOUT |= MTXCLK; //Clk high
        MTXOUT &= ~MTXCLK;//Clk low
    }

    MTXOUT |= MTXLATCH; //Latch high
}

void lightMtx(const uint8_t mtxBuf[8][3])
{// displays mtxBuf values on matrix displays

    int x,colour,row;
    uint8_t val;

    MTXOUT &= ~MTXLATCH;    //Latch low

    for(row=7;row>=0;row--)    //for each row from 7 downto 0
    {
        for(colour=2;colour>=0; colour--)
        {//for each colour from 2 downto 0
            val = mtxBuf[row][colour];
```

```
            for (x=1; x<=8;x++)    //for each bit in val shift it out
            {
                if (val&BIT0) // is LSb of val high
                    MTXOUT |= MTXDATA; //Data high
                else
                    MTXOUT &= ~MTXDATA; //Data low

                MTXOUT |= MTXCLK; //Clk high
                MTXOUT &= ~MTXCLK;//Clk low
                val>>=1;
            }
        }
    }
    MTXOUT |= MTXLATCH; //Latch high
}


/*
 * mtxdisp.h
 *
 *    Created on: 27 Oct 2016
 *          Author: mgill1
 */

#ifndef MTXDISP_H
#define MTXDISP_H

#include <msp430.h>
#include <stdint.h>
```

```
#define MTXDIR P1DIR
#define MTXOUT P1OUT
#define MTXDATA BIT7
#define MTXCLK BIT5
#define MTXLATCH BIT4

void initMtx(void);
void clearMtx(void);
void lightMtx(const uint8_t mtxBuf[8][3]);

#endif /* MTXDISP_H */



/*
 * Move.c
 */
#include <placeMove.h>
#include "mtxdisp.h"
#include "switch.h"

uint8_t getMove( uint8_t BGR[8][3],int player)
{
    unsigned char val,litLEDs;//val is the status of switch
    volatile long x;
    uint8_t ch=0x02;//start at rightmost position

    do{
        if(ch>0x02)//loop in the fist row from left to right
            ch = ch >> 1;//swift to the right
```

```
            else
                ch=0x80;//back to the left

            BGR[0][player]=ch;
            lightMtx(BGR);//light the LED

            for(x=0;x<20000;x++)
                ;//delay

            val=readSwitchDB();//checks the switch status
            litLEDs = BGR[2][GREEN]|BGR[2][RED];//check the row 2 position is
not occupied

        }while(val!=DOWN||(litLEDs&ch)!=0);//loop while the switch not down or this
position corresponding the position on row 2 is not occupied

        BGR[0][player]=0x00;//clear all on row 0
        return ch;
}

int placeMove(uint8_t BGR[8][3],int player,uint8_t Move)
{
        uint8_t litLeds;
        int row = 7;
        litLeds = BGR[7][RED]|BGR[7][GREEN];//gets the bottom position

        while((litLeds&Move)!=0)//loop until find the empty position
        {
            row--;//check upper one
```

```
            litLeds = BGR[row][RED]|BGR[row][GREEN];//get the lowest empty
position
        }

        BGR[row][player]|=Move;//add move to the right position
        return row;
}



/*
 * Move.h
 */
#ifndef PLACEMOVE_H_
#define PLACEMOVE_H_

#include <stdint.h>
#define BLUE 0
#define GREEN 1
#define RED 2

uint8_t getMove( uint8_t BGR[8][3],int player);

int placeMove(uint8_t BGR[8][3],int player,uint8_t Move);

#endif



/*
 *playerWon.c
 */
```

```
#include "mtxdisp.h"
#include "switch.h"

//horizontal
int playerWon( uint8_t BGR[8][3],int player,uint8_t move,int row)
{
	unsigned char b,r;
	unsigned int count=0;//counter

	for(b = 0x80;b>0x01;b=b>>1) //for each bit in b from left to right
	{
		if(BGR[row][player]&b)//chech whether lit in this player colour
		{
			count++;//counter++
			if(count>=4)
				return 1;//won
		}
		else
			count=0;//continues
	}

	//Vertical Test
	if (row<=4)    //check whether has enough room below
	{
	   if ((BGR[row+1][player]&move)&&          //check row+1
		(BGR[row+2][player]&move)&&          //check row+2
		(BGR[row+3][player]&move))          //check row+3
		return 1;
	}
```

```
//Diagonal Test bottom left to top right

//find bottom left start pos
b = move;
r = row;
while (b<0x80 && r<7)//build the borders
{
    b<<=1;//reaching to bottom left
    r++;
}

//count 4
count=0;//count from 0
while(b>0x01 && r>1)//bulid the borders
{
    if(BGR[r][player]&b) //check whether a light LED in this player colour
    {
        count++;
        if (count==4)//won
            return 1;
    }
    else
        count=0;//count from 0 over again

    b>>=1;//reaching to top right
    r--;
}

//Diagonal Test bottom right to top left
```

```
    //find bottom right start pos
    b = move;
    r = row;
    while (b>0x02 && r<7)//bulid the borders
    {
        b>>=1;//reaching to bottom right
        r++;
    }

    //count 4
    count=0;
    while(b!=0x00 && r>1)//bulid the borders
    {
        if(BGR[r][player]&b) //check whether a light LED in this player colour
        {
            count++;
            if (count==4)
                return 1;//won
        }
        else
            count=0;//count from 0 over again

        b<<=1;//reaching to top left
        r--;
    }
    return 0;
}


/*
```

```
 * playerWon.h
 */
#ifndef PLACEMOVE_H_
#define PLACEMOVE_H_

#include <stdint.h>

int playerWon(uint8_t BGR[8][3],int player,uint8_t move,int row);

#endif

/*
 * switch.c
 */
#include "switch.h"
#include "delay.h"
#include <msp430.h>
unsigned char readSwitchPin(void);

void initialiseSwitch(void)
{
    //put comments here
    P1DIR &= ~BIT3; // Clear bit 3 making P1_3 an input i.e ~BIT3 (11110111
BIN)
    P1OUT |= BIT3; // Set bit 3 of P1OUT to select pullup i.e BIT3(00001000BIN)
    P1REN |= BIT3; // Enable the pullup resister i.e BIT3 (00001000 BIN)
}

unsigned char readSwitchPin(void)
{
```

```
    //put comments here
    if( P1IN & BIT3 ) //BIT3 (00001000 BIN)
        return UP; //pin was high (Switch un-pressed)
    else
        return DOWN; //pin was low (Switch pressed)
}

unsigned char readSwitchDB(void)
{
    // put comments here
    static unsigned char lastRead=UP; // assume switch is un-pressed at startup
    unsigned char val;
    val=readSwitchPin(); //read switch

    if( val != lastRead ) //check if it has changed
    {
        delay(200); //wait for 20ms
        val=readSwitchPin(); //read switch

        if(val != lastRead ) //check switch again to make sure it has
        { // really changed
            lastRead = val; //update lastRead
            return val;
        }
    }
    return SAME;
}
```

```
/*
 * switch.h
 *
 *    Created on: 18 Nov 2021
 *           Author: mgill
 */

#ifndef SWITCH_H_
#define SWITCH_H_

#define UP 1
#define DOWN 0
#define SAME 2

void initialiseSwitch(void);
unsigned char readSwitchDB(void);

#endif /* SWITCH_H_ */
```