ENGR 40M Project 3c: Switch debouncing

For due dates, see the overview handout

1 Introduction

This week, you will build on the previous two labs and program the Arduino to respond to an input from the environment or users. There are a number of things you can do. This handout covers one of these options.

Specifically, in this lab, you will learn about **switch debouncing**. Ideally, when you press (or flick) a switch, the connection would be made or broken cleanly and exactly once. It turns out that in real switches, the contacts bounce off each other many times. This phenomenon is called switch bouncing, and the act of working around it is called switch debouncing.

The goal of this lab is to get you on the road with using pushbutton switches correctly, and it is shorter than the music-responsive cube lab. That leaves you more time to implement something cool with your Arduino, and we will expect you to do so. See the lab overview handout (a separate handout) for some ideas.

By completing this lab you will:

- Learn how to use the oscilloscope.
- Observe switch bouncing in action and understand why it needs to be avoided.
- Learn how to debounce a switch.

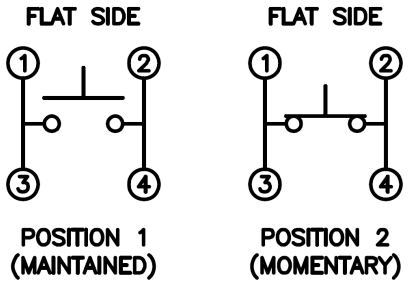
2 Prelab and new equipment

2.1 Connecting a switch

P1: Draw a schematic for how you would connect a switch to your Arduino. The schematic should show the switch, the Arduino input pin, and any other components (if any) to interface the switch with the Arduino. Use the least number of other components you can. (*Hint: You've seen this circuit before.*)

The pushbutton switches we have look like the one in the image below. (We have a few colors.) Notice that it has four pins, but only two connections: pins 1 and 3, and pins 2 and 4, are each connected inside the

switch. It's not clear from the photo, but the "flat side" in the schematic (on the right) corresponds to a flat side on the switch, a bit like the cathode of an LED (if you've noticed the flat side of an LED before).



P2: How would you orientate your switch on the breadboard? Draw the position of the switch on the breadboard. Make the rows of the breadboard clear. Label the flat side.

P3: Make sure there's enough space on your breadboard to support the switch! You don't need to submit anything for this question, it just pays to be prepared.

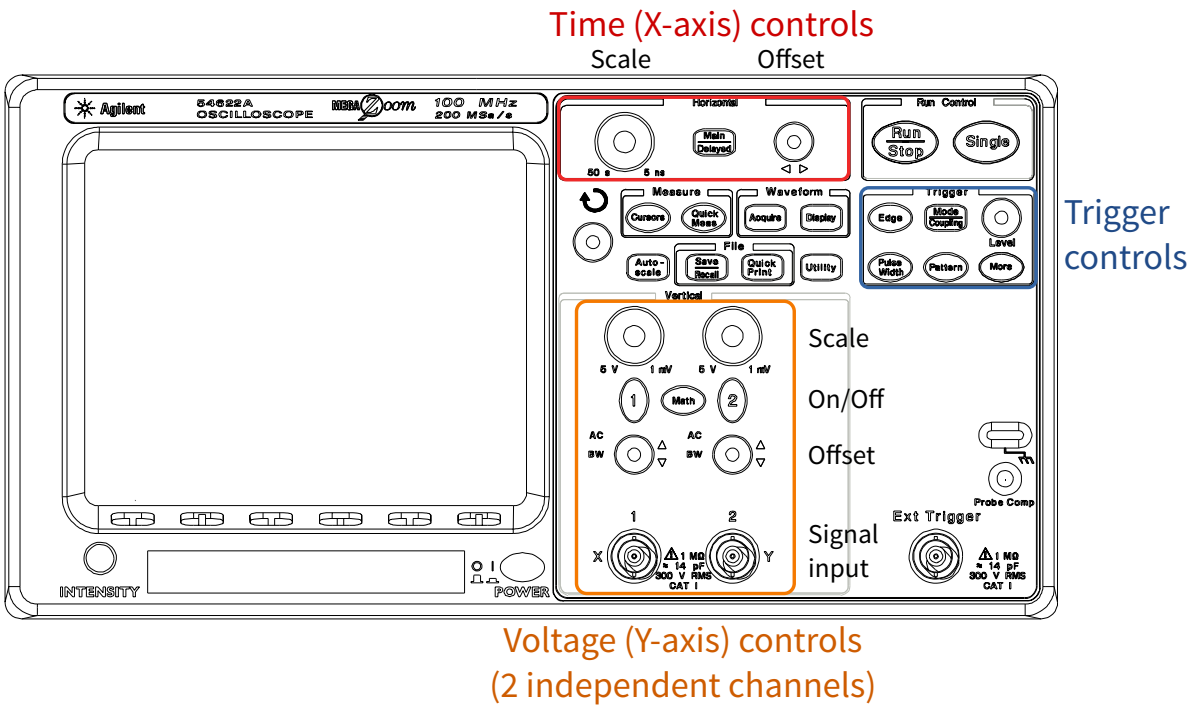
P4: Decide what you're going to use your buttons for! There are some ideas in section 2.2 of the lab overview handout.

