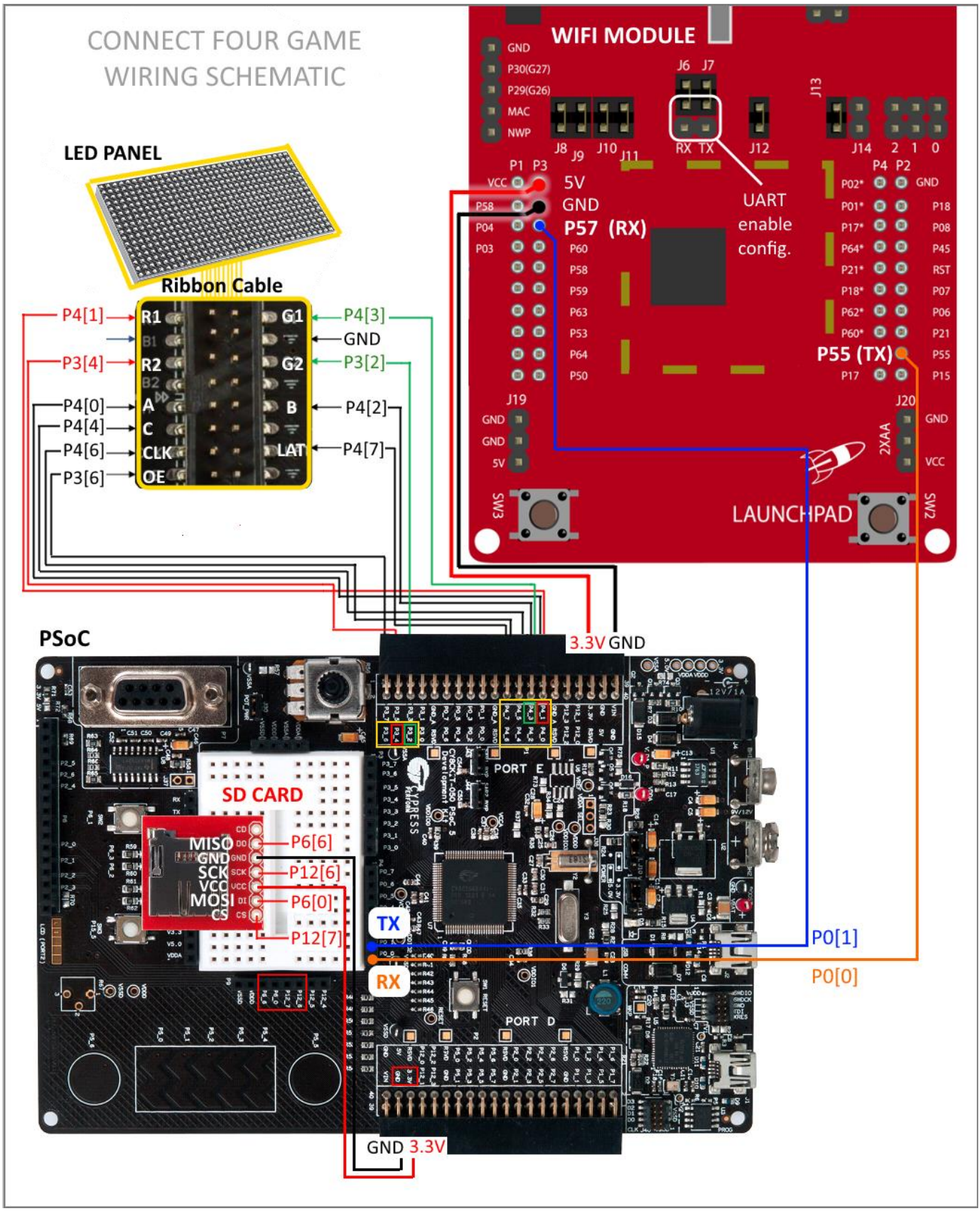
Connect Four Game

Lab Project Report

Ruben K. Lopez

CE 121L: Microprocessor System Design Lab, Fall 2014

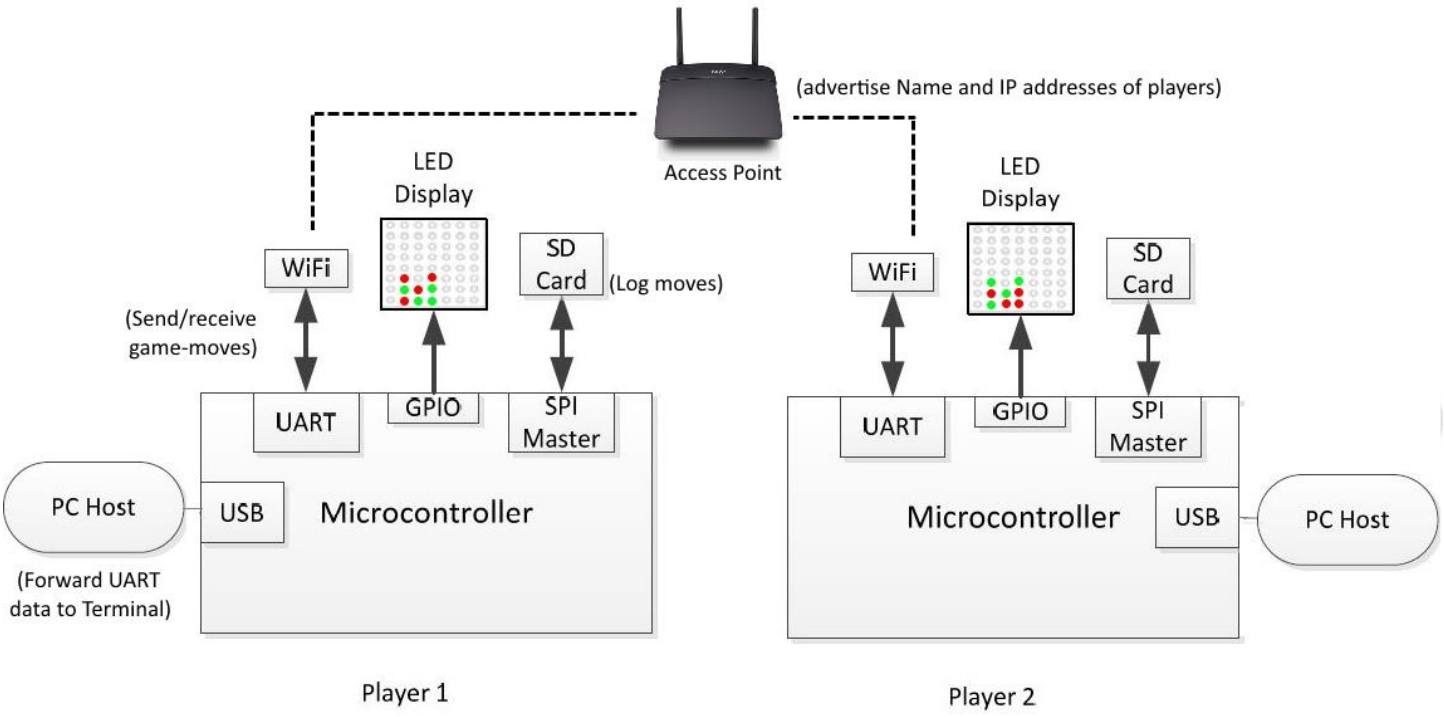


Author: Ruben K. Lopez
Tutor: John Ash
Section: Monday 9-11am
Date: 12/15/2014

Description

The purpose of this project was to put together multiple components into one system in order to play an internet-connected two player Connect Four game. This required us to integrate the applications of previous labs into our project such as the UART lab, the USBUART lab, and the LED Display lab. The CapSense example project was also integrated into the project.

