Assignment 2: Lab Report

Digital Design Lab

Sarah Brown

I. INTRODUCTION

A. Overview

This assignment focuses on designing a circuit with LEDs and switches to allow players to play a game of Tic-Tac-Toe. This circuit is integrated with the LPC1769 using its I/O subsystem to interact with the LEDs and switches. This assignment assumes that there are two players for simplicity.

B. Objectives

The objective of this assignment is to use a 3-by-3 grid of 9 bi-color LEDs to create a Tic-Tac-Toe board. This grid should be controlled by 9 switches. After there are no more moves available or a player has won, the game pauses and then restarts the game. In addition, 3 yellow LEDs and 3 green LEDs are used to keep score for player 1 (yellow) and player 2 (green).

In addition, the LEDs should have a maximum brightness without exceeding any of the current limits of the devices. The current measured on the USB cable should be under 100 mA limit for an USB spec.

C. Demonstratables

Show a game for each player winning and that the game ends and show another game where the game ends in a tie. Since I included the score LEDs, I also demonstrated the score LEDs resetting when one of the players gets to 3 wins.

II. SOLUTION DESIGN

A. Component Value Derivation

The high-level output current for the LPC1769D board specifies V_{DD} = 3.3 V – 0.4 V when the current is at least 4 mA. Therefore, the logic high can be estimated to be 2.9 V. The high-level short-circuit output current is I_{OHS} = 45 mA while not exceeding the overall current limits. Therefore, each pin can have a maximum of 45 mA. The following calculations are to ensure LEDs are within their safety limits. The green and yellow LEDs were measured by hand with a power supply to find a good current level and brightness balance.

LED Type	Forward Voltage	Peak Forward Current
Bi-Color Yellow	2.1 V	20 mA
Bi-Color Green	2.1 V	30 mA
Green	2.37 V	1 mA
Yellow	1.92 V	2 mA

Table 1: Voltage and current specs for LED parts

$$V = IR$$
$$V = V_{out} - V_{forward}$$
$$I = \frac{V_{out} - V_{forward}}{R}$$

Bi-Color Yellow LED:

$$20\ mA = \frac{2.9\ V - 2.1\ V}{R}$$
$$R = 40\ \Omega$$

This value is then rounded to 100 Ω for safety and due to values in lab kit. Testing the change in brightness with a 68 Ω versus an 100 Ω resistor had no noticeable change in brightness. The current is then recalculated with this value.

$$I_{470\Omega} = \frac{2.9\ V - 2.1\ V}{40\Omega}$$

$$I_{40\Omega} = 8\ mA$$

Bi-Color Green LED:

$$30\ mA = \frac{2.9\ V - 2.1\ V}{R}$$

R = 27Ω

This value is then rounded to 100 Ω for safety and due to values in lab kit. This does not result in noticeable brightness loss. The current is then recalculated with this value.

$$I_{100\Omega} = \frac{2.9\ V - 2.1\ V}{100\Omega}$$

$$I_{100\Omega} = 8\ mA$$

Green LED:

$$1\ mA = \frac{2.9\ V - 2.37\ V}{R}$$

R = 530 Ω

This value is then rounded to 560 Ω for safety and due to values in lab kit. Due to the LEDs in use, this does not result in noticeable brightness loss. The current is then recalculated with this value.

$$I_{560\Omega} = \frac{2.9\ V - 2.37\ V}{530\Omega}$$

$$I_{560\Omega} = 1\ mA$$

Yellow LED:

$$2\ mA = \frac{2.9\ V - 1.92\ V}{R}$$

R = 490 Ω

This value is then rounded to 560 Ω for safety and due to values in lab kit. Due to the LEDs in use, this does not result in noticeable brightness loss. The current is then recalculated with this value.

