Lab 3 – Motors

Overview:

In this lab, you will learn to drive DC motors, servos, and stepper motors, and create the software to control them. You will learn to deal with inductive kickback, and how to snub it. You will learn to drive a stepper motor directly using an H-bridge, and also using a dedicated driver board.

Comments:

Lab checkoff:

- Check off each part with a tutor or TA to verify functionality and each team member's understanding. **You need to print out the last page to get signed off. Include this sheet in your lab report.**
- Strongly recommended to check off each part before moving to the next.

Lab3 specific comments:

- Use ECE-118 motor drive boards and high current driver boards for your project. These are robust but not abuse-proof.
- Utilize the Uno32 stack (I/O board on top, Uno32 in the middle, power distribution board on the bottom). Always power circuits from the fused power distribution board to prevent damage.
- ATX power supplies provide 12V, 5V, and 3.3V rails. Handle with care as they can source several amps. The ATX will save itself, but not likely your circuit.
- Plan thoroughly for each part, day, and week to avoid delays and errors.
- Prepare before starting lab work to save time and reduce mistakes.
- Test hardware and software separately to isolate issues.
- If stuck on a task for over an hour, seek help from peers, tutors, or TAs.
- Create a cable to connect boards and the Uno stack for convenience.Jumper wires are easy to get wrong. Make power connectors for the power distribution board.

Bumps and Road Hazards:

This lab involves high currents, which can be dangerous for both you and your hardware. Mistakes can blow traces off your board, fry chips, or melt plastic, causing delays and frustration. Maintaining good "electronic hygiene" is **CRITICAL** to your success.

Good practices include:

- **ALWAYS TURN OFF** power before modifying wiring. Never plug or unplug components with the power on.
- Use banana plugs for stable power connections. Avoid alligator clips.

- **Triple-check** polarity before connecting power. Color-code your rails (GND, 3.3V, 5V, 12V) for clarity.
- Keep wires and connectors tidy to prevent shorts. Maintain a clean workspace.
- Inspect boards for solder bridges or stray metal that could cause shorts.
- Make reliable connectors:
  - Ensure proper length and cover joints with heat shrink.
  - Test connections with a multimeter in continuity mode.
  - Cross-test to confirm correct polarity. The black lead on one end should connect to the other black lead, AND should not connect to the red lead.
- Diagnose errors before proceeding.
- Avoid frustration. Take breaks, ask for help, and communicate with your lab partner.
- Plan for user error. Keep a lab notebook for notes and instructions.

  Skipping these steps may save time initially but can lead to costly mistakes. Taking an extra 15 minutes now can prevent significant setbacks later.

PreLab:

1. Select a partner and join a group on CANVAS. For each lab, you must join a new group (preferably someone you haven't worked with before). **Navigate to People → Lab3 Group and register with your teammate.** If you need assistance finding a partner, use Piazza to connect with others. If you do not select a partner by the prelab deadline, one will be randomly assigned to you.
2. Complete prelab exercises and submit a PDF file individually on CANVAS. Follow the requirements detailed in each section. Part0, Part2.

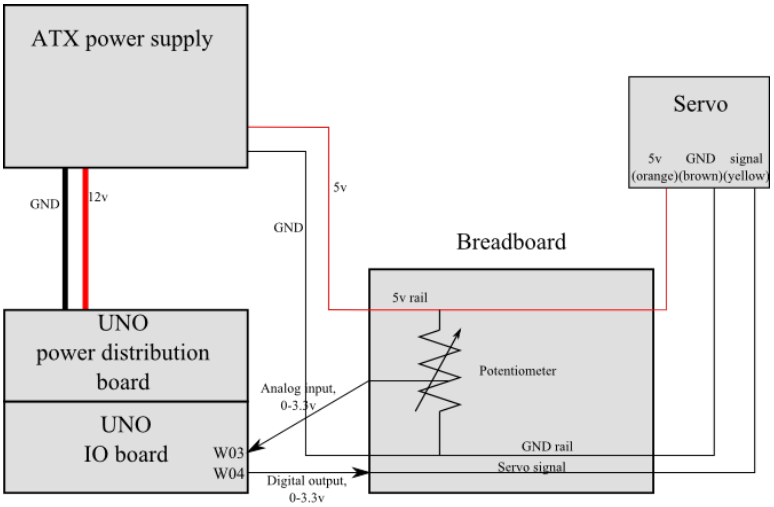
Guide for Prelabs: Create a Block Diagram for each part of the lab. Your block diagram should communicate the following information:

- Names of the pins on the boards you are using (both on the Uno and on the daughter boards). For the Uno, these should be of the form (Port X Pin 3 or X3).
- An arrow indicating whether an Uno pin is an input or an output (or both).
- Your power source (12V rail, 5V rail, power distribution board, etc.).

