CE121 Final Project Report
Name: Wuyuan Chen
date: 12/6/16
section: Tu 10-12 pm

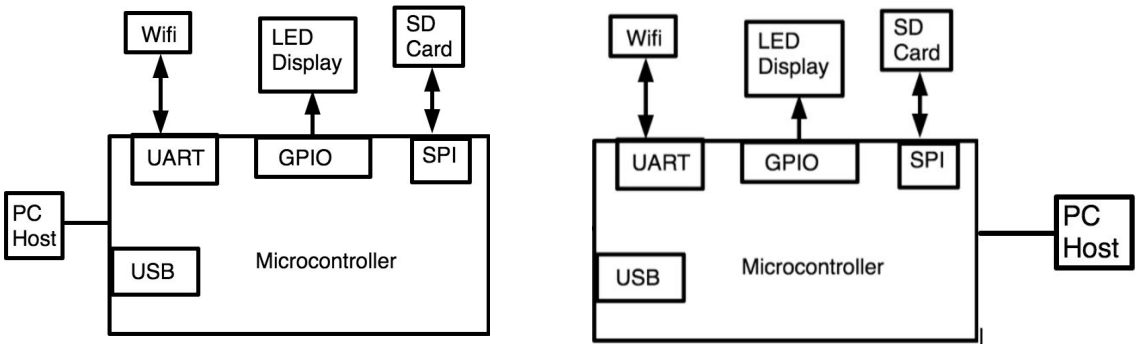
Purpose:
The purpose of this lab is to integrate all the skills learned from previous labs and use PSoC-5 microcontroller, the 16x32 RGB LED display, WiFi module, and a SD card interface to design an Internet connected two player game called Reversi(also known as Othello). This game will be able to perform on a 4x4, 8x8 and 16x16 board.

Introduction

The Game

Reversi is a two player board game played on an 8x8 board with black and white tiles. Each of player picks a color at the start of the game, whoever at the end has the most tiles on their side wins the game. The goal of the game is to turn your opponent's tiles into your own. Each time you place down a tile, you are trying to line it up with your own tiles on the other end, so you can flip your opponent's tile into yours. More in depth of the game could be found here.

System Design



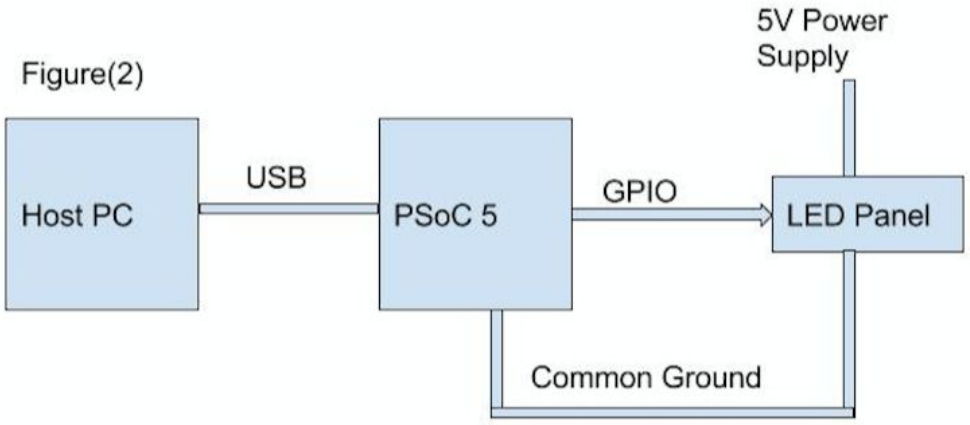
Above diagram shows player 1 and player 2's system high level diagram of the system. WiFi will be communicate with UART, LED Display will be driven by the GPIO, SD Card is interface with SPI and PC Host is the key board.

The numbering conversion for row and column is 1 and 1. While playing the game, both of the player should be updating their board constantly to update their board and display the same state of the game. In addition, the microcontroller must check the legality of each move, and winning conditions.