2.2 Oscilloscope

An oscilloscope is one of the most useful tools for electrical engineers in debugging electronic circuits. At its core, an oscilloscope makes plots of voltage over time. This is very useful for trying to understand circuits that have rapidly changing voltages.

While an oscilloscope has an intimidating number of buttons and knobs, its basic operation is straightfoward. The diagram below shows the front panel of the Agilent 54621A (which is very similar to the DSO6012 that we have in the lab), but all oscilloscopes have a similar set of controls.

By the way: You might notice a button labelled "Auto-scale". **Don't use the Auto-scale button!** While this can be a convenient shortcut, it often fails and gives you something that you didn't want. More importantly, it short-circuits the process of thinking about what you expect to see. Too many students press "auto-scale" and assume that whatever they see must be the right answer. Once you get the hang of it, adjusting the scope manually will be nearly as fast, and far more likely to get you the right answer.



- Time (X-axis, Horizontal) controls:
  - The **large knob** controls the **scale** of the X-axis. The time covered in one vertical grid division is shown on the display (e.g, 10 ns, 50 µs).
  - The **small knob** shifts the signal left and right. The **offset** from the center is shown on the display.
- Voltage (Y-axis, Vertical) controls:
  - The BNC connector at the bottom is where you connect your probe.
  - The **On/Off** button (the one with the number on it) toggles whether the channel is displayed. The button lights up when the channel is active.
  - The **large knob** controls the scale. The display indicates the voltage per grid division.
  - The **small knob** shifts the waveform up and down. The **offset** from the middle is shown on the display.

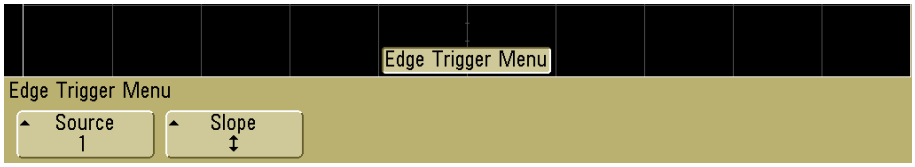
The **trigger** tells the oscilloscope when it should start its plot of voltage versus time. You set a trigger **event**, and it will center that event on the display. It's easiest to describe these by example:

- "Trigger when the voltage goes from below 3 V to above 3 V."
- "Trigger when the voltage crosses 0 V in either direction."
- "Trigger when the voltage stays below −1 V for at least 5 ms."

To set a trigger:

- The little knob controls the **trigger level**, the voltage threshold the oscilloscope uses to judge an "event".
- The buttons **Edge**, **Pulse Width** and so on are different **types** of triggers. In our work, we will always want to use the **edge** trigger. An edge trigger is when the voltage goes from below the trigger level to above it ("*rising edge*"), or from above the trigger level to below it ("*falling edge*").
- These buttons all open menus which allow you to adjust other details about the trigger.

