Laboratory 01 - GPIO

By: **Caleb Probst**

Spring 2023 ECE 381 Microcontrollers

Introduction

This laboratory is used as an introduction to GPIO and how each pin can be used as either an input or an output, as well as the different drive modes that each pin has on the PSoC4. As well as learning GPIO, we also learned the basics of polling and strobing with the button matrix. We also learned how the LCD works and using it for debugging during the build process.

Materials

PsoC4 4200M Microcontroller	1
Hitachi LCD	1
100µF Capacitor	1
470Ω Resistor	1
4x4 Button Matrix	1

Procedure and Results

Part 1 – Setup in PSoC Creator

For this laboratory, the setup in PSoC Creator was relatively simple. We need 4 inputs and 4 outputs for the button matrix, as well as the LCD module. We named the LCD module “LCD”, the 4 inputs as C0 through C3, in resistive pull down mode, and the 4 outputs as R0 through R3, in strong drive mode. We also made sure that the LCD had the proper configuration, by setting it to a non-custom character set and with the conversion from ASCII to number in the configuration. Next, we setup the pins on the board to match what we needed for the inputs and outputs. The table of pin outs is shown below in *Table 1*.

R0	P3[0]
R1	P3[1]
R2	P1[0]
R3	P1[1]
C0	P1[2]
C1	P1[3]

C2	P5[3]
C3	P5[5]
LCD	P2[0 - P2[6]

Table 1: Pin Outs of Hardware on PsoC4

Part 2 – Hardware and Wiring

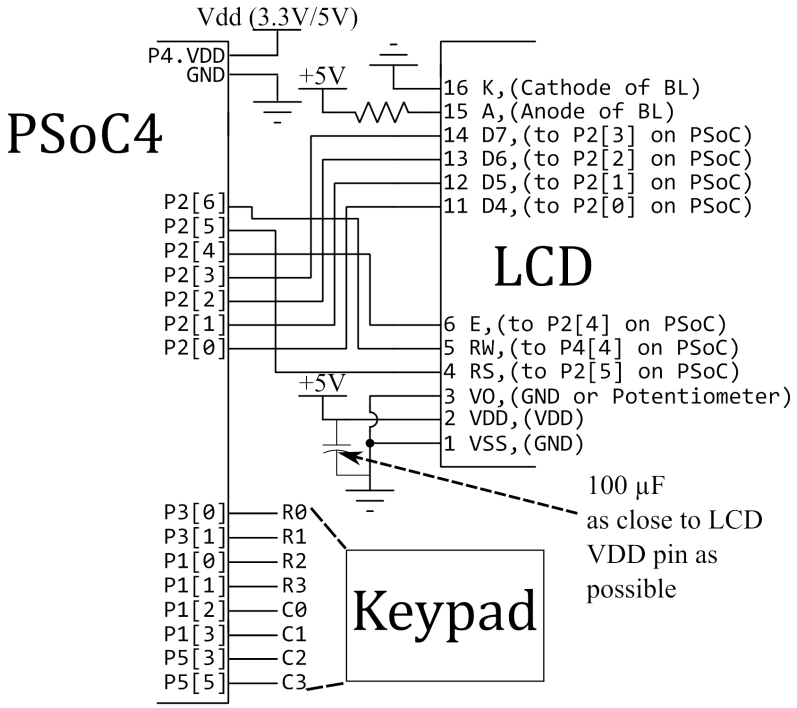

Figure 1: Wiring Diagram

For the hardware and wiring, we simply just connected it together using the wiring diagram given by *Figure 1*. For the V0 pin on the LCD we opted on using ground instead of a potentiometer. For the capacitor, we used a 100µF and for the resistor, we used a 470Ω We did run into a few issues, such as the wires not staying the breadboard properly, but we were able to workout the hardware bugs relatively quickly.

Part 3 – Software

For this part of the laboratory, we had a simple plan on how to get this to work. After doing the basic setup, such as declaring the variables we will be using and starting the LCD module, we started our infinite loop for the microcontroller, where our code will run. First, while checking for changes in the input pins, strobe the output pins on the button matrix. Once there is a change on the input pins, break out and complete the next step. The code for polling and strobing is given by *Figure 2* below.

```
while(C0_Read() == 0 && C1_Read() == 0 && C2_Read() == 0 && C3_Read() == 0){
    switch(x){
        case 0:
            R3_Write(0);
            R0_Write(1);
            x++;
            break;
        case 1:
            R0_Write(0);
            R1_Write(1);
            x++;
            break;
        case 2:
            R1_Write(0);
            R2_Write(1);
            x++;
            break;
        case 3:
            R2_Write(0);
            R3_Write(1);
            x=0;
            break;
    }
}
```

Figure 2: Code Block for Strobing and Polling

In this case, we used x as a variable to keep track of which output pin was being strobed, which will later become useful.

Next, we added a small delay to account for a small amount of debouncing, and then checked which input pin had a voltage on it. We used an extended-if statement for this part. Inside each check was a switch statement on the variable x, which told us which output pin had been detected at the time of change on the input pin. The code for this section is given below by *Figure 3*.

```
CyDelay(10);
// Do whatever
if(C0_Read() == 1){
    switch(x){
        case 1:
            // Print 1 (0,0)
            length = printChar('1', length);
            break;
        case 2:
            // Print 4 (0,1)
            length = printChar('4',length);
            break;
        case 3:
            // Print 7 (0,2)
            length = printChar('7',length);
            break;
        case 0:
            // Print * (0,3)
            length = printChar('*',length);
            break;
    }
}
else if(C1_Read() == 1){ ... }
else if(C2_Read() == 1){ ... }
else if(C3_Read() == 1){ ... }
```

Figure 3: Code Block for Checking Input Pins

Inside each else-if statement is a very similar chunk of code as the first. The function printChar is a custom function that includes a built in clear if the length of the text goes beyond the length of the LCD. This function is given below by *Figure 4*.

```
int printChar(char print, int length){
    if(length == 16){
        length = 0;
        LCD_ClearDisplay();
    }
    LCD_WriteData(print);
    length++;
    return length;
}
```

Figure 4: printChar Function

For this function, we opted to use the LCD_WriteData() function, instead of the LCD_PrintString() function as we were only printing one character at a time. Note that instead of clearing the display without printing the character, this setup will print the character immediately

after overflowing the limit on the LCD. This was a choice by the designer and can easily be implemented to only clear and not print the next character.

Finally, the last step was to debounce the buttons. Because these buttons were cheap and prone to debouncing, we used a very simple while loop to check for when the button gets released. This code is shown by *Figure 5*.

```
// Check for release
while(C0_Read() == 1 || C1_Read() == 1 || C2_Read() == 1 || C3_Read() == 1);
// Restart
```

Figure 5: Software Debouncing

Conclusion

This laboratory was very straightforward and fun to complete. In total, the project took around 2 and a half hours for us to complete. We had a few issues with software at first, which was quickly debugged and completed properly.