ENGR 40M Project 2b: Making the Useless Box More Awesome

Prelab due 24 hours before your section, July 17–20
Lab due before your section, July 25–28

1 Objectives

You've built your first useless box, spent countless hours playing with it and impressed your friends with your fun toy. You might, justifiably, be asking yourself, "how can we make this even better?" In this lab, we'll do just that—by adding a computer!

Computers are everywhere in electronics. It's not just your computer or even your phone. Countless everyday objects have a microcontroller in them—fridges, door locks, pacemakers, dot-matrix displays and cars (where you'd find several dozen), to name a few. These microcontrollers interact with the physical world, vastly expanding what we can design as engineers.

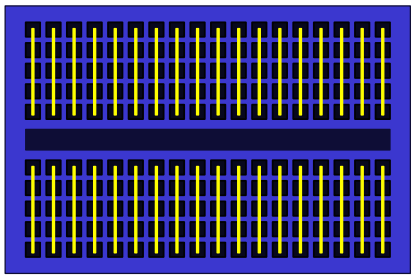
By the end of this lab, you should be able to

- Use MOS transistors to make a driver circuit, interfacing between an Arduino and a motor
- Program your Arduino to interact with physical inputs (switches) and outputs (motor) via digital pins
- Understand finite state machines and use them to organize your code effectively

2 Parts

2.1 Breadboard

A breadboard is a jig that makes it convenient to temporarily build and prototype circuits. It contains lots of pinholes into which you can plug wire-leads. Each row of five pinholes, indicated in yellow on the diagram on the right, is electrically connected, allowing you to connect up to five leads to a single node. Each row is electrically isolated from the rest of the board, allowing you to prototype large, multi-node circuits.



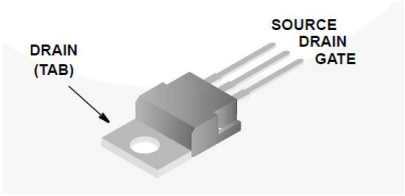
In this lab, due to the limited space available inside your box, we'll use the smallest breadboards.

2.2 MOS transistors

A metal-oxide semiconductor field effect transistor (MOSFET), or MOS transistor, is a three-terminal device that can be thought of as an electrically-controlled switch. (It's more complicated than that, but for our purposes, this is a good model.) We've studied this in class, so for details, refer to the course reader. To refresh your memory:

- An nMOS turns on when the gate is *higher* than the source by at least the threshold voltage.
- An pMOS turns on when the gate is *lower* than the source by at least the threshold voltage.
- Typically, we connect the source of an nMOS to ground, and the source of a pMOS to V_{DD}.

MOSFETs come in lots of different forms. We'll be using power transistors, designed to deliver high current: the RFP12N10L (nMOS) and NDP6020P (pMOS). Both devices use the *TO-220 package*, so they look similar—read them to tell them apart. The pinout is shown below, and is the same for both devices.

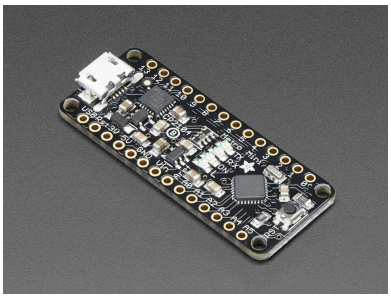


Some words of caution:

1. Transistors can make a short circuit if you connect them incorrectly. If this happens, your computer will (hopefully!) shut down its USB port in order to protect itself. So if nothing looks like it's working, measure V_{DD} to see if that's what happened.
2. **Floating gates can turn a transistor on!** We saw this in homework: Many students connect the source and drain appropriately, but then leave the gate disconnected. Don't do this! This can cause both the pMOS and nMOS to turn on simultaneously, short-circuiting your power supply. **Always** be sure to **actively** drive the gate of every transistor in your circuit, even during testing.[1]

   In particular, this **includes** *Arduino pins configured as inputs*, since they look like an open circuit.

2.3 Arduino



Arduino is a popular open-source electronics platform that eases the process of programming a microcontroller to interact with your circuit and the physical world. We'll be using a variant called the *Adafruit Metro Mini*, manufactured by Adafruit.