Block Diagram



PSoC Top Design Schematic

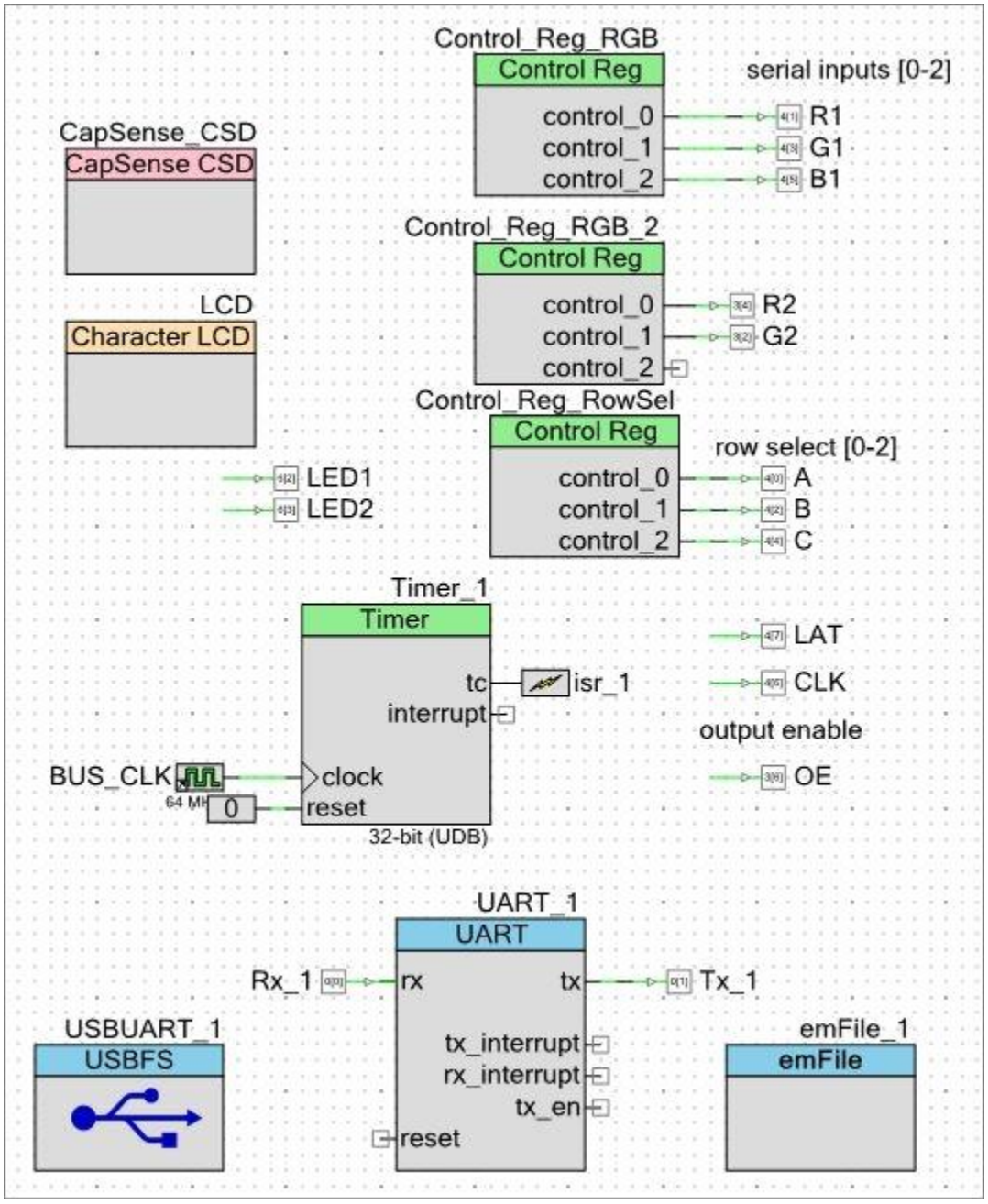
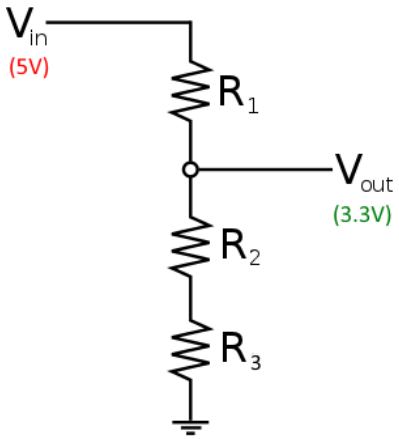