$$I_{560\Omega} = \frac{2.9\ V - 1.92\ V}{560\Omega}$$

$$I_{560\Omega} = 1.75\ mA$$

LED Type	Calculated Resistor Value	Chosen Resistor Value	Expected Current
Bi-Color Yellow	40 Ω	100 Ω	8 mA
Bi-Color Green	27 Ω	100 Ω	8 mA
Green	530 Ω	560 Ω	1 mA
Yellow	490 Ω	560 Ω	1.75 mA

Table 2: Calculated resistor values and the expected current with chosen resistors

The total current when all LEDs are turned on can then be calculated by summation.

$$I_{total} = 4(I_{biY}) + 4(I_{biG}) + 4(I_{blue}) + 4(I_{yellow})$$

$$I_{total} = 4(8\ mA) + 4(8\ mA) + 4(1\ mA) + 4(1.75\text{mA})$$

$$I_{total} = 32\ mA + 32mA + 5\ mA + 7\ \text{mA}$$

$$I_{total} = 75\ mA$$

Therefore, the expected current for running all 24 LEDs at once is 75mA.

B. Current Limitations

By using a RuiDeng USB Voltage Tester, it is possible to determine that the passive current draw of the LPC1769 board is about 70 mA. As the hardware requirements state that the overall current draw remain below the current limit on an USB spec (100 mA for a default state), this leaves roughly 30 mA for LED use. As turning all 24 LEDs on at the same time has an expected current of 75 mA, it will be necessary to ensure not all of the LEDs are on at the same time. Therefore, the code should ensure that only one or two LEDs are on at any one time. This can be simplified by assuming that one Bi-Color LED cannot have both the yellow and

the green parts on at the same time due to the aspects of a Tic-Tac-Toe game. Therefore, it is possible to cycle through the different score LEDs and Tic-Tac-Toe grid LEDs and stay within the current limits. While it is possible to stay within these limits, the circuit could be improved by selecting different bi-color LEDs that have increased luminosity for lower current requirements.

C. *Software Design and Primary Functions* ***explain the major functions and variables used as well as the relevant registers used to interface to the hardware. This should just be a general overview.***

The software uses register ports 0, 1, and 2 (using almost all of the I/O pins). These I/O pins are used to read in and output data to the Tic-Tac-Toe circuit. There are several essential variables used throughout the code. A structure is created to store register information, whether it is an input or an output, and the pin number. Using this structure, 4 pin_details arrays are created to store information on the yellow bi-color LEDs, the green bi-color LEDs, the switches, and the score LEDs. With the use of other variables used to store the state of the game throughout the code and the current LED information, it is possible to use a series of if-statements in the function grid_turn to control the game logic.

The change_leds function is used to cycle through the LEDs that should be on at any given time. By cycling through these LEDs with an integer count, it is possible to limit the current to meet the specifications while maintaining the brightness of the LEDs.

The only_one function is implemented with the change_leds function to ensure that only the LEDs matching the led_count are on at any given time.

The read_switches function is implemented to update an array with which buttons are pressed. Due to the dynamics of a Tic-Tac-Toe game, button debouncing is not required as it does not matter if a button is accidentally pressed more than once, a grid square can only be occupied by player 1 or player 2. However, if button debouncing was essential it would be added to this function with a time delay to compare two different time segments or by searching for a rising/falling edge depending on if the logic was logic-low or logic-high.

The check_score function compares the values stored in the grid array (0 for an empty spot, 1 for player 1, and 2 for player 2) and determines if a win or draw condition is met. It then updates the results value with the new score. The results value is then used to update the score LEDs.

There are several other smaller functions used for abstraction and to simplify the main loop, including functions used to reset the grid and score LEDs. These are documented in the code appendix below, but are not mentioned here to keep the software design overview to a general overview.