Hardware Design

Interfacing the LED Display

The LED panel needs a 5V power supply, and requires 2A when all LEDs are on. USB supply cannot supply that much power, so we need to connect the LED panel to the 5V power supply. Figure (2) shows the connecting.



Following steps show the procedure to take to connecting everything:

1. Set the PSoC-5 board to 5V by moving the jumpers(J10 and J11 to the left).
2. Connect to the GPIO pin by using the following diagram.
3. Turn on your board and load your program before turn on the 5V power supply.
4. Limit the output current to protect system overload.
5. Double check the polarity of power leads before turning on the power supply.
6. Turn off the power supply to the panel before turning the board off.

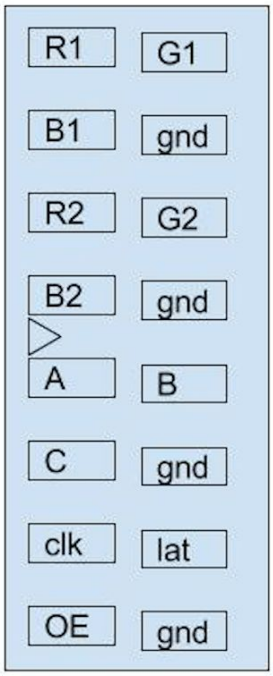
Control LED description:

The LED panel is divided into two sections, row 1-8 and row 9-16. The panel can only display one row in each section. Each row is displayed by selected inputs A, B and C. In order to display a pattern on the panel, it must cycle through the row 32 times. There are two sections, R1, G1, B1 and R2, G2, B2. Qe will be using R1, G1, B1 as the serial input used to shift the data for, red, green, blue LEDs respectively. In order to display 32 red LEDs in a row the LED must be shifted in one pair a time using the R1 serial input. The LED will seem to be in a constant on state if the clock is fast enough. Therefore the output an entire row of LEDs consist of following steps in an infinite loop:

1. All signals low, set OE high(to disable output).
2. Select first row.
3. Set your LED color.

4. Pulse clock 32 times.
5. Pulse latch once.
6. Set OE low (active low).
7. Select 2nd row. (constantly switch between row 1 and 2 to have make the panel active)

The same sequence applied to all color and you can display a total of 192 LEDs.



For the purpose of this lab, we will be connected to R1, R2, G2, R2,B2, G2 because we are working with the entire board, but we only need to display two different colors to differentiate player 1 and player 2. Figure to the left shows the Pins to connect to the PSoC5 board.

User Input

The user makes a move by pressing the keyboard, and it will select the corresponding row and column. Following shows the cursor movement:

- Left arrow: Move the cursor to the left by 1 column (decrease the counter)
- Right arrow: Move to the right by one column (increase the counter)
- Up arrow: Move up by one row (decrease the counter)
- Down arrow: Move down by one row (increase the counter)
- Home: Move the tile to the op row and leftmost column.
- Return Key: Execute the move and place the tile on the board.
- Type the letter S, followed by Return key, indicates that user wants to pass their turn.

SD Card

The SD card is used to record player moves, so that the game can be reconstructed later. Each move is to be recorded in the following format:
Player - ID pass Row-number Column-number