On the same page as each diagram, also include:

- Why you chose this port for that part of the lab.
- What, if any, initialization code you will need for the port. We want the literal code that you will execute: functions(), PIN_NAMES, and semicolons. You shouldn't need more than 4-6 lines per lab part. Look at the test code at the end of the libraries if you need inspiration.
- The type of signal each wire is carrying (analog, digital, etc.)

Below is an example block diagram for part 1 (which you still need to modify) to show you how these should look:



While you are picking out Uno pins and working out your breadboard layout, keep in mind that you will probably come back to many parts in this lab. You will find it advantageous to work out a system that makes it easy to switch between connection setups (the ribbon cable connectors are particularly useful for this).

Part 0 – Preparation for Lab

Overview:

This lab, more than the others that you have done so far requires preparation for a smooth time through. It is paramount that you take the time to read through the entire document, and to prepare each section so that you have a plan of what you are going to do and how.

Reference Material:

- ECE-118 Uno32 IO Stack Documentation
- Header files in the `ece118_base` directory
  - In particular `pwm.h`

PreLab:

Q1: What one thing and one thing only will you stick into the 14-pin connectors on the boards?

Q2: Draw a detailed state diagram for a hypothetical software PWM driver. Include an example plot of the expected output signal for a given frequency and duty cycle. **HINT**: The transitions between states should be triggered by timer events.

Tasks:

- Read through the entire lab and develop the drawings, schematics, block diagrams, and state diagrams asked for in the pre-lab section.
- Ensure that you maintain copies of these for yourself, as you will be referring to them often.
- Note that there is nothing to turn in with the final report as all of this goes in the prelab.

Part 1 – Driving an RC Servo

Overview:

You will be driving a standard RC servo using software control. The RC Servo is used in remote controlled applications (such as airplanes, boats, and cars) and is set up to have a large compound gear train and a feedback mechanism to accurately track a position command. The command is given as a pulse width modulated signal, and repeats at 50Hz.

Reference Material:

- ECE118_Uno32_IO_board documentation
- CKO Ch. 27 (especially 27.3)
- RC Servo Tutorial from Society of Robotics
- AD.h from the `ece118_base` directory
- LED.h from the `ece118_base` directory
- RC_Servo.h from `ece118_base` directory
- Appendix A on LDOs

PreLab:

A schematic of how you are going to hook up your potentiometer (remembering to keep the voltage into the Uno32 between 0 and 3.3V). A block diagram of the RC Servo setup similar to notes in *Overview-Prelab* section with all things labelled carefully.

Bumps and Road Hazards:

- **DO NOT** create a 5V or 3.3V rail using voltage dividers from the 12V rail. Use the ATX power supply's dedicated rails.
- The servo has three wires: ground, a **MAXIMUM** of 5V for power, and a control signal. **DO NOT CONNECT THE SERVO TO A HIGHER VOLTAGE**.
- **UNO PINS CAN BE DAMAGED BY MORE THAN 3.3V**. Ensure voltage dividers or regulators keep inputs below 3.3V.

- **VERIFY CONTROL SIGNAL**: Use an oscilloscope to confirm a square pulse ($\leq$2ms width) on the control pin before connecting the servo.
- Ensure a common ground connection between the Uno32 stack, servo, and power supply (via ATX or power distribution board).