III. SOLUTION DETAILS

A. *Schematic*

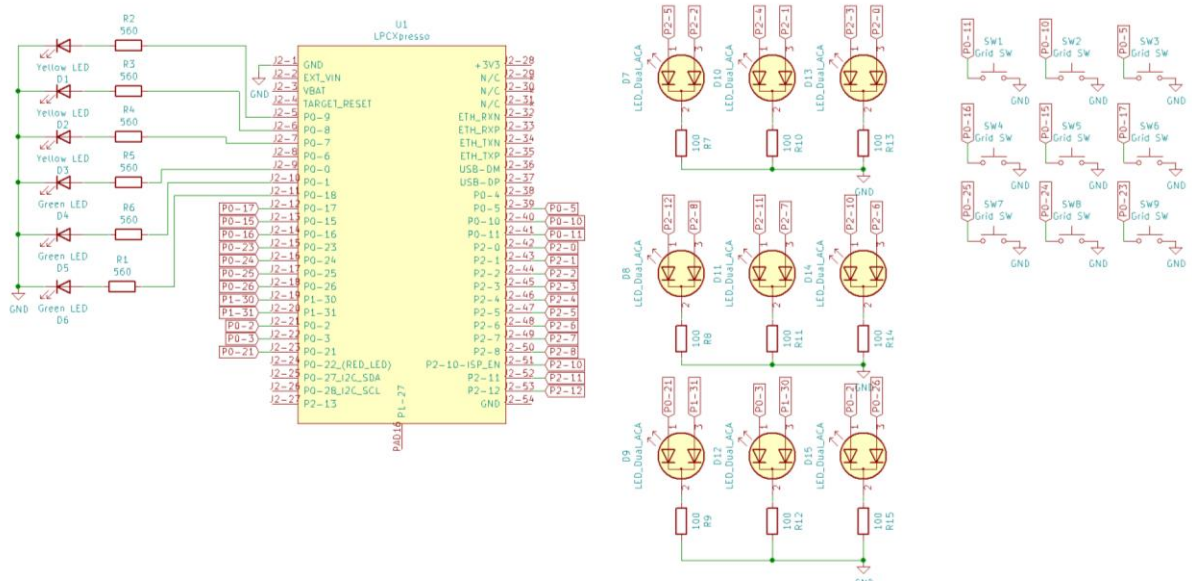

Figure 1: Schematic of hardware design

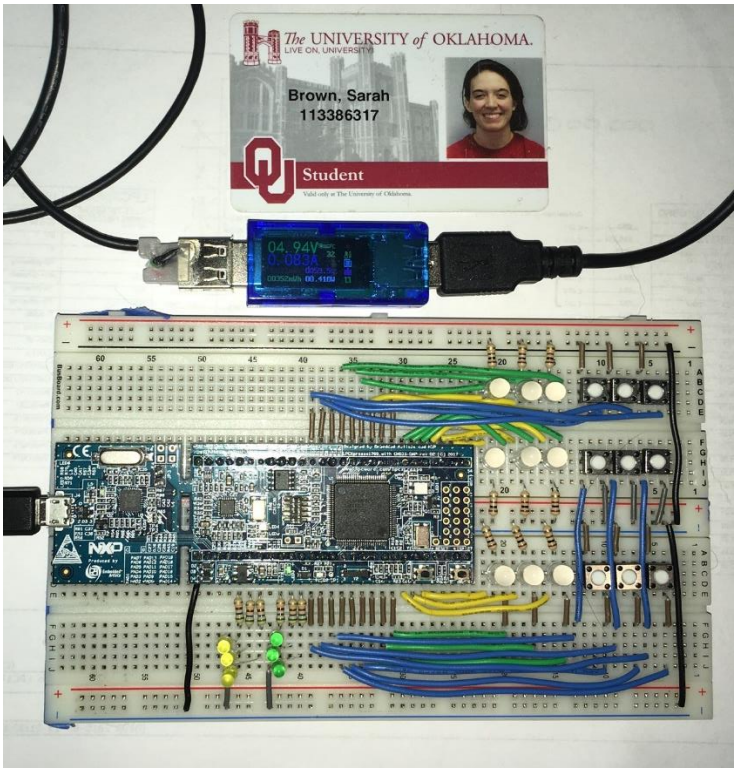

Figure 2: Breadboarded circuit

B. Code

See appendix for code.