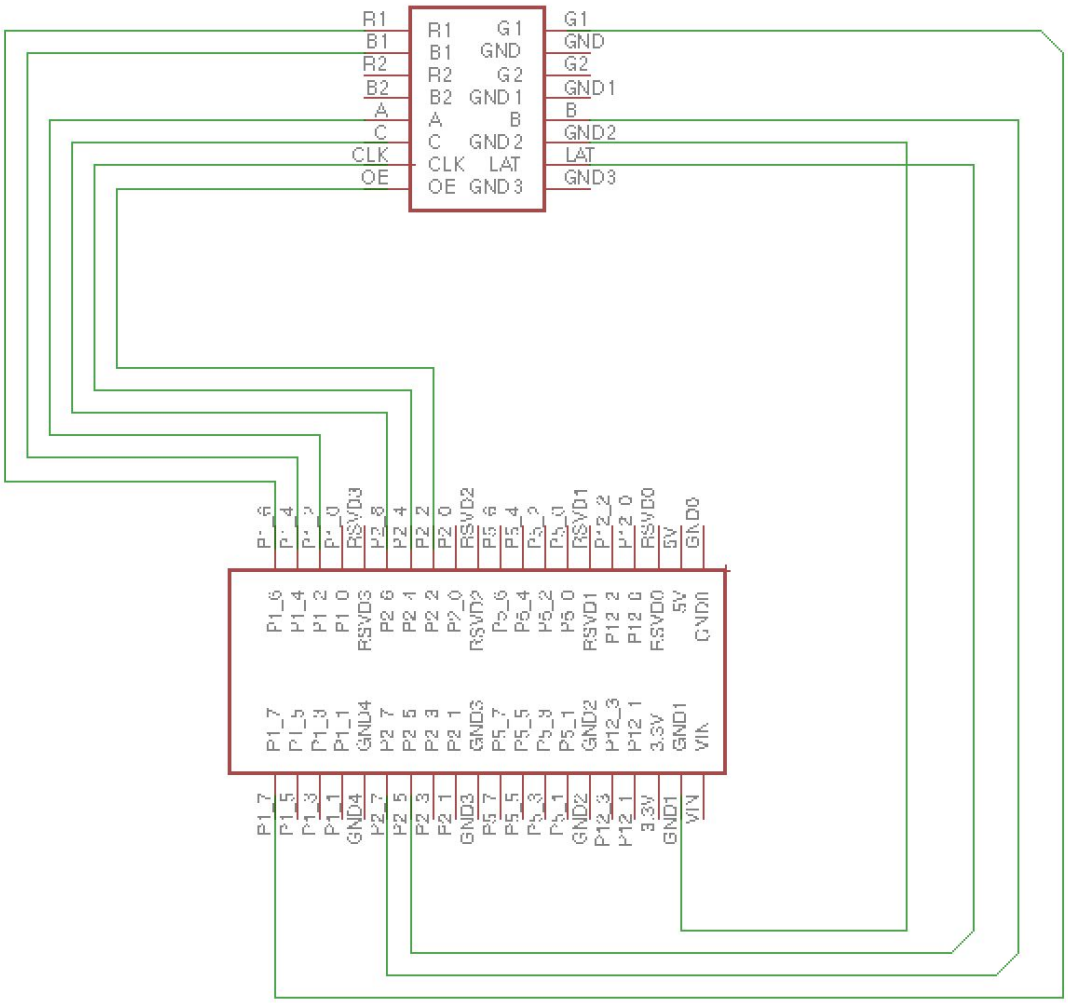
The Player ID identifies the players. A non -zero Pass filed identify skipped move. The fields must be separated by spaces or tabs, and each record must end with a newline. For more detail click here.