You'll need to install the Arduino IDE, the software that allows you to load code onto the Arduino, from `http://arduino.cc/`. The Metro Mini derives from the Arduino Uno, so select "Arduino Uno" in the IDE.

The Arduino interfaces with electric circuits via *input/output pins*, or *I/O pins*. In your code, you use the `digitalRead()` and `digitalWrite()` functions to interact with them. Input pins take almost no current, like a voltmeter. As for output pins, we'll learn about their *output resistance* in the prelab.

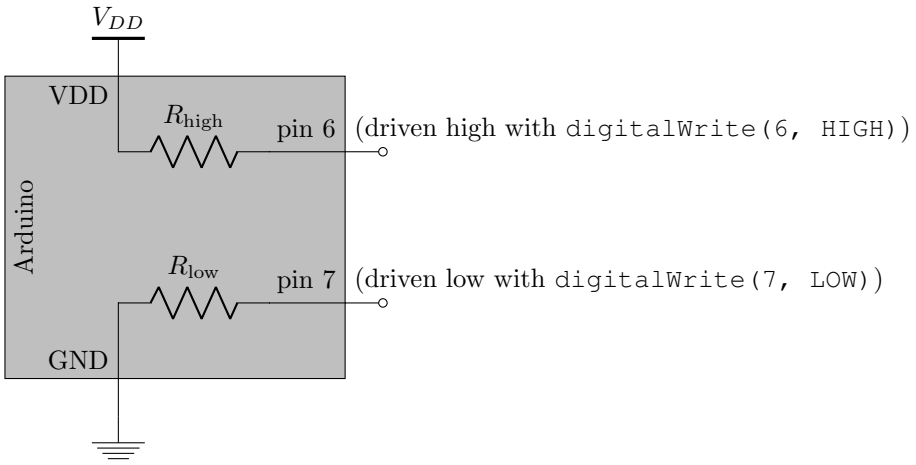
3 Prelab

3.1 Arduino output resistance

In this section, we'll measure the *output resistance* of an Arduino I/O pin. The output resistance is the resistance "seen" by a load on the output, when it is connected to V_{DD} or ground. For example, in the schematic below, pin 6 is *driven high* (set to `1`) and pin 7 is *driven low* (set to `0`). We can think of the output resistance when driven high as R_{high}, and the output resistance when driven low as R_{low}.[2]

[1] In practice, for reasons beyond the scope of ENGR 40M, a floating gate tends to "remember" the last voltage that was connected to it. However, you should never rely on this behavior, because it's unreliable and sensitive to brief accidental contacts.

[2] Actually, the output pins are themselves controlled by pMOS and nMOS transistors, which aren't shown in the figure, and the resistance is just the resistance of those transistors.



To get started, connect your Arduino to a USB port, and try loading one of the sample programs to make sure everything is in order.

P1: Choose your two favorite I/O pins, but don't use pins 0, 1 or 13. Write a simple program that configures both of them to be outputs using `pinMode()`, then drives one of those pins to `HIGH` and the other to `LOW`. Load the program into your Arduino.

This program should be four lines long. You don't need to submit it with your prelab.

P2: Measure the output resistance of the I/O pin that you drove high.

Note: Because this is actually a transistor, the polarity of your ohmmeter matters. The ohmmeter measures resistance by pushing a small test current from the positive lead to the negative lead, so you need to make sure this current would go in the "correct" direction. In this case, this means putting the positive lead on VDD, and the negative lead on your I/O pin, so that the test current goes from VDD to the I/O pin.

P3: Measure the output resistance of the I/O pin that you drove low.

Again, the polarity of your ohmmeter matters—think through which way the test current should run.

A naïve way to control the direction of your useless box's motor would be to connect it directly between two I/O pins. To drive the motor one way, you set one pin to `HIGH` and the other to `LOW`; to drive it the other way, you set the pins the other way round.

P4: Draw a model of the circuit that would be formed if you connected the motor directly between two I/O pins, one of which was driven to high and the other of which was driven to low. Model the motor as a 20 Ω resistor.[a]

[a] It's not actually a resistor, but this gives us some idea of the load it provides when it's running continuously.