IV. CONCLUSION

This assignment served as an overview of interfacing a circuit with the LPC1769's general purpose I/O subsystem. It was possible to design a circuit and write code so that a Tic-Tac-Toe game could be played while meeting certain hardware requirements. However, the bi-color LEDs used had low luminosity values for high current requirements. Therefore, different LEDs could increase the brightness of the Tic-Tac-Toe grid.

Appendix

```
/*
===============================================================================
 Name        : LabAssignment2.c
 Description : main definition
===============================================================================
*/

// register definitions
#define FIO0DIR ((volatile unsigned int *)0x2009c000)
#define FIO0PIN ((volatile unsigned int *)0x2009c014)
#define FIO1DIR ((volatile unsigned int *)0x2009c020)
#define FIO1PIN ((volatile unsigned int *)0x2009c034)
#define FIO2DIR ((volatile unsigned int *)0x2009c040)
#define FIO2PIN ((volatile unsigned int *)0x2009c054)

#define NUM_GRID 9 // number of squares in tic-tac-toe grid
#define NUM_SCORE 6 //num of score leds

#define TRUE 1
#define FALSE 0
#define PRESSED 0 // active low

// active high leds. starting states all off
unsigned int yellow_leds_state = 0b000000000;
unsigned int green_leds_state = 0b000000000;
unsigned int score_leds_state = 0b000000;

unsigned int grid[9] = {0,0,0,0,0,0,0,0,0}; // store state of grid. 0 - off. 1 - p1. 2 - p2
unsigned int results = 0; // 0 - game in progress, 1 - p1 won, 2 - p2 won, 3 - draw
unsigned int yellow_wins = 0;
unsigned int green_wins = 0;

unsigned int p1_turn = TRUE;
unsigned int grid0_lock = FALSE; unsigned int grid1_lock = FALSE; unsigned int grid2_lock = FALSE; unsigned int grid3_lock = FALSE;
unsigned int grid4_lock = FALSE; unsigned int grid5_lock = FALSE; unsigned int grid6_lock = FALSE; unsigned int grid7_lock = FALSE;
unsigned int grid8_lock = FALSE;
int led_count = 0; // to switch which led is on

/*
 * Structure to store details about pin groups
 */
typedef struct {
         volatile unsigned int * direct_reg;
         volatile unsigned int * pin_reg;
         const unsigned int is_output; // boolean if pin is input or output
         const unsigned char pin_num; // pin number
} pin_details;

pin_details grid_leds_yellow[9] = {
                  {FIO2DIR, FIO2PIN, TRUE, 5}, {FIO2DIR, FIO2PIN, TRUE, 4}, {FIO2DIR, FIO2PIN, TRUE, 3}, {FIO2DIR, FIO2PIN, TRUE, 12},
{FIO2DIR, FIO2PIN, TRUE, 11}, {FIO2DIR, FIO2PIN, TRUE, 10}, {FIO0DIR, FIO0PIN, TRUE, 21}, {FIO0DIR, FIO0PIN, TRUE, 3}, {FIO0DIR,
FIO0PIN, TRUE, 2}
};

pin_details grid_leds_green[9] = {
                  {FIO2DIR, FIO2PIN, TRUE, 2}, {FIO2DIR, FIO2PIN, TRUE, 1}, {FIO2DIR, FIO2PIN, TRUE, 0}, {FIO2DIR, FIO2PIN, TRUE, 8},
{FIO2DIR, FIO2PIN, TRUE, 7}, {FIO2DIR, FIO2PIN, TRUE, 6}, {FIO1DIR, FIO1PIN, TRUE, 31}, {FIO1DIR, FIO1PIN, TRUE, 30}, {FIO0DIR,
FIO0PIN, TRUE, 26}
};

pin_details grid_switches[9] = {
                  {FIO0DIR, FIO0PIN, FALSE, 11}, {FIO0DIR, FIO0PIN, FALSE, 10}, {FIO0DIR, FIO0PIN, FALSE, 5}, {FIO0DIR, FIO0PIN, FALSE,
16}, {FIO0DIR, FIO0PIN, FALSE, 15}, {FIO0DIR, FIO0PIN, FALSE, 17}, {FIO0DIR, FIO0PIN, FALSE, 25}, {FIO0DIR, FIO0PIN, FALSE, 24},
{FIO0DIR, FIO0PIN, FALSE, 23}
};

pin_details score_leds[6] = {
                  {FIO0DIR, FIO0PIN, TRUE, 18}, {FIO0DIR, FIO0PIN, TRUE, 1}, {FIO0DIR, FIO0PIN, TRUE, 0}, {FIO0DIR, FIO0PIN, TRUE, 7},
{FIO0DIR, FIO0PIN, TRUE, 8}, {FIO0DIR, FIO0PIN, TRUE, 9}
};

/*
 * Function to waste time
 *
 * @param ticks - x variable in function
 * y = 2.73x + 8.5. units: y - usec, x - ticks
 */