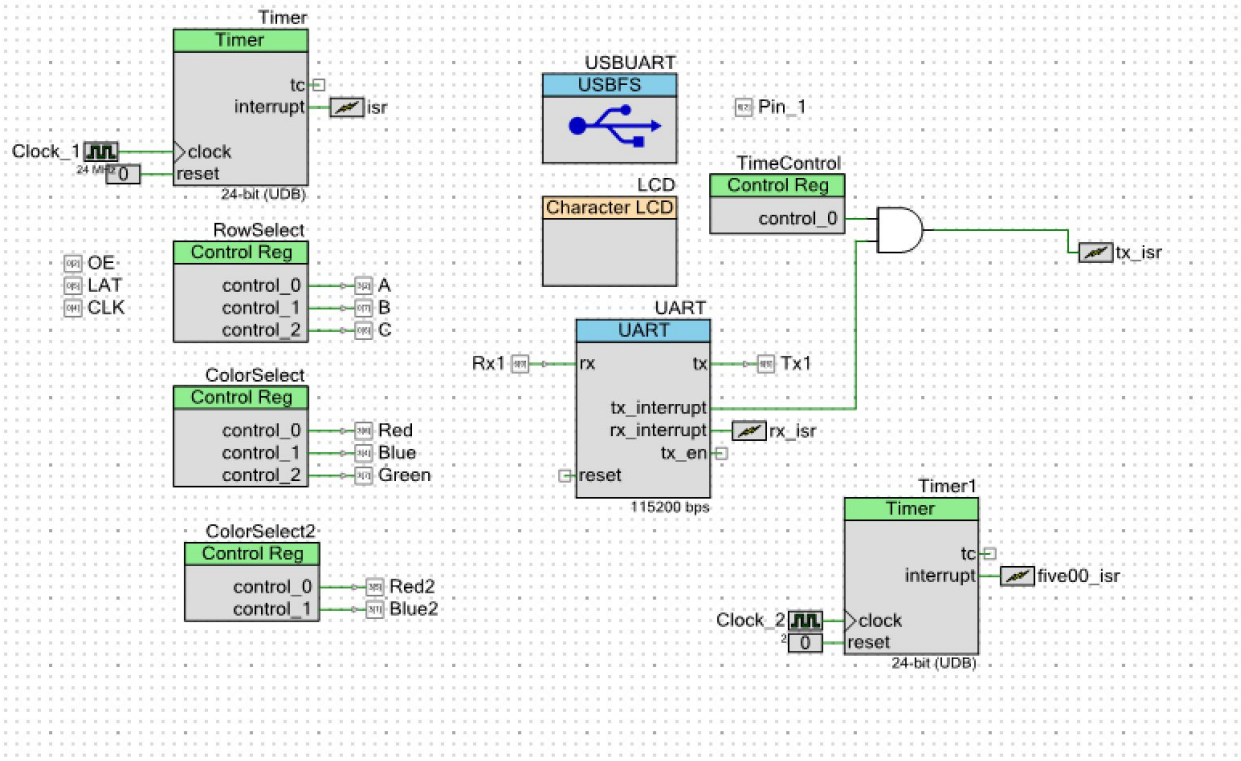
Using the WiFi Modules

CC3200 from Texas Instruments is the WiFi module. Using this module to transmit and receive data to/from the CC3200 using a UART in the PSoC-5. Only need to connect 4 pins of this board: Ground, 5V power, UART0_TX and UART)_RX.

Procedure of the Project:

The first goal is to get the game logic working, but before that, I need to connect all the wires together. I started off creating an schematic of the design as my reference, then soldered my board together following the reference. Below shows my schematic and my top level design.





As you can see from above, I used an AND gate between the TimeControl register and the TX interrupt, the reason is to turn on TX interrupt every 500ms.

Game logic:

While soldering the wires on to the PCB, I did the continuity test using a digital meter to make sure there are no short circuits or open circuits. After everything is soldered and ready to go, I made a test program to insert any color of LED onto the 16 by 32 board. Once this is working, then I moved on to interface with host keyboard so that the LEDs can move according to the direction arrow. In order to make this work, I used USBUART to be the communication between Host Key and my game. I defined two global variables to represent Row and Column so that it could change according to the key press. Once that works, then I started to implement the game logic. The game logic consist of the board game always start with four tiles in the middle, tiles cannot go out of boundary, tiles gets flipped .

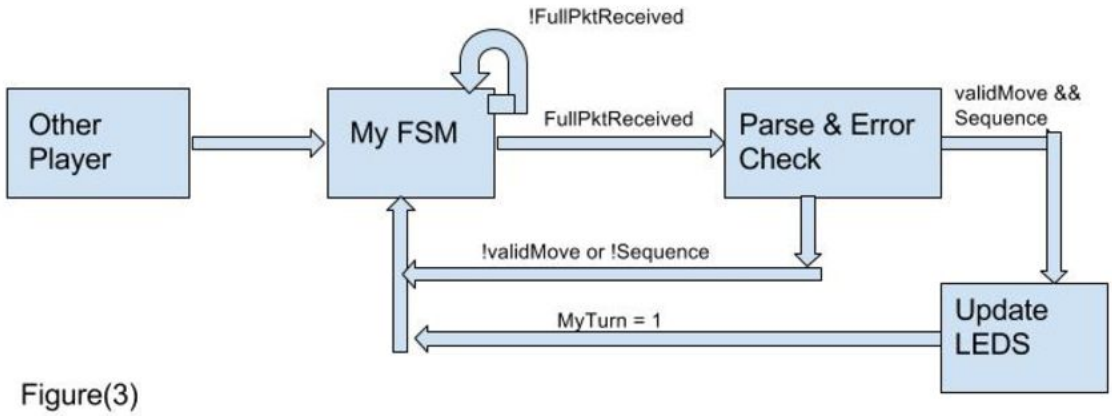
int CheckMovesValid(int direction, uint8 currPlayer[BOARDSIZE][BOARDSIZE],uint8 oppPlayer[BOARDSIZE][BOARDSIZE], int PlayerRow, int PlayerCol);