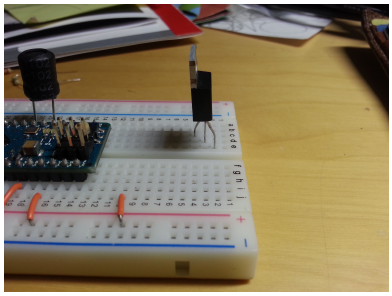
P5: Use the model you just drew to estimate the maximum current that this design would be able to provide to the motor. Compare this to your measurement of motor current from prelab 2a. Would this design work? Why, or why not?

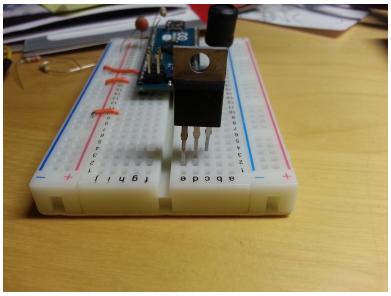
3.2 MOS transistor resistance

Our goal in this section is to measure the *on resistance* and *off resistance* of your power MOS transistors.

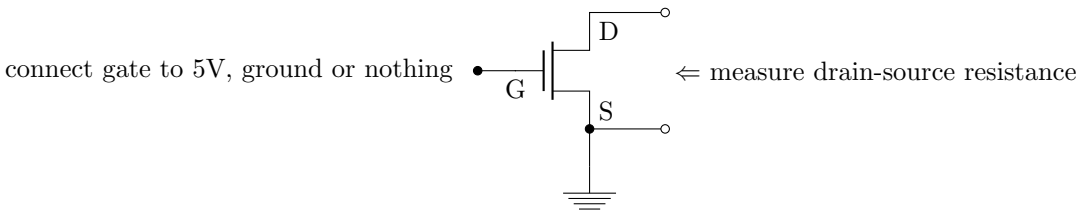
To do this, you'll need to connect a breadboard to a power supply. The easiest way to do this is to use the USB cable you cut in half during lab 1 to power the breadboard directly from any USB charging port (*e.g.* on your computer, a phone charger or your solar charger from lab 1). Alternatively, you could plug your Arduino into the breadboard, connect the Arduino to your computer, and connect the VDD and GND pins of the Arduino to the side rails on the breadboard.

Plug your nMOS power transistor (RFP12N10L) into the breadboard. *Tip: You might find that they plug in easier if you plug the device in so its wide body is parallel with the rows of five, and you push one lead forward and one lead back so the different pins go into different rows, as in the picture below.*





To measure the on and off resistances of the nMOS transistor, we'll want to connect the source to ground, so that we can control the gate-source voltage. The gate connects to 5V, GND or nothing (floating), depending on the measurement. Do **not** connect the drain to anything other than the ohmmeter.

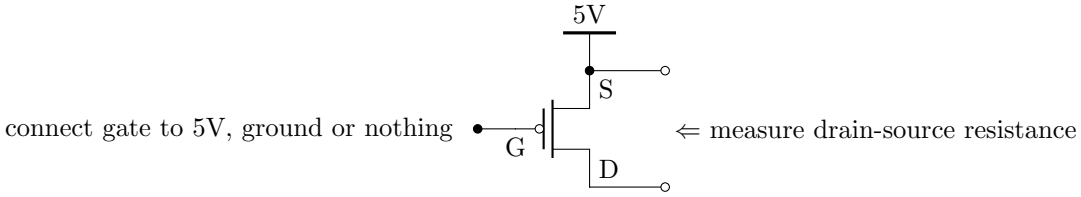


Important measurement technique: All wires have some resistance, including the leads of your multimeter. This means that even when you touch the two probes together, you won't measure zero resistance. Since this is typically less than 1 Ω, it can usually be ignored. But if you are measuring a small resistance, you should measure this baseline resistance first, and subtract it from the resistance you measure.

P6: What is the drain-source resistance of the nMOS transistor when the gate is (a) connected to 5V, (b) floating and (c) connected to GND?

As for earlier questions, the polarity of the ohmmeter matters: you should ensure the test current would go in the "correct" direction for the transistor.

Now, do the same thing with the pMOS (NDP6020P). Recall that pMOS sources connect to V_{DD}.



P7: What is the drain-source resistance of the pMOS transistor when the gate is (a) connected to 5V, (b) floating and (c) connected to GND?

