Lab 2: I/O

September 30, 2016

Contents

1 Objectives 1

2 Pre-Lab: Creating the circuit 1

3 Activating the LEDs 2

4 Reading the switches 3

5 Debounce 3

6 Bonus: switch matrix 4

1 Objectives

By the end of this lab, you will:

• Create a matrix circuit

• Read input and respond using this circuit

• Have a better understanding of the circuit and I/O code necessary for the Tic-Tac-Toe project

2 Pre-Lab: Creating the circuit

We will be providing the parts necessary for this lab (other than the Arduino) if you do not already have them.

You will need:

1. 3 LEDs (RGB or single color)

2. 3 switches

3. resistors (10k and 1k)

4. diodes (optional)
Using your patch board, wire up the following circuit. Every dangling wire besides the power (on the left hand side) should be connected to a different I/O port on your arduino (5 total).

![LED Switch Circuit Diagram]

Figure 1: LED / switch circuit

Note: you can skip the 10k pullup resistors if your arduino supports \texttt{PINMODE(INPUT\_PULLUP)}

3 Activating the LEDs

First, we are going to turn on the LEDs using this circuit. This is as simple as:

- Provide power to the row of LEDs
- Ground each column to turn on that corresponding LED

Task: Use an array of 3 booleans to determine whether an LED should be on or off and write your main loop code to activate the LEDs based on this array.
4 Reading the switches

Now to use the LEDs and switches at the same time, we’ll use the following loop logic:

1. Activate row of LEDs with HIGH signal
2. Ground the columns for LEDs that should be on
3. Wait some period of time (50 milliseconds should do)
4. Deactivate row of LEDs and active row of switches with LOW signal
5. Read each column (LOW corresponds to a switch being pressed)

In order to wait some period of time in step 3, we will not use the `delay()` function. While it may work for a simple scenario like in this lab, it is bad practice since it freezes your arduino from performing any computational steps. Instead, use the `millis()` function to read time from the onboard clock and use comparisons.

Task: Make your switches turn a corresponding LED on or off.

5 Debounce

Switches have some strange physical properties. While pressing a switch, there is a small window of time (a few milliseconds) during which the switch may bounce between HIGH and LOW before settling into a consistent state:

If your software performs an action for each switch state change without accounting for bounce, the output might be unpredictable. We can account for this physical issue in
code:

```plaintext
if switch state has changed then
    state ← switch state
    Wait 15 milliseconds
    if state is same as current switch state then
        perform state change action
    end
end
```

**Algorithm 1:** Debounce

**Task:** implement this debouncing step in your code (again, without using `delay()`). Keep in mind the time spent while the LEDs are activated and how it might affect or help your code (assuming that a switch will be depressed for longer than 50 milliseconds).

Note that debouncing is not always necessary. For example, if a switch state changes only once in the life of a program, then action can be taken immediately upon switch state change because no further changes will have an effect.

6 Bonus: switch matrix

Consider the following circuit of switches that is read by providing a HIGH signal to I/O’s 1 and 2 individually and reading I/O’s 3 and 4:

![Figure 3: source: 3 switches closed](image)

Looking at this circuit, we see that S2 will read as closed since S1 completes a circuit between I/O 2 and 4.

To fix this, we can use diodes to prevent this masking:
Since I/O’s 1 and 2 provide a HIGH signal, we can see how the diode prevents that HIGH signal travelling backwards through S1 to I/O 4.

**Task (optional):** Include diodes in your circuit and test that pressing multiple buttons does not create a masking issue.