void wait_ticks(int ticks) {
```

```
        volatile int count;
        for (count=0; count<ticks; count++) {
                //do nothing
        }
}

/*
 * Function to configure pins as inputs/outputs
 *
 * @param pin_group[] - array of pins
 * @param num_pins - how many pins are in the group
 */
void config_port(pin_details pin_group[], int num_pins) {
        for (int i = 0; i < num_pins; i++) {
                        if (pin_group[i].is_output) {
                                *(pin_group[i].direct_reg) |= (1<<pin_group[i].pin_num);
                        }

                        else {
                                *(pin_group[i].direct_reg) &= ~(1<<pin_group[i].pin_num);
                        }
                }
}

/*
 * Function to turn off a group of leds
 *
 * @param pin_group[] - array of pins
 * @param num_pins - how many pins are in the group
 */
void turn_off_leds(pin_details pin_group[], int num_pins) {
        for (int i = 0; i < num_pins; i++) {
                        *(pin_group[i].pin_reg) &= ~(1<<pin_group[i].pin_num);
        }
}

/*
 * Function to one toggle led in a pin group to on/off
 *
 * @param pin_group[] - array of pins
 * @param led_num - stores which led num in array is being toggled
 * @param is_high - boolean value stores whether led is on or offf
 */
void change_led(pin_details pin_group[], int led_num, int is_high) {
        if (is_high) {
                        *pin_group[led_num].pin_reg |= (1<<pin_group[led_num].pin_num);
        }

        else {
                        *pin_group[led_num].pin_reg &= ~(1<<pin_group[led_num].pin_num);
        }
}

/*
 * Debounce function
 *
 * @param pin_group[] - array of pins
 * @param switches - pointer to array of led values
 * @param num_switches - number of switches to loop through
 */
void read_switches(pin_details pin_group[], unsigned int * switches, int num_switches) {
        unsigned int state_one[num_switches];
        unsigned int state_two[num_switches];

        for (int i = 0; i < num_switches; i++) {
                if (!pin_group[i].is_output) {
                        state_one[i] = *pin_group[i].pin_reg & (1 << pin_group[i].pin_num);
                }
        }

        //wait_ticks(50); //tic tac toe doesnt need debounce, but wait delay would go here if it did

        for (int i = 0; i < num_switches; i++) {
                if (!pin_group[i].is_output) {
                        state_two[i] = *pin_group[i].pin_reg & (1 << pin_group[i].pin_num);
                }
        }

        for (int i = 0; i < num_switches; i++) {
                if (!pin_group[i].is_output) {
                        switches[i] = state_one[i] & state_two[i];
```

```
                }
        }
}

/*
 * Function to alternate which leds are on to limit current draw
 *
 * @param led_count - current loop count (1-9)
 * @param pin_group[] - array of pins
 * @param is_high - boolean whether led is high
 */
void only_one(int led_count, int which_led, pin_details pin_group[], int led_num, int is_high) {
        if (led_count == which_led) {
                change_led(pin_group, led_num, is_high);
        }

        else {
                change_led(pin_group, led_num, FALSE);
        }
}

/*
 * Function to loop through and only turn on certain leds at once
 *
 * @param led_count - which group of leds should be on
 */
void change_leds(int led_count) {
        only_one(led_count,  0, grid_leds_yellow, 0, yellow_leds_state & (1 << 0));
        only_one(led_count,  1, grid_leds_yellow, 1, yellow_leds_state & (1 << 1));
        only_one(led_count,  2, grid_leds_yellow, 2, yellow_leds_state & (1 << 2));