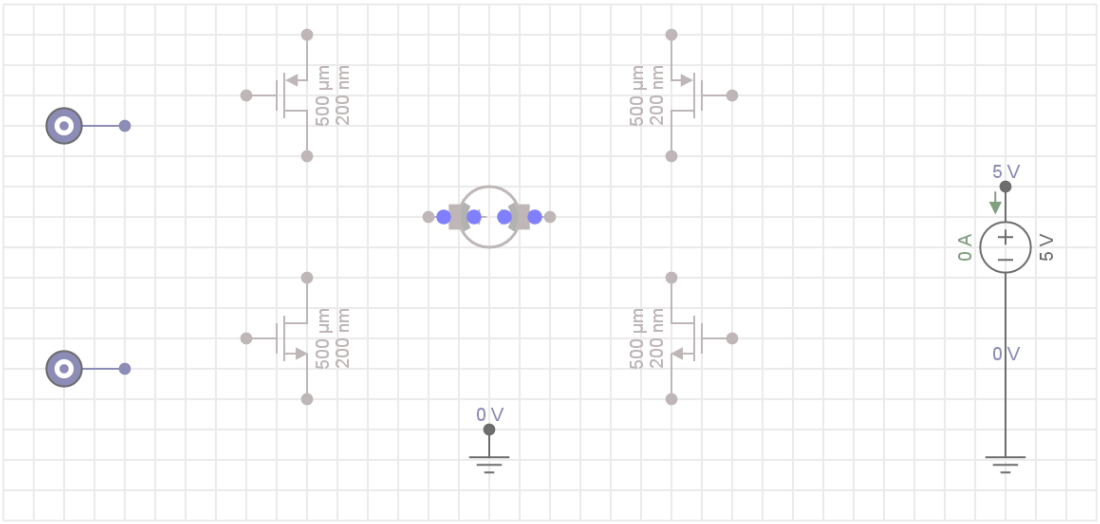
3.3 Designing the circuit

Having established that we need these "power" transistors to interface between the Arduino and the motor, let's design the circuit that drives the motor. The motor needs to be able to do *three* things: go forward, go backward, and stop. This means that we'll need two output pins to drive the motor, via a circuit that uses

power MOS transistors. By changing the states of these output pins (in Arduino code), you should be able get the motor to do each of those three actions.

We've created a starter circuit on EveryCircuit that contains the parts you need. You can search for it using the search term "ENGR 40M prelab 2b starter" (or just "ENGR 40M"), or use this URL: http://everycircuit.com/circuit/5069979055816704.

You should see the circuit shown below:



- The two circles on the left are "logic sources", which represent your Arduino output pins. If you click on them, they'll change from 0 (low) to 1 (high) and vice versa.
- The circular item in the middle is the motor. When current goes through it, it animates either clockwise or counter-clockwise.
- The 5 V source on the right represents V_{DD}.

P8: Complete the circuit on EveryCircuit. By clicking the logic sources into appropriate combinations (in the middle of a simulation), you should be able to get the motor to spin clockwise, spin counter-clockwise and stop. It should also be impossible to short-circuit the 5 V source. Please attach a screenshot of your completed circuit to your prelab submission.

This button might be useful: 

That's most of the work needed to design the circuit. Now we just need to add the switches and be a bit more explicit about how the Arduino fits in.

P9: Draw a complete schematic of the entire circuit you need to build in the lab to make your More Awesome Useless Box. This should include the motor driver, the two switches in your basic useless box, the Arduino, batteries and it should indicate which Arduino pins the motor driver and switches connect to.

Please submit this with your prelab, and bring a copy to lab, since it will guide you as you make your More Awesome Useless Box a reality.

3.4 Programming the Arduino

It makes your life easier as an engineer—and as a maker—if you build things up step by step. This makes debugging much easier: if something doesn't work, you know it should be the thing you just added. As far as the Arduino goes, here's our plan of attack:

1. First, we'll verify that we can read our switches correctly (without touching the motor).
2. Then, we'll verify that we can drive our motor correctly (without touching the switches).
3. Then, we'll write a program that replicates our original useless box, incorporating both the switches and the motor.
4. Finally, we'll extend the program to make the useless box more awesome.