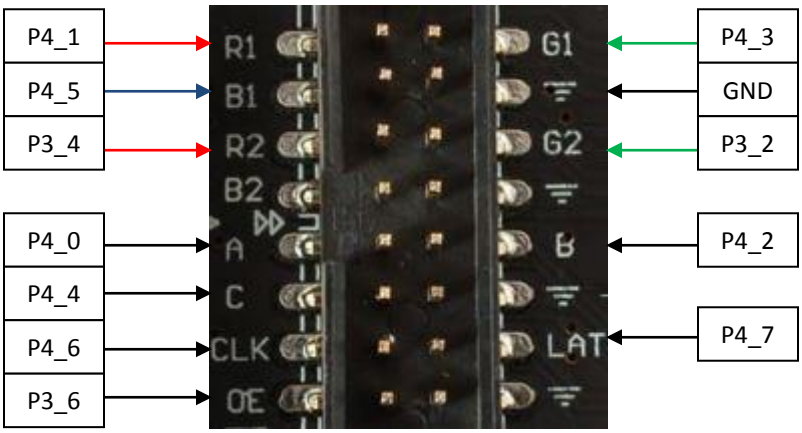
Figure 1. Top Design Schematic of Project

Hardware Design

Power Supply Voltages:
Since all the components of my project worked fine with a power supply of 3.3V, I didn't bother to solder anything to my Proto-Board since I didn't need to use any resistors for a voltage divider. However I understand that if I did need to use 5V and 3.3V, then I would implement a voltage divider like so:



How I interfaced the LED Display to my GPIO pins:
In order to connect the LED panel to my PSoC board, I connected one end of the ribbon cable to the INPUT connector (on the LED panel), and the other end of the ribbon cable to 12 wires which went to GPIO pins on my board. The following is a wiring diagram showing which GPIO pin is connected to which input of the LED panel:



Note: when I actually inserted my wires into the other end of the ribbon cable, I had to do it in reverse order (such that R1 and G1 are switched, B1 and GND are switched, etc). This is due to how the ribbon cable is wired from one end to the other.

For this project I only used Red and Green, which required me to use R1, R2, G1 and G2. (Note that, R2 and G2 are for lighting up the bottom half of the LED panel, i.e. rows 9-16 on the LED panel).

How I interfaced the Wi-Fi Module to my PSoC:
As shown by the image below, I only used 4 pins on my Wi-Fi card: GND, POWER, UART_RX (P57), and UART_TX (P55). Note that I supplied the Wi-Fi card 3.3V power from my PSoC board despite using the "5V" pin on the Wi-Fi card. As for the UART communication, I connected the Wi-Fi's RX to the PSoC's UART TX (P0[1]), and connected Wi-Fi's TX to the PSoC's UART RX (P0[0]).