  For example, if you press **Edge**, you'll pull up the "Edge Trigger" menu, which allows you to select which channel to monitor and whether to trigger on a rising edge, falling edge or either.



The oscilloscope will horizontally align the plot so that the **trigger event** happens at $t = 0$. You adjust where $t = 0$ is on the display using the **time offset** knob (the small knob in the "Horizontal" section described above).

Don't worry if triggering doesn't make sense now; it'll make more sense when you see it in action.

3 Investigating switch bouncing

3.1 What does the Arduino see?

The Arduino program `switch_bounce.ino`, available on the website, checks the state of the button on every `loop()`, and prints to the Serial Monitor every time it detects the button being pressed. It also prints the time since the last press.

1. Add a button switch to your breadboard and connect it an unused pin on your Arduino.
2. Download the file `switch_bounce.ino` from the website, and load it into your Arduino. Modify the constant `BUTTON` to reflect the input pin that you're using.
3. Open the Serial Monitor, and set its baud rate to 115200. Press the button a few times. Does it do the same thing every time? Keep pressing until you're satisfied you have a good idea of what's happening.

> **L1:** Does the counter increment by one every time you press the button? Describe what happens.

Does this seem weird? Maybe we should have a closer look at what's going on with this switch.

3.2 Connecting the oscilloscope

1. Plug an oscilloscope probe into channel 1 of the oscilloscope, and connect the probe to your Arduino input pin. (Alligator clips and bits of wire might be useful here.)

2. Make sure channel 1 on the oscilloscope is on, and channel 2 is off. (The light for channel 1 should be lit, and the light for channel 2 should not.) Set up the oscilloscope to inspect a signal that varies between 0 V and 5 V. You'll need to adjust the **vertical scale and offset** for channel 1 to do this. Adjust the **time scale** about 5 ms (the exact value doesn't matter, as long as it's around that range), and the **time offset** to zero.

3. Press the button, and verify that the signal jumps between 0 V and 5 V.

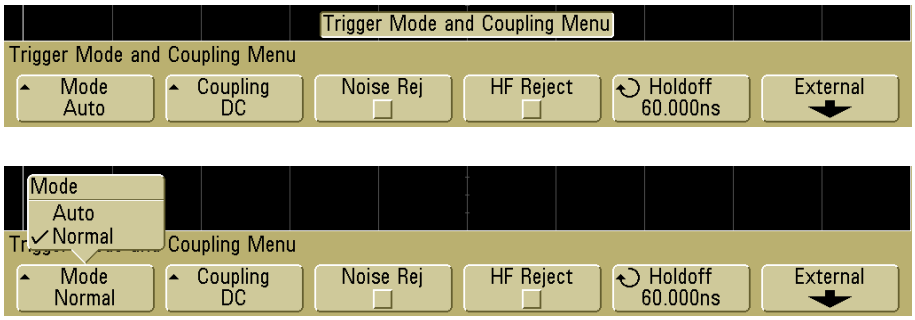
3.3 Setting the oscilloscope trigger

At this stage you've got the oscillscope to plot the voltage versus time, but it's doing it continuously, so most of the time it's just plotting when the switch is on or off. It's not very interesting just to plot voltage versus time when the switch is in a constant state. What we're interested in is the *transition* from on to off, or off to on. So we'll use the **trigger** to help us focus on this (as described in section 2.2).

The trigger event we want is: "Trigger when the voltage crosses 2.5 V in either direction." This way, we can catch the switch turning on or off.

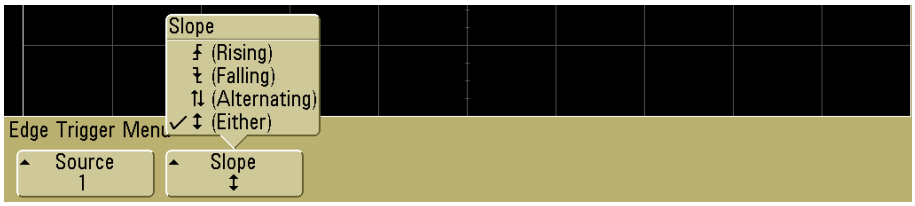
1. Set the trigger level to roughly 2.5 V. (The exact value doesn't matter, so long as it's somewhere between 2 V and 3 V.)