        only_one(led_count,  3, grid_leds_yellow, 3, yellow_leds_state & (1 << 3));
        only_one(led_count,  4, grid_leds_yellow, 4, yellow_leds_state & (1 << 4));
        only_one(led_count,  5, grid_leds_yellow, 5, yellow_leds_state & (1 << 5));
        only_one(led_count,  6, grid_leds_yellow, 6, yellow_leds_state & (1 << 6));
        only_one(led_count,  7, grid_leds_yellow, 7, yellow_leds_state & (1 << 7));
        only_one(led_count,  8, grid_leds_yellow, 8, yellow_leds_state & (1 << 8));
        only_one(led_count,  0, grid_leds_green, 0, green_leds_state & (1 << 0));
        only_one(led_count,  1, grid_leds_green, 1, green_leds_state & (1 << 1));
        only_one(led_count,  2, grid_leds_green, 2, green_leds_state & (1 << 2));
        only_one(led_count,  3, grid_leds_green, 3, green_leds_state & (1 << 3));
        only_one(led_count,  4, grid_leds_green, 4, green_leds_state & (1 << 4));
        only_one(led_count,  5, grid_leds_green, 5, green_leds_state & (1 << 5));
        only_one(led_count,  6, grid_leds_green, 6, green_leds_state & (1 << 6));
        only_one(led_count,  7, grid_leds_green, 7, green_leds_state & (1 << 7));
        only_one(led_count,  8, grid_leds_green, 8, green_leds_state & (1 << 8));
        only_one(led_count,  0, score_leds, 0, score_leds_state & (1 << 0));
        only_one(led_count,  1, score_leds, 1, score_leds_state & (1 << 1));
        only_one(led_count,  2, score_leds, 2, score_leds_state & (1 << 2));
        only_one(led_count,  3, score_leds, 3, score_leds_state & (1 << 3));
        only_one(led_count,  4, score_leds, 4, score_leds_state & (1 << 4));
        only_one(led_count,  5, score_leds, 5, score_leds_state & (1 << 5));
}

/*
 * Function to pause board in between rounds and in between games
 *
 * @param ticks - how long to pause for
 * y = 2.73x + 8.5. units: y - usec, x - ticks
 */
void pause(int ticks) {
        volatile int count;
        int led_count = 0;
        for (count=0; count<ticks; count++) {
                change_leds(led_count);
                led_count++;
                if (led_count > 9) {
                        led_count = 0;
                }
        }
}

/*
 * Function to handle game turns and keep track of grid variables
 *
 * @param switch_press - whether a switch is pressed
 * @param grid_num - which grid number
 */
void grid_turn(unsigned int switch_press, unsigned int grid_num) {
        if (grid_num == 0) {
                if(switch_press == PRESSED && !grid0_lock) {
                        if (p1_turn) {
```

```
                        yellow_leds_state |= (1 << grid_num);
                        grid[grid_num] = 1;
                }

                else {
                        green_leds_state |= (1 << grid_num);
                        grid[grid_num] = 2;
                }

                p1_turn = !p1_turn;
                grid0_lock = TRUE;
        }
}

else if (grid_num == 1) {
        if(switch_press == PRESSED && !grid1_lock) {
                if (p1_turn) {
                        yellow_leds_state |= (1 << grid_num);
                        grid[grid_num] = 1;
                }

                else {
                        green_leds_state |= (1 << grid_num);
                        grid[grid_num] = 2;
                }

                p1_turn = !p1_turn;
                grid1_lock = TRUE;
        }
}

else if (grid_num == 2) {
        if(switch_press == PRESSED && !grid2_lock) {
                if (p1_turn) {
                        yellow_leds_state |= (1 << grid_num);
                        grid[grid_num] = 1;
                }

                else {