Tasks:

- Drive the RC Servo under software control from your Uno32.
- Use a potentiometer to feed a voltage into an analog pin and scale the RC Servo pulse output based on that voltage.
- Display a scaled version of the potentiometer input value on the LED bars of the IO Stack.
- To power RC Servo, connect the ATX power supply's 5V output directly to your breadboard or use a 5V regulator from the 12V line with filtering caps.
- You do not use any control daughter boards for this part of the lab. Use UNO32 stack and your circuit.
- Create a new MPLABX project and examine the RC_Servo.h library header.
- Control the RC Servo, which uses a PWM signal (1.5ms high time is centered) to slew to and hold a commanded position.
- Ensure the servo moves from one end of its range to the other based on the potentiometer's minimum to maximum value. The servo should move +/- 40-60 degrees,
- Do **NOT** alter the MINPULSE or MAXPULSE values in the header file.
- Experiment the following (You may discover that commanding the servo directly from your software rather than using the potentiometer for commands will work better in these experiments):
  - Minimum change in pulse high time that results in servo motion.(The module allows you to control the pulse width to 1uS resolution).
  - Servo range in degrees.
  - Maximum angular velocity.
  - Minimum angular velocity for smooth motion.
- Demonstrate to TA/tutors:
  - Control over the servo.
  - LEDs tracking the potentiometer.
  - Demonstration of experiments mentioned above.
- Include in the final report:
  - Annotated scope traces of the servo control pin output.
  - Code snippets.
  - Block diagrams.
  - Observations and writeup including the experiments.

Part 2 – Unidirectional Drive of a DC Motor

Overview:

This assignment will drive a small DC motor in a single direction under software control using the DS3658 high current peripheral driver. You will use the supplied PWM software module to drive the motor, and demonstrate that you can control the speed of the motor by reading the input of the potentiometer.

Reference Material:

- ECE118_Uno32_IO_board documentation
- ECE118_DS3658 documentation
- CKO Ch. 8
- AD.h from the `ece118_base` directory
- LED.h from the `ece118_base` directory
- PWM.h from the `ece118_base` directory
- Appendix A on LDOs

PreLab:

A schematic of how you are going to hook up your potentiometer (remembering to keep the voltage into the Uno32 between 0 and 3.3V). A block diagram of the DS3658 and DC Motor setup as noted in Part 0 with all things labelled carefully. Include your initialization code for the software modules you will write.

Bumps and Road Hazards:

- You will need to make a power connector to power the DC3658. Color-code the wires, use heat shrink on both ends of the connector, and triple-check the polarity.
- The high voltage side of the motor should be sourced directly from the UNO power distribution board, using a single-wire power connector you will need to make just for this purpose.
- **THE UNO PINS CAN BE DAMAGED BY MORE THAN 3.3V**. Be sure that your voltage divider (and anything else you ever connect to the Uno32) doesn't exceed 3.3V in any situation (anticipate user error). From the 5V rail, use a regulator or pull the 3.3V rail from the ATX directly.
- **DO A SCOPE TRACE OF THE OUTPUT FROM THE CONTROL PIN ON THE UNO** BEFORE YOU CONNECT THE DS3658. You should see a PWM signal, with the duty cycle varying with your potentiometer. If you do not see this, DO NOT connect the DS3658 to that signal!
- Before you drive the motor, be sure your DC3658 board is operating in **CLAMPED MODE**! If you fail to do so, kickback could permanently damage the board.

- Your Uno32 stack and the DS3658 need to have a common ground connection back to the power supply. This can be directly to the ATX, or through the power distribution board.

Tasks:

- Control a DC motor using a DS3658 High Current Peripheral Driver capable of sinking up to 600mA per output.
- Read the input from a potentiometer directly into the Uno32 to control motor speed (unidirectional control).
- Drive a PWM signal from the Uno32 to the DS3658 to control the motor.
- Output the scaled value of the potentiometer on the LEDs of the Uno32 stack:
  - 0V corresponds to 0 LEDs and 0% duty cycle.
  - 3.3V corresponds to 12 LEDs and 100% duty cycle.
- Use a ribbon connector or wires soldered to male-male pins to connect the IO ports on the Uno32 to the DS3658 inputs.
- Connect the DS3658 to the low voltage side of the DC motor, with the high voltage side sourced directly from the UNO power distribution board using a single-wire power connector.
- Make a power connector to power the DC3658:
  - Color-code the wires.
  - Use heat shrink on both ends of the connector.
  - Triple-check the polarity.
- Verify the potentiometer output with a multimeter to ensure it never exceeds 3.3V before connecting it to the UNO IO board.
- Ensure the DC3658 board is operating in clamped mode before driving the motor to prevent damage from kickback.
- Run the motor at 20% and 80% duty cycles and take oscilloscope traces of the motor voltage:
  - Identify different areas of the drive, inductive kickback, snubbing, and other features of interest on the trace.
  - Optionally include the PWM as another signal on the trace.
- Demonstrate to the TA/tutors:
  - Control over the DC Motor.
  - LEDs tracking the potentiometer.
  - Code functionality for a checkoff.
- Include in the final report:
  - Annotated scope traces of the PWM control pin output.
  - Traces of the 20%/80% tests.
  - Interesting code snippets.
  - Block diagrams.
  - Observations and a writeup of findings.