Before you come to lab, we ask you to write two simple programs, for steps 1 and 2 respectively. Once you come to lab and have built the improved box, we'll continue with step 3. Your homework for the week after the lab is to do step 4.

A useful reference for the Arduino language can be found at `https://www.arduino.cc/en/Reference/HomePage`.

A few reminders:

- You need to use `pinMode()` to set whether each pin is an input or an output, and if it's an input, whether the pull-up resistor is enabled.
- You have two output pins to your motor driver circuit, which **both** need to be set whenever you want to change direction or stop the motor.

P10: Write a simple program that reads input from the Serial Monitor and drives the motor forward when it reads `'f'`, backward when it reads `'r'` (for "reverse") and stops the motor when it reads `'s'`. You can use the starter code on the website; it shows how to read input from the Serial Monitor. Please attach it to your prelab submission.

P11: Write a simple program that continually reads each of **two** switches using `digitalRead()` and prints both their states (`"high"` or `"low"`) to the Serial Monitor. Please attach it to your prelab submission.

You've made it! Congratulations! All this work should help reduce the time it takes you to build your More Awesome Useless Box during your lab section.

4 Assembling the Smart Useless Box

As we mentioned in section 3.4, life is easiest if we build the circuit up step by step, checking each step before moving to the next. This helps us catch bugs earlier, when they're part of a simpler circuit and easier to find.

By the way, don't panic if things don't work the first time. Even the most experienced makers don't expect things to work on the first go—in fact, they expect things not to, and build things in stages to make them easier to debug, as we're about to now.

1. **Build the motor driver circuit.**

   Connect an nMOS and an pMOS transistor into half of the motor driver circuit you designed in the prelab (*i.e.*, an inverter), but without the motor. Remember never to leave your gates floating—always have them connected to either VDD or GND before you turn on the circuit!

   Before continuing, you should test that each inverter works by connecting its input to GND, then to VDD, and checking that its output is as you expect in each case.

   Tip: You might find the breadboard easier to manage if you build one inverter on one side of the breadboard, and the other inverter to be a mirror image of the first.

2. **Add the motor.**

   Connect your motor to your motor driver circuit, as you designed it. Be sure that you know which side of the motor is which. Again, by connecting the inputs to the motor driver (inverters) to GND or VDD, verify that you can make your motor go forward, go backward and stop, by changing only whether the inputs are connected to GND or VDD. Double-check that the direction that the motor matches what you expect for each input combination, so that once you connect the Arduino, the motor goes the right way.

   Important: Before continuing, power down the circuit and remove the wires you were using to connect the motor driver inputs to VDD/GND.

3. **Prepare the Arduino.**

   Your Arduino probably doesn't yet have legs, but it came with two pin headers for you to solder on. Break them to the correct length (14 pins), then solder them on. Then, plug your Arduino into the other small breadboard.

   Soldering tip: Getting the Arduino header pins aligned at a right angle to the board can be tricky. The easiest way to do this is to gently push both rows of headers partway into a breadboard, and set the Arduino onto them. This way everything is immobilized, and you can quickly solder all the pins.

   Tip: There are notches on the breadboards that allow you to slide them together, which might make the construction more stable.

Before continuing, plan which I/O pins you're going to connect to your motor driver. It's best to avoid pins 0 and 1, but any of pins 2–13 and A0–A5 will work. Make sure that the program you wrote in question P10 configures your chosen pins to be outputs using `pinMode()`, so that your motor driver won't have a floating gate when you turn on the circuit in step 4. Load this program into your Arduino *before* continuing to step 4.

4. **Connect your motor to the Arduino.**

   The 5V and GND pins on the Arduino can provide power to the rest of your circuit. Connect them to your motor driver appropriately. Connect the I/O pins you chose to the inputs to the motor driver. Remove the fingers from your motor's shaft, so that if anything goes wrong, it won't make your box try to destroy itself.

   Turn your Arduino on (with your P10 program already loaded from step 3) by connecting it to your computer via the USB cable. If it works, great! If not, don't panic—most things don't work first time. Debug your circuit. We've got a debugging guide in section 4.1.

5. **Connect your switches to the Arduino.**

   Connect the switches to the Arduino. Load the program you wrote in question P11 into the Arduino, and verify that it prints the correct states when you play with the switches.