                        green_leds_state |= (1 << grid_num);
                        grid[grid_num] = 2;
                }

                p1_turn = !p1_turn;
                grid2_lock = TRUE;
        }
}

else if (grid_num == 3) {
        if(switch_press == PRESSED && !grid3_lock) {
                if (p1_turn) {
                        yellow_leds_state |= (1 << grid_num);
                        grid[grid_num] = 1;
                }

                else {
                        green_leds_state |= (1 << grid_num);
                        grid[grid_num] = 2;
                }

                p1_turn = !p1_turn;
                grid3_lock = TRUE;
        }
}

else if (grid_num == 4) {
        if(switch_press == PRESSED && !grid4_lock) {
                if (p1_turn) {
                        yellow_leds_state |= (1 << grid_num);
                        grid[grid_num] = 1;
                }

                else {
                        green_leds_state |= (1 << grid_num);
                        grid[grid_num] = 2;
                }

                p1_turn = !p1_turn;
                grid4_lock = TRUE;
        }
}
```

```
        else if (grid_num == 5) {
                if(switch_press == PRESSED && !grid5_lock) {
                        if (p1_turn) {
                                yellow_leds_state |= (1 << grid_num);
                                grid[grid_num] = 1;
                        }

                        else {
                                green_leds_state |= (1 << grid_num);
                                grid[grid_num] = 2;
                        }

                        p1_turn = !p1_turn;
                        grid5_lock = TRUE;
                }
        }

        else if (grid_num == 6) {
                if(switch_press == PRESSED && !grid6_lock) {
                        if (p1_turn) {
                                yellow_leds_state |= (1 << grid_num);
                                grid[grid_num] = 1;
                        }

                        else {
                                green_leds_state |= (1 << grid_num);
                                grid[grid_num] = 2;
                        }

                        p1_turn = !p1_turn;
                        grid6_lock = TRUE;
                }
        }

        else if (grid_num == 7) {
                if(switch_press == PRESSED && !grid7_lock) {
                        if (p1_turn) {
                                yellow_leds_state |= (1 << grid_num);
                                grid[grid_num] = 1;
                        }

                        else {
                                green_leds_state |= (1 << grid_num);
                                grid[grid_num] = 2;
                        }

                        p1_turn = !p1_turn;
                        grid7_lock = TRUE;
                }
        }

        else if (grid_num == 8) {
                if(switch_press == PRESSED && !grid8_lock) {
                        if (p1_turn) {
                                yellow_leds_state |= (1 << grid_num);
                                grid[grid_num] = 1;
                        }

                        else {
                                green_leds_state |= (1 << grid_num);
                                grid[grid_num] = 2;
                        }

                        p1_turn = !p1_turn;
                        grid8_lock = TRUE;
                }
        }

}

/*
 *Function to check if game is over and update results
 */
void check_score() {
        if ((grid[0] == 1 && grid[1] == 1 && grid[2] == 1) || (grid[3] == 1 && grid[4] == 1 && grid[5] == 1)  || (grid[6] == 1 &&