Part 3 – Snubbing the Inductive Kickback

Overview:

In this assignment, you will explore the use of diodes and Zener diodes in different configurations to snub the inductive kickback on the DC motor. The DC motor will still be under software control identically to the previous part. You will also explore the limits of the PWM drive to the DC motor, and how the snubbing arrangement will affect those limits.

Reference Material:

- ECE118_Uno32_IO_board documentation
- ECE118_DS3658 documentation
- CKO Ch. 22, 23, 24, 26, and 27
- AD.h from the `ece118_base` directory
- LED.h from the `ece118_base` directory
- PWM.h from the `ece118_base` directory

PreLab:

Block diagram of the set up you will be using with everything labeled. Identify which of the schematics in the outline section corresponds to the DS3658 in CLAMP mode.

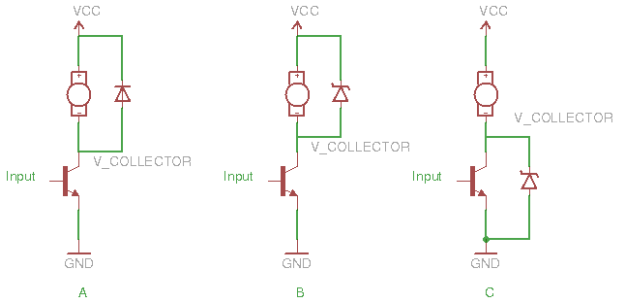
Bumps and Road Hazards:

- You will need to make a power connector to power the DC3658. Color-code the wires, use heat shrink on both ends of the connector, and triple-check the polarity. The high voltage side of the motor should be sourced directly from the UNO power distribution board, using a single-wire power connector you will need to make just for this purpose.
- **THE UNO PINS CAN BE DAMAGED BY MORE THAN 3.3V**. Be sure that your voltage divider (and anything else you ever connect to the Uno32) doesn't exceed 3.3V in any situation (anticipate user error). From the 5V rail, use a regulator or pull the 3.3V rail from the ATX directly.
- **DO A SCOPE TRACE OF THE OUTPUT FROM THE CONTROL PIN ON THE UNO BEFORE YOU CONNECT THE DS3658.** You should see a PWM signal, with the duty cycle varying with your potentiometer. If you do not see this, DO NOT connect the DS3658 to that signal!
- As you will be adding in clamp diodes in different parts of your circuit, it is extremely important to only make changes to your circuits with the **POWER OFF**! Electronic hygiene is very important here.
- The Zener diode is NOT a normal diode (it goes into reverse conduction easily at a specific voltage). Make sure you use the Zener diode in the circuit where you need it, and a normal diode where you need it.

- To measure the waveform across the motor, don't connect the probe's ground lead to $V_{collector}$—you will cause a short across the DS3658 through the o-scope.
- Your Uno32 stack and the DS3658 need to have a common ground connection back to the power supply. This can be directly to the ATX, or through the power distribution board.

Tasks:

- Explore inductive kickback in a motor and how to snub it using various diode placements in the drive path of the motor. The hardware and software setup should be identical to the previous part, when you controlled the DC motor in a unidirectional manner using the DS3658.
- Investigate the limits of the PWM drive to the DC motor and how the snubbing arrangement affects those limits. Refer to the PWM.h header file for details on inherent limitations (maximum frequency and duty cycle resolution).
- Use a Zener diode in schematic C and a normal diode in schematic A. Ensure proper identification and placement of the diodes before applying power.



- Study the waveforms generated when the DC motor is hooked up with different snubbing arrangements:
  - Capture oscilloscope traces of the motor's waveform at intermediate, high, and low duty cycles.
  - Observe and explain differences between the circuits.
  - Identify and quantify the kickback peak and decay time for each circuit.
- Measure the waveform across the motor indirectly:
  - Connect the oscilloscope's ground lead to ground and the high lead to $V_{collector}$.
  - Avoid connecting the probe's ground lead to $V_{collector}$ to prevent shorting the DS3658 through the oscilloscope.
- Compare the decay time of each circuit and adjust the timescale for accurate measurement.
- Determine the frequency and duty cycle limits for each circuit. Identify conditions where the motor's speed is a linear function of the duty cycle.
- Add resistive torque to the motor (e.g., grip the shaft with fingers or pliers) and observe the effect on the waveform. Explain the observed behavior.
- Demonstrate the various waveforms live on the oscilloscope to the TA/tutors for checkoff. Be prepared to answer questions about the observations.