int FindBracket(int direction,uint8 currPlayer[BOARDSIZE][BOARDSIZE], uint8 oppPlayer[BOARDSIZE][BOARDSIZE], int PlayerRow, int PlayerCol);

Both function have the same parameters, one checks for valid move, one flips opponent's tiles.
Int direction takes in the direction that the player wants to go.
uint8 currPlayer[BOARDSIZE][BOARDSIZE]
uint8 oppPlayer[BOARDSIZE][BOARDSIZE]
These two parameters will take in two players depending on who's turn it is.
int PlayerRow, int PlayerCol is which row and col that the player wants to insert.

Testing Game Logic
Now that the game logic is done, I configured it to a 4 by 4 board. I played the game against myself, each time I hit enter, the player switch turns. Initially, there were a lot of bugs because of the borderlines, but after spending sometime, I covered all the edge cases.

TX, RX Interrupt
Once I can play a game with myself, then I moved on to implementing my program so that it could play with other people using UART. I first came up with a rough high level diagram of my program flow. Shown in Figure(3).



Figure(3)

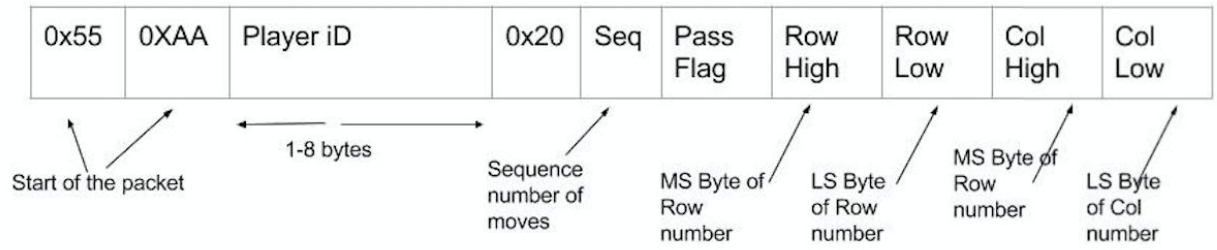
Then I added UART component to my top design and configured the UART and connect to someone else to see if I can transmit and receive correctly. Once that's done, then I moved on to coming up with an algorithm to parse all the information in a packet. I added helper functions to this part of the program.

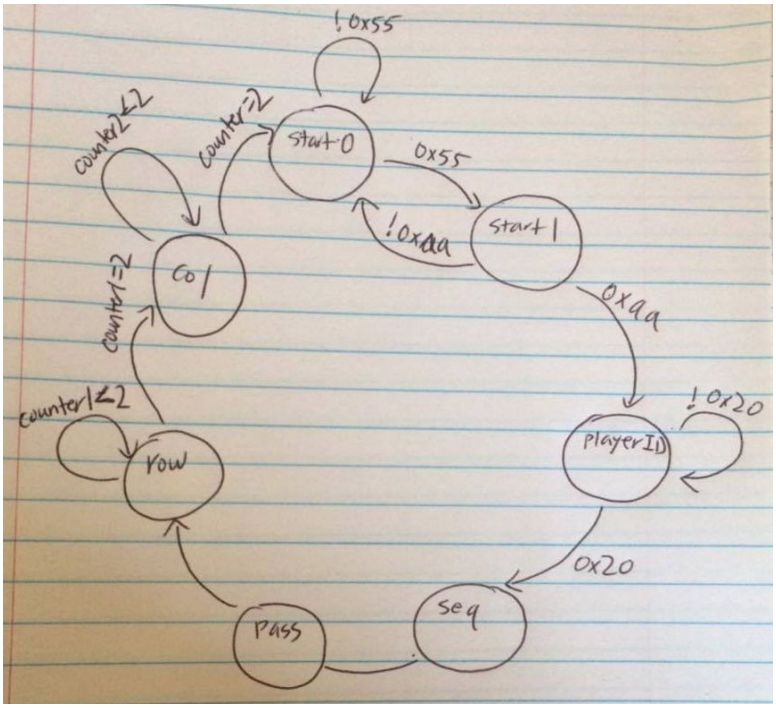
void PacketUpdate(void);
void Parse(void);
void PlaceOppTile(void);