6. **Put it all together!**

   Write the program that controls the useless box, to replicate how the original basic useless box worked.

   You should start by drawing the FSM for the useless box, and then code these into the Arduino using a `switch` statement in the `loop()` function. You'll need to code the transitions and the outputs appropriately. You might find it helpful to have the Arduino constantly print its current state to the Serial Monitor, but it's best to remove such debug lines from your code before you submit your lab report.

   When you think it works as you expect, reattach the fingers to the motor's shaft, and marvel at your (re-)completed useless box.

7. **Improve the box.**

   Once you get the box working again, show your CA that your box works. Now, the fun begins! Write an improved program to control the box. Use your creativity to change how the box behaves. You will need to bring your box into the start of the next lab to show off your box to your group.

   Important note: You must add at least two "tricks" to your box to get full credit for lab completion. These can include (but are not limited to): a shy box that takes longer to hit the switch, an angry box that bobs the finger back and forth before hitting the switch, a mischievous box whose finger "watches" the switch before flicking it, or a box whose "mood" depends on how recently or how often you hit the switch.

8. **Submit your code.**

   Include the program listing for both your basic useless box program (step 6) and your more awesome program (step 7) with your lab report. Make sure that you have used good coding style, and that the code can be easily understood by others.

4.1 A guide to debugging

Experienced makers don't avoid debugging: they just get better at it through experience. To help you get started, here are some ideas for where to start.

(A) If you are trying to use your battery **disconnect your battery and use the USB cable** to power the Arduino and the motor. This supply is more robust, and it also allows you download new programs which will be handy.

(B) Next you should **check your supply voltages** to make sure your circuit has 5V on Vdd. If you don't measure 5V, you probably have messed up the supply routing, or your transistors are shorting out the supply. We suggest you disconnect the supply to the power inverters, and see if that fixes the situation. If it does, you know where to look. **No component should get hot in this design.** If you transistors are warm, you are doing something wrong.

(C) If the supply is fine, you should then **check the input switch connections.** You should change your program to print out on the console the state of the two inputs it is reading, to make sure it matches with what values you think it would read. If you forgot to use the `INPUT_PULLUP` when you set the `pinMode()` you can get very strange results when you read the input (it can depend on your hand position!)

(D) Once the inputs and supply are correct, you should **check the outputs of the Arduino** to see if they are in the right state. This can be easily checked using your multimeter in voltage mode. Once these voltages are correct, you should check the voltages at the output of the motor driver.

5 Analysis

If you want to keep your tricked out useless box and have it run on batteries, you will need to make one more change to the way it is wired. When on, an Arduino takes a little bit of current. While the current draw is not large, it will over time use up the batteries even when you are not playing with the box. It would be better to have the Arduino take no power when no-one is playing with it. It turns out you can make a few small changes to your box to have it behave this way. The key is to power down the Arduino when it is in the "stop" state.

A1: Using only the switches in the box, figure out how to wire the Arduino to the battery so the battery is disconnected from the rest of the circuit in the stopped state, but the Arduino does get power in the forward/reverse states. Draw a schematic of this new design. If you power the Arduino this way, how can you simplify your code?

Note: You do not need to actually build this design unless you want to make your box last longer.

6 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?

7 Build Quality rubric

In this project both your construction and your coding style will be evaluated. In addition, you have a chance to show off your creativity in your tricked out useless box. Your build quality grade considers all of these issues.

Plus

- All solder joints are clean
- Wires are color coded and use the correct type of wire core (stranded/solid)
- Wires are about the right length
- Box internals are easy to follow and nice to look at inside
- Box tops lie flat without additional force (or glue)
- The finger nicely rises and falls hitting both switches almost exactly
- Code is well commented and easy to read/follow
- Code follows a consistent set of style/indenting rules

Check

- Everything fits in the box, but one side of the lid has to be slightly held down
- All solder joints are clean
- Wires are color coded
- Code can be followed with little effort, but would benefit from more comments and/or better structure

Minus

- The box does not consistently work
- Everything fits in the box, but one side of the lid has to be held down
- Marginal solder joints
- Excessive wire lengths demonstrate lack of planning
- Wires are not color coded
- Code is highly unorganized and difficult to follow