- Include the following in the lab report:
  - Pictures of the waveforms, clearly labeled.
  - Detailed discussion of observations and insights.

Part 4 – Bidirectional Control of a DC Motor

Overview:

This assignment is control a DC motor in both directions using an H-bridge. The input to the Uno32 to determine speed is the same potentiometer that you have been using in the previous parts, but direction will be determined by a switch.

In the previous two parts, you could only make the motor spin in a single direction. In this part you will be able to reverse directions using an H-bridge module. We will be using small commercially available L298 modules (the red boards). You will need to supply enable and direction inputs as well a PWM signal.

You will create a new MPLABX project to control the motor bi-directionally in response to the switch and potentiometer.

Reference Material:

- ECE118_Uno32_IO_board documentation
- ECE118_L298 Dual H-Bridge documentation
- CKO Ch. 22, 23, 24, 26, and 27
- AD.h from the `ece118_base` directory
- LED.h from the `ece118_base` directory
- PWM.h from the `ece118_base` directory
- Appendix B: Hooking up a switch

PreLab:

Block diagram of the set up you will be using with everything labeled. Add in the initialization code that you need for your project.

Bumps and Road Hazards:

- You will need to make a power connector to power the L298 module. Color-code the wires, use heat shrink on both ends of the connector, and triple-check the polarity.
- **THE UNO PINS CAN BE DAMAGED BY MORE THAN 3.3V**. Be sure that your voltage divider (and anything else you ever connect to the Uno32) doesn't exceed 3.3V in any situation (anticipate user error). From the 5V rail, use a regulator or pull the 3.3V rail from the ATX directly.

- **DO A SCOPE TRACE OF THE OUTPUT FROM THE CONTROL PIN ON THE UNO BEFORE YOU CONNECT THE L298.** You should see a PWM signal, with the duty cycle varying with your potentiometer and switch. If you do not see this, DO NOT connect the L298 to that signal!
- It is extremely important to only make changes to your circuits with the **POWER OFF**! Electronic hygiene is very important here.
- Your Uno32 stack and the L298 need to have a common ground connection back to the power supply. This can be directly to the ATX, or through the power distribution board.

Tasks:

- Control a DC motor in both directions using an H-bridge.
- Use the same potentiometer as in previous parts to determine speed.
- Use a switch to determine the direction of the motor.
- Utilize the L298 Dual H-Bridge Driver Module board.
- Supply enable and direction inputs along with a PWM signal.
- Create a new MPLABX project to control the motor bi-directionally.
- Follow Appendix B for hooking up a switch to the Uno32.
- Map 0V to 0% duty cycle PWM and 3.3V to 100% duty cycle PWM.
- Use the switch to control the motor's direction.
- Display speed and direction on LEDs using bit shifting and masking.
- Determine the frequency limits of operation and compare them to parts 1 and 2.
- Add a load to the motor (e.g., fingers or pliers) and observe the behavior.
- Demonstrate the motor drive software and hardware to TA/tutors for checkoff.
- Include in the final lab report:
  - A picture of the setup.
  - Snippets of interesting code.
  - A detailed explanation of findings and observations.

Part 5 –Control of a Stepper Motor

Overview:

This assignment is write software to control a Stepper motor using an H-bridge. You will explore the step rate at which it is possible to drive the stepper motor, and the various methods of driving the steppers.

You will create a new MPLABX project to control stepper motor using your software, and try various stepper driving techniques and experiment with them.

Reference Material:

- ECE118_Uno32_IO_board documentation
- ECE118_L298 Dual H-Bridge documentation
- CKO Ch. 26
- Stepper.h from the `ece118_base` directory

PreLab:

Block diagram of the set up you will be using with everything labeled. Add in the initialization code that you need for your project. Additionally State machine diagrams to drive a stepper motor in Half-step, Full-step, and Wave modes.

