**Lab 3 Report**

*Matthew Dragotto, Luke Moore, David Chen*

1. **Implementation:**

**Part 1**: Hello World

We set up the microblaze platform on our FPGA to run the Hello World program.

**Part 2A**: Serial Communication - Multiply two numbers

After finding out our serial communcation COM port, we were able to receive data from the terminal to our application. We used the scanf function to get the data from the user. The stange thing about the scanf function's implementation on the board is that it outputs a NULL byte after every character received. Thus, we had to manually parse our input string to remove all of the NULL bytes except for the null terminating character. Next, we find the product of the numbers and use the GPIO API to light an led if the product is greater than 100. Our delimiter character between the two numbers is an asterisk so an example input is "5\*8" which should output 40 and not light up an led on the FPGA. However if the input is "10\*11" then the program will output 110 to the terminal and light an led on the FPGA.

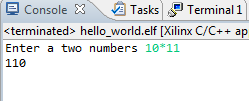


Figure 1: Example terminal output from part 2A. In this scenario, the LED lit up because the product was greater than 100.

**Part 2B**: Serial Communication - Rock Paper Scissors

We began by configuring the Diligent keypad inside of the EDK. This consisted of first adding it to the data bus, generating an base address for it, and then mapping the corresponding pins inside of our .ucf file. We then exported our new design to the SDK, where we began working on the code. In order to distinguish which key was pressed on the keypad, we had to manipulate which columns and rows are interpreted as outputs and inputs. By looking at the schematic, we can see that the rows are connected to VDD, and the columns are disconnected. This basically makes the columns a pull down path with a resistor. We could see by the schematic that by sending a high signal to all the columns besides 1, we can read corresponding row values to determine which (row, column) index was pressed.

We found a Diligent keypad demo library on the internet that did this for us, so we didn't have to trial and error with the bitmasks and find out the masks manually. Intuitively, the library works the same way that we imagined, by using bitmasks to distinguish the row column index of each key were reading from. This work is done in the KYPD\_getKeyStates() function, which relies on the KYPD\_lookupShiftPattern() function that has all of the bitmasks as returns. The getKeyStates() function outputs a 16 bit binary number, with each bit corresponding to a different key on the keypad. If a key was pressed, the bit is set to 1, and if not, it gets set to 0. See the "References" section for URL to keypad code.

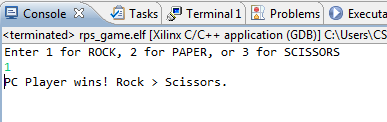


Figure 2: Example terminal output from part 2B. In this case, the PC player inputted a '1' for Rock, and the FPGA player inputted a '3' for Scissors, causing the PC player to win.

1. **Problems Encountered**

**Part 2A**: Serial Communication - Multiply two numbers

We faced a couple issues when starting out on this project. The first issue we faced was when adding a function call, the program would begin printing out random characters to the terminal. We eventually figured out that this was a memory issue, because when we added one line of code it would break, and when we removed that line it would work fine. The way we fixed this was by using smaller buffers, and also doing less printf calls (which we were using for debugging). Instead of print debugging, we used the IDE debugger tools and were able to solve this dilemma.

The second issue we faced was we were only filling our number buffers with one number. We eventually figured out through debugging and talking to the TA that the scanf() call was adding a NULL byte to the buffer after every other byte received. This meant that our number was basically null terminated after the first digit. We fixed this by manually looping through and removing the NULL bytes.

**Part 2B**: Serial Communication - Rock Paper Scissors

In our initial implementation of the keypad code, we experimentally figured out the values that were sent by pressing the '1', '2', and '3' keys. This however meant that the other outputs in the same column, such as '4', '5', and '6' were also being interpreted as Rock, Paper, and Scissors respectively by the board. In order to solve this problem we looked at the schematic for the keypad, and figured out how it worked (as described in Implementation). When we realized that we would need to manually figure out the bitmasks for data direction, we managed to find some code online that had functions for doing this already.

**3. References**

https://github.com/Digilent/vivado-library/blob/master/ip/Pmods/PmodKYPD\_v1\_0/drivers/

PmodKYPD\_v1\_0/src/PmodKYPD.c