2. Press the **Mode/Coupling** button in the trigger section of the oscilloscope. It will display a menu at the bottom of the screen. Use the buttons along the bottom of the screen:
   - Set the **Mode** to "**Normal**". It's normally in "Auto". In "Auto", the oscilloscope constantly plots voltage versus time even if the trigger event does not occur. In "Normal", the oscilloscope will not generate new plot until another trigger event occurs.
   - Set the **Coupling** to DC.



3. Press the **Edge** button in the trigger section of the oscilloscope. Then, using the menu at the bottom of the screen:
   - Set the **Source** to channel 1.
   - Set the **Slope** to "**Either**".

     (This is the setting for "in either direction". The others mean "goes from below the trigger level to above", "above to below" and "whichever direction did not happen last time", respectively.)



Now press the button a few times. Every time you press the button, a new plot should be displayed on the oscilloscope. The plot should show the voltage on the input pin going from `HIGH` to `LOW`, or vice versa.

If this were an ideal switch, it would just transition once, precisely when you hit the button. But zoom in horizontally: adjust the time scale to about 100 µs. On this time scale, does it look like a clean transition?

L2: Draw a typical waveform. Don't forget to draw and label your axes, so we have some idea of the time scale you're looking at. It only needs to be approximate.

L3: The Arduino processor runs at 16 MHz, that is, it processes 16 million instructions per second. How long does *one* instruction take?

In a single invocation of `loop()`, your Arduino processes a lot more than one instruction, of course. But it's still pretty fast compared to a human, so it can probably read the switch every few microseconds or so, depending on how much the loop is doing.

L4: Press the button a few more times and see how long you can get the bouncing to last. You might like to adjust the time scale and offset to get a better view of all the bounces. How long can you get the bounces to last? Draw the waveform for your "worst case" bouncing.

4 Optional: Did we catch all the bounces?

This section is optional. *It uses external interrupts, which are beyond the scope of ENGR 40M. However, knowledge of interrupts is* ***not*** *necessary to complete this section, so long as you believe the code does what we say it does. Skip to section 5 if you don't want to do this section.*

The loop in `switch_bounce.ino` actually does quite a lot: it turns out that printing to the Serial Monitor takes a long time. This seems unavoidable, but it turns out there's an advanced technique you can use to make sure you *never* miss a switch transition. They're called *external interrupts*[1]. The file `switch_bounce_interrupt.ino` uses them to make sure it counts every single switch bounce.

For this to work, your switch must be connected to pin D2 or D3. You might have to temporarily disconnect one of your anode/cathode connections to do this.

1. Download `switch_bounce_interrupt.ino`. Modify the constant `BUTTON` to either 2 or 3, depending on which input pin you're using. Upload it into your Arduino.
2. Run it in the Serial Monitor. It won't print on *every* press: printing takes time, so if it did this, it would miss some switch bounces. But it will print a cumulative count of how many bounces it's detected. (Each transition from `HIGH` to `LOW`, sometimes called a *falling edge*, is counted.) Look at how many bounces there are!

[1] For the curious: An interrupt is when some event causes the Arduino program to stop dead in its tracks, run a specially dedicated function (*interrupt service routine*), and then return to what it was doing. In our case, the interrupt event is "The input on pin D2 changed state from `HIGH` to `LOW`", and our interrupt service routine increments a counter. If this didn't make sense to you, don't worry about it. Or if you're still curious, ask your TA.

5 Debouncing the switch

Debouncing a switch is simple in concept: you just refuse to read the switch input until it's been stable for some short time, which we'll call `DEBOUNCE_DELAY`. We've set `DEBOUNCE_DELAY` to 75 ms, but you should feel free to play with this value to see what happens. In more detail, our method is:

- On every switch transition, reset a timer.
- When, and only when, the timer hits `DEBOUNCE_DELAY`, accept the current switch state.

We've provided code that does this in `switch_debounce.ino`.