grid[7] == 1 && grid[8] == 1)
                        || (grid[0] == 1 && grid[3] == 1 && grid[6] == 1) || (grid[1] == 1 && grid[4] == 1 && grid[7] == 1) ||
(grid[2] == 1 && grid[5] == 1 && grid[8] == 1)
                        || (grid[0] == 1 && grid[4] == 1 && grid[8] == 1) || (grid[2] == 1 && grid[4] == 1 && grid[6] == 1)) {
                results = 1;
```

```
        }

        else if ((grid[0] == 2 && grid[1] == 2 && grid[2] == 2) || (grid[3] == 2 && grid[4] == 2 && grid[5] == 2)  || (grid[6] == 2 &&
grid[7] == 2 && grid[8] == 2)
                        || (grid[0] == 2 && grid[3] == 2 && grid[6] == 2) || (grid[1] == 2 && grid[4] == 2 && grid[7] == 2) ||
(grid[2] == 2 && grid[5] == 2 && grid[8] == 2)
                        || (grid[0] == 2 && grid[4] == 2 && grid[8] == 2) || (grid[2] == 2 && grid[4] == 2 && grid[6] == 2)) {
                results = 2;
        }

        else if (grid[0] && grid[1] && grid[2] && grid[3] && grid[4] && grid[5] && grid[6] && grid[7] && grid[8]) {
                results = 3;
        }
}

/*
 * Function to reset grid in between game rounds
 */
void reset_grid() {
        pause(1000);
        for (int i = 0; i < NUM_GRID; i++) {
                grid[i] = 0;
        }

        results = 0; p1_turn = TRUE;
        grid0_lock = FALSE; grid1_lock = FALSE; grid2_lock = FALSE; grid3_lock = FALSE; grid4_lock = FALSE; grid5_lock = FALSE;
grid6_lock = FALSE; grid7_lock = FALSE; grid8_lock = FALSE;

        yellow_leds_state = 0b0;
        green_leds_state = 0b0;
}

/*
 * Function to reset game in between games
 */
void reset_game() {
        reset_grid();
        yellow_wins = 0;
        green_wins = 0;
        score_leds_state = 0b0;
}

/*
 * Function to update score leds
 */
void update_score_leds() {
        score_leds_state |= (((1<<green_wins)-1));
        score_leds_state |= (((1<<yellow_wins)-1) << 3);
}

/*
 * Function to setup pins and turn off all leds
 */
void setup() {
        // configure ports
        config_port(grid_leds_yellow, NUM_GRID);
        config_port(grid_leds_green, NUM_GRID);
        config_port(grid_switches, NUM_GRID);
        config_port(score_leds, NUM_SCORE);

        // turn all off all leds
        turn_off_leds(grid_leds_yellow, NUM_GRID);
        turn_off_leds(grid_leds_green, NUM_GRID);
        turn_off_leds(score_leds, NUM_SCORE);
}

int main(void) {
        setup();

        unsigned int switches[9] = {1,1,1,1,1,1,1,1,1};
//
    while(1) {
        change_leds(led_count);
        led_count++;
        read_switches(grid_switches, switches, NUM_GRID);

        grid_turn(switches[0], 0);
        grid_turn(switches[1], 1);
        grid_turn(switches[2], 2);
        grid_turn(switches[3], 3);
        grid_turn(switches[4], 4);
```

```
            grid_turn(switches[5], 5);
            grid_turn(switches[6], 6);
            grid_turn(switches[7], 7);
            grid_turn(switches[8], 8);
            check_score();

            // p1 (yellow) wins
            if (results == 1) {
                    yellow_wins++;
                    reset_grid();
            }

            // p2 (green) wins
            if (results == 2) {
                    green_wins++;
                    reset_grid();
            }

            // a draw, no winner
            if (results == 3) {
                    reset_grid();
            }

            update_score_leds();

            if (yellow_wins == 3 || green_wins == 3) {
                    reset_game();
            }

            if(led_count > 8) {
                    led_count = 0;
            }
    }
    return 0 ;
}
```