Zachary DeLuca
ECE 341
Ben and Ethan
January 31st, 2024

ECE 341 Lab 1

Introduction

In this lab we will work with inputs and outputs to and from the PIC32 board and with passing values to and from functions. This lab will show how to read from the buttons and then process the values before passing them to a function that writes to the LEDs on the board. The processing also shows how to use bit masks and how to read variables in the IDE using the single step debugging.

Implementation

In this lab, there were 3 main steps. The entire setup could be described as “Read-Modify-Write” (RMW) . This overall scheme is what we see in the main function, as shown below in Listing 1.1:

Listing 1.1

```
int main()
{
   initialize_system();
   int ovaltine, literate;
   while(1)
   {
      literate = read_buttons();
      ovaltine = decode_buttons(literate);
      control_leds(ovaltine);
   }
   return 1;
}
```

In this code, literate (as in “well read”) is the variable that stores the read values from the buttons that pertain to the function of this lab. This will read and store only the values of the buttons we care about. The way it does this is extremely simple, as shown in Listing 1.2 below:

Listing 1.2:

```
int read_buttons(void)
{
   int mask = (BIT_6 | BIT_7);
   int garbo = PORTG & mask;
   return garbo;
}
```

This is the read part of the RMW process as it accepts and uses the read values of port. We then jump right to the modify part by filtering the value through the bitmask to sift out just the values of the buttons we care about.

The variable ovaltine (reference to “A Christmas Story”) then stores the value that will be written to the port that controls the led lights. The decoding step is also simple and shown below in Listing 1.3.

Listing 1.3

```
int decode_buttons(int buttons)
{
   int passed = 0;
   switch(buttons){
      case 0x00000040:
         passed = (BIT_13);
         break;
      case 0x00000080:
         passed = (BIT_14);
         break;
      case 0x000000C0:
         passed = (BIT_15);
         break;
      default:
         passed = (BIT_12);
         break;
   }
   return passed;
}
```

The decode_buttons function excepts the value read from the buttons filtered through to only show the buttons states then uses that to implement a truth table using a switch statement. The output of the truth table is the value passed back to the main function.

The last part is writing the values out to the LEDs using the control_LED function as shown in Listing 1.4.

Listing 1.4

```
void control_leds(int leds)
{
   LATGCLR=(BIT_12 | BIT_13 | BIT_14 | BIT_15);
   LATGSET = leds;
}
```

This function first clears the bits so that way the new value can be written fresh and not have any accidental bits keeping high values when the new value should drive it low.

Testing and Verification

For this lab, testing was both easy and simple as there were buttons to press and LED's to stare at. The LED's shone just as the given truth table had instructed, verified by the lab assistants. To verify that the code was doing what was expected throughout the development phase, breakpoints at each of the main function lines were used in conjunction with watch variables. The variables literate and ovaltine were watched to make sure the masks were correctly isolating the bits that were intended, and the breakpoints were used to make sure that the values were changing at the right times.

Post Lab Questions

Figure 1.1

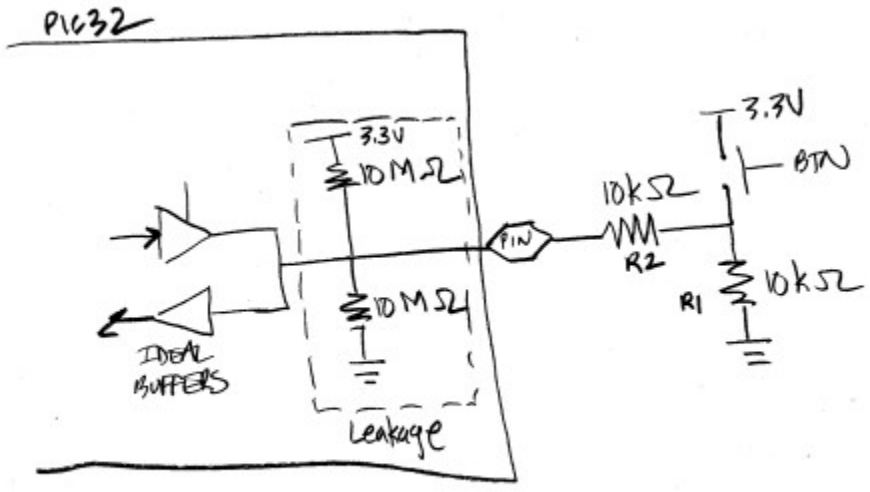

Figure 1: Cerebot push-button schematic

In this lab, we will venture off into left field now to discuss how these resistors effect the voltage of the pin. If we are interested in the voltage at the node, a single node voltage equation should get us what we want. The value of the pin when the button is pressed is shown below in figure 1.2.

Figure 1.2

$$0 = \frac{V_{PIN} - 3.3V}{10k\Omega} + \frac{V_{PIN}}{20k\Omega} + \frac{V_{PIN}}{10M\Omega} + \frac{V_{PIN} - 3.3V}{10M\Omega}$$

$$V_{PIN}(\frac{1}{10k\Omega} + \frac{1}{20k\Omega} + \frac{2}{10M\Omega}) = 3.3V(\frac{1}{10M\Omega} + \frac{1}{10k\Omega})$$

$$V_{PIN} = 3.3V(\frac{1}{10M\Omega} + \frac{1}{10k\Omega})(\frac{1}{10k\Omega} + \frac{1}{20k\Omega} + \frac{2}{10M\Omega})^{-1}$$

$$V_{PIN} = 2.2V$$

-3

The next figure shows what to expect when the button is not pressed:

Figure 1.3

$$0 = \frac{V_{PIN}}{20k\Omega} + \frac{V_{PIN}}{10M\Omega} + \frac{V_{PIN} - 3.3V}{10M\Omega}$$

$$V_{PIN}(+\frac{1}{20k\Omega} + \frac{2}{10M\Omega}) = 3.3V(\frac{1}{10M\Omega})$$

$$V_{PIN} = 3.3V(\frac{1}{10M\Omega})(\frac{1}{20k\Omega} + \frac{2}{10M\Omega})^{-1}$$

$$V_{PIN} = 0.00658V \approx 0V$$

The voltages are both reasonable values to drive a bit in the software high or low based on whether or not the button is pushed.

The R1 resistor in this setup allows for a grounding path with reasonable current draw so that the bit can read a zero when not pushed. The 10M resistor within the board would not dissipate the built charge fast enough for the software to read a zero when the button is no longer pressed for most applications, so the extra resistor and path are introduced.

As for how much current is supplied during LED operation, the board pushed 5.5mA, as shown in the figure below:

Figure 1.4

$$3.3V = 0.7 + 470I$$

$$I = \frac{3.3 - 0.7}{470} = 5.5mA$$

✓

This is well within the maximum as listed on the manual of 600mA.

Conclusion

In this lab we were able to use the LAT SFRs to write out to LEDs and also use the ports to read button data so that we could use the read modify write style of programming in a round robin organizational scheme. We showed how each of the three processes more or less fit into its own function. After we did the actual lab, the post lab assignment showed us how to use analog circuit analysis to determine the voltage of the port and why a certain resistor was there. The most useful part of the lab was learning how to read from the ports, use the bit masks and the bit operators and then writing out to the ports as well.