ECE341

**Lab1 - Digital Input and Output with the Cerebot MX7ck**

Report

**Seth Cram**

**9/14/2021**

Introduction:

The goal of lab one was to become more familiar with some of both the input and output modules of the PIC32. So, the lab required reading from buttons and writing to LEDs, since these are rather primitive and easy to deal with.

We had to read inputs. Reading inputs referred to reading from button 1 and button 2, and decoding these inputs. In order to read these inputs we had to set the button 1 and 2 corresponding pins to digital inputs.

The next goal was to write to outputs. The outputs for this lab were LED1-4. So, using the buttons input, we wrote to one of the LEDs. In order to write to the correct LED we had to decode the button input and light up an LED. To light up an LED we first needed to set the corresponding pins to digital outputs to allow writing to them. Initializing these LEDs is also important so our first output to the board isn’t ambiguous.

Implementation Discussion:

The software implementation went as I expected for the most part. I added four new sections to the ‘Project1.c’ file provided.

Section 1:

Above while() but below initialize\_system() in main(), I added:

PORTSetPinsDigitalIn(IOPORT\_G, BTN1 | BTN2);

PORTSetPinsDigitalOut(IOPORT\_G, LED1 | LED2 | LED3 | LED4);

PORTClearBits(IOPORT\_G, LED1 | LED2 | LED3 | LED4);

As abstractly described in the introduction, I set the PortG bits corresponding to button 1 and button 2 as digital inputs to later read from. I then set the PortG bits corresponding to all four LEDs as digital outputs. In order to initialize the values of the LEDs, I clear their bits.

Section 1 relies heavily on the peripheral library for the sake of readability. A different approach could utilize the SFRs, but would result in a lower level of readability. Overall, this section serves to initialize and setup the needed inputs and outputs.

Section 2:

Within the while loop:

int btns\_State;

int leds\_State;

btns\_State = read\_buttons();

leds\_State = decode\_buttons( btns\_State );

control\_leds( leds\_State );

Testing and Validation:

Conclusion:

How to read from and write to specific IO pins on the processor without affecting any other pins:

Reading from specific IO pins without affecting other pins is rather simple. You just need to configure the desired pins as digital inputs, by either using the peripheral library’s PORTSetPinsDigitalIn() and passing a mask of the pins we want with their corresponding port, or by manipulating the SFRs of the corresponding port. For the second approach, we’d need to configure TRISx to 1 to set the pin direction to an input, then read from PORTx and bitwise “AND” the result with a bitmask for the desired pins. This approach assumes the drain control is already set to 0, and an analog function isn’t MUX’d with our wanted pin to read from.

We have to be a bit more careful writing to specific IO pins without affecting other pins. If we aren’t careful, we could accidently set neighboring pins. Firstly, we’d have to set the pins to outputs. Our first method uses the peripheral library again with the PORTSetPinsDigitalOut() function which takes the port, and a bitmask for the desired pins to set as outputs. Before writing to pins, we should use PORTClearBits() to clear these bits in order to accurately test our writing. We can then use PORTSetBits() to write to the correct port and its pins. A second method involves using the port’s SFRs, but the peripheral library method is preferred for both readability and simplicity.