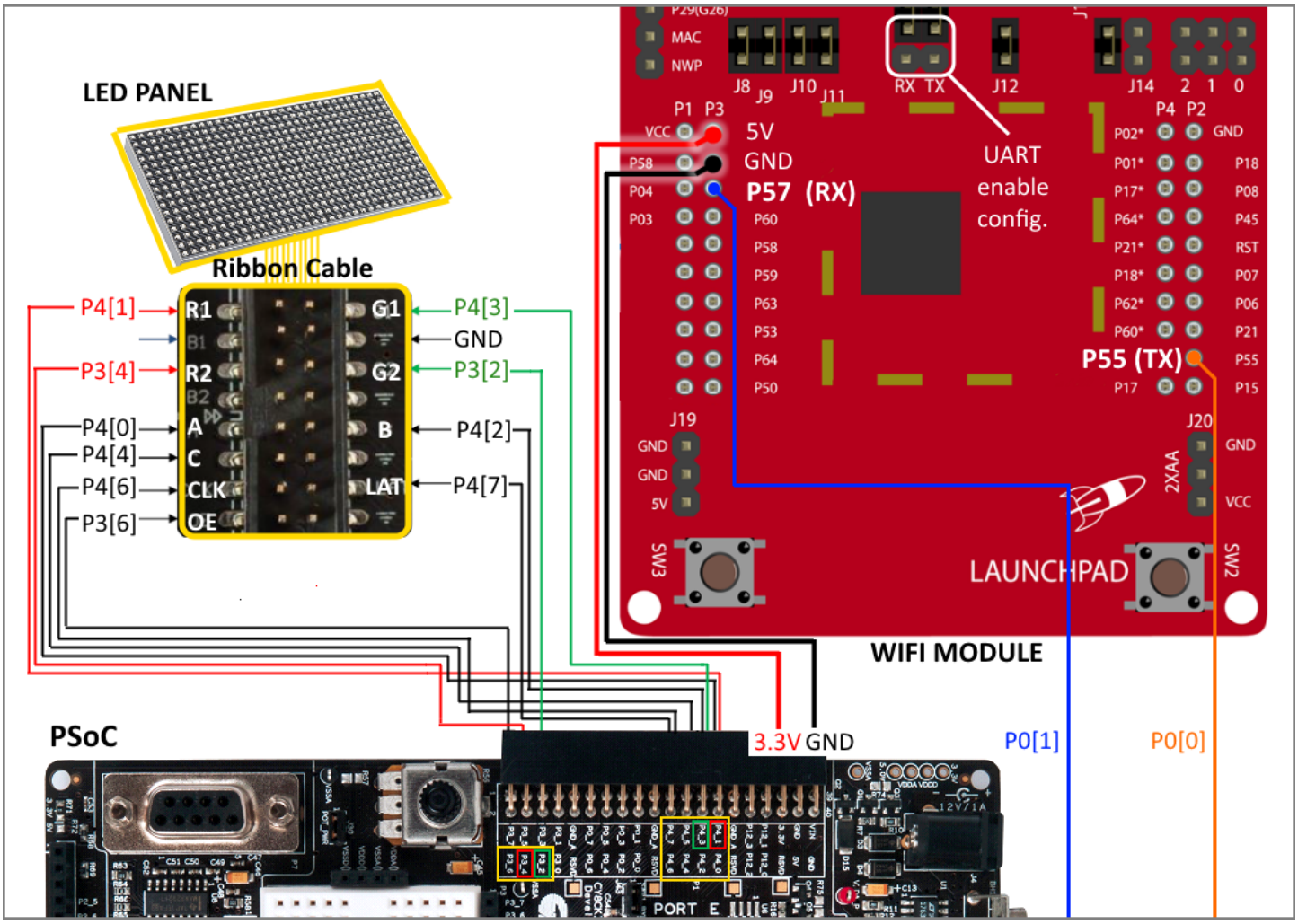


Figure 2. Interfacing the LED Panel and Wi-Fi module to my PSoC

How I interfaced the SD card to my PSoC:
After soldering a male header onto the SD card, I stuck the 7-pin header into the breadboard on the PSoC and then used 6 wires to connect the pins to the GPIO pins on my PSoC. (Note that I didn't use the Card Detect (CD) pin on the SD card since it wasn't necessary). As you can see from the image below, I used pins that were fairly close to the breadboard so that my wiring looked tidy. Note: The SD card uses the SPI interface method for communication.