1. Download the file `switch_debounce.ino` from the website, and load it into your Arduino. Modify the constant `BUTTON` to reflect the input pin that you're using.
2. Press the button a few times. Verify that the counter increases exactly once each time you press it. Take a moment to read the code and understand what's going on.
3. Increase `DEBOUNCE_DELAY` and upload the modified program.

   L5: How big can `DEBOUNCE_DELAY` be before it stops working effectively? Describe what happens, and explain why it stops working for values of `DEBOUNCE_DELAY` that are too large. *Hint: Try pressing the button repeatedly really fast.*

4. Decrease `DEBOUNCE_DELAY` and upload the modified program.

   L6: How small can `DEBOUNCE_DELAY` be before it stops working effectively? Describe what happens, and explain why it stops working for values of `DEBOUNCE_DELAY` that are too small.

   Hint: Try hitting the button more aggressively (not necessarily repeatedly).

6 Onwards and upwards

Now that you've learnt about switch debouncing, you can add buttons or other switches to your circuit and get your LED cube to interact with users!

We deliberately have *not* provided template code that includes both switch debouncing and the LED cube driver. We leave it to you to merge the two together.

If you hand in this lab (rather than the music one), we expect you to implement something fairly complex with your Arduino. See our style rubric in the lab overview handout for guidelines on this.

7 Analysis (optional)

The analysis section of the debouncing lab is optional. We've left it in from previous years because it's interesting, but you don't need to do it.

A1: We had two switches in the useless box, and debounced neither of them. Why did this not turn out to be a problem?

Debouncing a switch is simple in concept, but getting the implementation right can be tricky. Here are a few *bad* ways to debounce switches. See if you can figure out why they won't work. For each one of these, describe what *would* happen, and why it does not detect one switch per user press like we expect it to.

In all of these code extracts, assume the following appears before the `loop()` function:

```
const int BUTTON = 2; // button pin
const int DEBOUNCE_DELAY = 75; // in milliseconds

void setup() {
  pinMode(BUTTON, INPUT_PULLUP);
}
```

Now, have a look at these attempts to debounce switches:

```
void loop() {
  static byte lastButtonState = HIGH;
  byte buttonState1 = digitalRead(BUTTON);
```

```
  byte buttonState2 = digitalRead(BUTTON);
  byte buttonState3 = digitalRead(BUTTON);
  if (buttonState1 == buttonState2 && buttonState2 == buttonState3
          && lastButtonState != buttonState2) {
    // register button state and do something
  }
  lastButtonState = buttonState2;
}
```

A2: Why won't the above code debounce the switch?

```
void loop() {
  static byte lastReading = HIGH;
  static unsigned long lastFallTime = millis();
  byte reading = digitalRead(BUTTON);
  unsigned long now = millis();
  if (lastReading == HIGH && reading == LOW) {
    if (now - lastFallTime > DEBOUNCE_DELAY) {
      // register a press and do something
    }
    lastFallTime = now;
  }
  lastReading = reading;
}
```

A3: The above code will debounce the user pressing the button, but it will also register spurious "presses" at another time. When, and why?

This next code uses the `display()` function you wrote for the LED cube in lab 3b.

```
void loop() {
  static byte values[4][4][4];
  static byte lastButtonState = HIGH;
  byte buttonState1 = digitalRead(BUTTON);
  delay(DEBOUNCE_DELAY);
  byte buttonState2 = digitalRead(BUTTON);
  if (buttonState1 == buttonState2 && lastButtonState != buttonState2) {
    // register button state and do something to values[][][]
  }
  display(values); // display a pattern on the LEDs
  lastButtonState = buttonState2;
}
```

A4: The above code will debounce the switch fine, but what is the problem with using the `delay()` statement like that?

8 Reflection

Individually, answer the questions below. Two to four sentences for each question is sufficient. Answers that demonstrate little thought or effort will not receive credit!

R1: What was the most valuable thing you learned, and why?

R2: What skills or concepts are you still struggling with? What will you do to learn or practice these concepts?

R3: If this lab took longer than the regular 3-hour allotment: what part of the lab took you the most time? What could you do in the future to improve this?