PacketUpdate() contains the packet information that I will send out.

Parse() parses the packets received from opponent so I can use it in my program.
PlaceOppTile() place down the tile use the parsed information.

The following shows the packet format:





The most work will be done inside the Parse() function because I will be using a statement to grab all the information to store into different buffers. Those buffers will be used later to check for opponents valid move, or place down the tiles. Once I iterated through the states, then I have all my information parsed at the end of my state, which is *col.* From here, I can use the information to check for different rules, such as the sequence number, valid moves. If both of these cases are true, then I can place down a tile on the board. At this point, I connected TX and RX wires on the GPIO to someone else, and check whether if they could place a tile on their board, and if I could place their tile on my board. Once that's in check, then I started to implementing game rules to tell whether if a move is valid or not. I added a function called *int Sequence(void)* to help me check to make sure that the opponents sequence number is incremented one by one. In order to check opponent's valid move placement, I used my old function and passed in my opponent's row and column.

Testing TX,RX:

I changed the row and column parameter in the *FindBracket()* and *CheckValidMove()* function to the opponent's move. So everytime I hit enter, my packet will be updated. Then, the parsed Row and Column will be placed on the game board.

Connecting to WiFi

Once the UART is transmitting and receiving correctly, I moved on to working on the WiFi. From reading the instruction, I downloaded the program onto my WiFi board. Then, connected the TX pin on WiFi board to RX pin PSoC5 board and vise versa. There are keyswords that the WiFi board will take notice, "advertise", "connect", "disconnect" and "data". I made two character arrays:

char array[19] = {"advertise wuyuan12\n"}
char connect[24] = {"connect 192.168.0.102\n"}

Using the API USBUART_PutString(), we could send these two message to the WiFi board, and it will scan in the keyword 'advertise' and 'connect'. Then the program will be connected to WiFi. Once the WiFi is connected, I have to modify my packet to match the WiFi board format, which is just adding 'data' in front of the packet. Then, I will send that packet to the WiFi board. Furthermore, I have to modify my *Parse()* function so it could read the WiFi packets.

SD Card

Using the EmFile block, I will be able to communicate the SD card with the SPI master, by using the API, I will be able to read and write to the SD card.

Result:

My game play works completely, however, I had a lot of problems debugging my UART transmitter and receivers. When I first tested my program, I just place the tiles without any check validation for the moves, and it worked fine. Problem started to occur when I added check function to check for valid move and the sequence number. I was able to place a tile onto the opponent's board, but my opponent wasn't able to place his tile onto mine. I started to debug, and realize that I might have an algorithm in my parsing function, where I calculate the sequence number. Also, I might have a problem with my check for valid move function. Since I know that my transmit and receive works, I moved on to figure out how WiFi works, I was able to advertise and connect to WiFi. Unfortunately, I was unable to get to the SD portion of this lab. But I have read documents on how to use the EmFile API to communicate with SPI.

Conclusion:

Overall, this is a fun project, because it is cool to build a game using embedded C programming. Also, I was able to learn how to solder for the first time. It is very frustrating to work on the TX, RX portion of this project. However, through the process of debugging, I have learned a lot about event driving program, and also communication between PC and USB. This class have taught me a lot about both hardware and software perspective of engineering. I believe I want to focus my study on embedded systems.