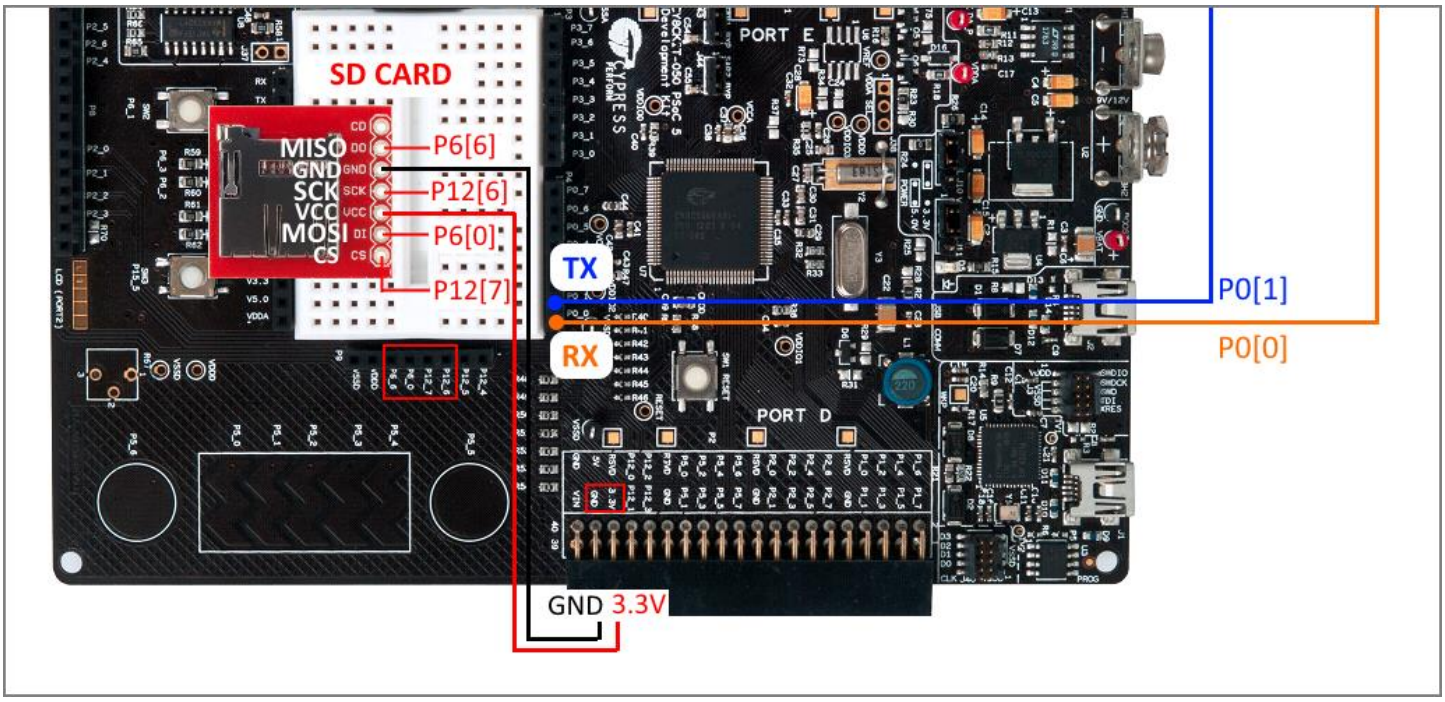


Figure 3. Interfacing the SD Card to my PSoC

Software Design

How the key modules of the software interact with each other:
In order to describe the events and interactions of my software, I'll go through a brief walk-through…

Game Initialization:

1. First we start in the **main** function. Here we initialize some components by calling a function such as **initSDcard**() and **initUSBUART**(), which contains the initialization code for the component.
2. Then we start our Timer, which enables the LED Panel by triggering a periodic interrupt to refresh the rows and columns.
3. Next, we enter the **handleWIFIadvertisement**() function, which advertises "RUBEN" and parses for a specific name to connect to. This was done with a Polling-loop, which is how I handled my UART communications. Eventually a flag would indicate we have connected to someone, and we would exit the loop and the function.

Game-loop:

1. Once connected to a player, we enter the main-loop, which constantly scans for CapSense input (via the slider or the CapSense button).
2. When the slider is touched, we enter the **changeColumn**() function, which passes in the current position of the slider, and checks for a range, thus updating the position of the LED on the Home Row.
3. Once I tap the CapSense button, we enter the **dropIntoColumn**() function, which uses the Home Row position to choose the column. The coordinates of this drop are then written to myDropMap (2D array).

4. Then, we enter **makePacket**(), which fills my txArray with the Row and Col position, and other packet details (according to the specified data format). A flag, “packetMade” goes high at the end of this function which triggers UART communication.
5. Once a packet is made, our main-loop detects the flag, and we enter the **handleUART**() function, which performs the UART communication. The function polls in a while-loop until I receive a valid packet from my opponent. During this polling-loop, I transfer my packet repeatedly (checking if TX_FIFO_NOT_FULL), and also call **TxPacketToUSB**() and **RxPacketToUSB**() in order to forward the data to a PC host terminal. After parsing a valid packet from my opponent, we get out of the polling-loop.
6. Once we exit the polling-loop, I call the **SDlog**() function, which appends the Player ID, Row, and Col to “log.txt”.
7. I also call other functions at the end of this polling-loop: **checkForWin**(), **writeOpponentCoordinates**(), and **checkForLose**(), which work together by checking myDropMap and my OpponentsDropMap (2D arrays).

Testing

How I assembled and tested my system:
I followed the steps laid out by the professor in the lab project specifications. I used sample projects such as the CapSense example project to get started and then progressed with previous labs: LED display lab, UART lab, and the USBUART lab.

I also had a test function which would place pseudo-random through the UART so that I could test the game playing by myself on my own micro.