Bumps and Road Hazards:

- You will need to make a power connector to power the L298. Color-code the wires, use heat shrink on both ends of the connector, and triple-check the polarity.
- The stepper motor has four wires, one for each end of two coils. Use a multimeter or find the datasheet to determine how they are paired.
- **THE UNO PINS CAN BE DAMAGED BY MORE THAN 3.3V**. Be sure that your voltage divider (and anything else you ever connect to the Uno32) doesn't exceed 3.3V in any situation (anticipate user error). From the 5V rail, use a regulator or pull the 3.3V rail from the ATX directly.
- **DO A SCOPE TRACE OF THE OUTPUT FROM THE CONTROL PINS ON THE UNO BEFORE YOU CONNECT THE L298.** You should see a stepper signal. If you do not see this, DO NOT connect the DRV8814 to that signal!
- It is extremely important to only make changes to your circuits with the **POWER OFF**! Electronic hygiene is very important here.
- Your Uno32 stack and the L298 need to have a common ground connection back to the power supply. This can be directly to the ATX, or through the power distribution board.

Tasks:

- A stepper motor is a clever arrangement of coils and magnets such that energizing the coils in the correct sequence causes the motor to turn in one direction or the other a fixed amount each step (hence the name).
- You are going to control a stepper motor using software and the L298 H-bridge. That is, you are going to manually control the coils individually in order to switch the stepper coils to make it go forwards and backwards in Full Step, Wave, and Half-Step drives.
- Begin with the Stepper.c/h module, which drives the stepper in Full Step drive mode. Note that this is designated by the #define FULL_STEP_DRIVE in the header file, and that only one of the #defines should be defined at any one time.

- The stepper motor has four wires, one for each end of two coils. Use a multimeter or find the datasheet to determine how they are paired.
- Determine the maximum step rate you can start, and the maximum rate at which you can step without losing any steps (try marking the shaft and rotating forwards and backwards 360 degrees to see if the marker creeps).
- Investigate whether these rates are changed by:
  - Loading the stepper (e.g., applying fingers to the shaft).
  - The drive method (Full Step, Wave, Half-Step).
- For bragging rights, demonstrate that you can successfully enter the pull-in region of stepper motor operation without losing steps (refer to the CKO chapter on steppers for details).
- Demonstrate your software and stepper drive to the TA/tutors for checkoff:
  - Change your #define and recompile to show the stepper working in all modes of drive.
  - Hard code the sequence of steps forwards/backwards in your code.
  - Replicate how you determined the maximum step rate for each mode of driving for the TA/tutors.
- Include in the report:
  - Interesting snippets of code.
  - A description of what you discovered about your stepper limits.

Part 6 –Stepper Motor using Dedicated Board

Overview:

This assignment is write software to control a Stepper motor using a dedicated stepper driver board. You will explore the step rate at which it is possible to drive the stepper motor, and the various methods of driving the steppers using the dedicated hardware.

You will create a new MPLABX project to control stepper motor driver board using your software, and try various stepper driving techniques and experiment with them.

Reference Material:

- ECE118_Uno32_IO_board documentation
- ECE118_DRV8811 Stepper Board documentation
- CKO Ch. 26
- Stepper.h from the `ece118_base` directory

PreLab:

Block diagram of the set up you will be using with everything labeled. Add in the initialization code that you need for your project.

Bumps and Road Hazards:

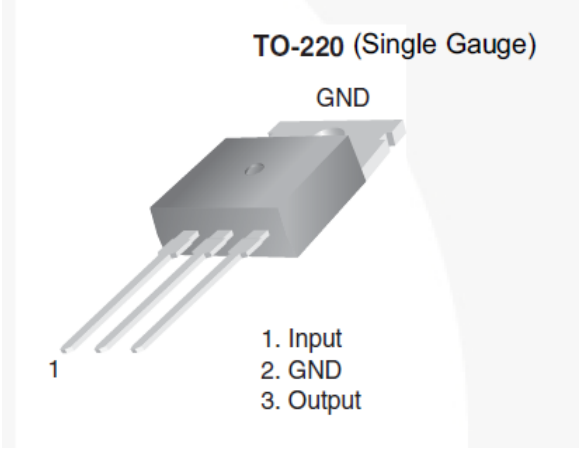
- You will need to make a power connector to power the DRV8811. Color-code the wires, use heat shrink on both ends of the connector, and triple-check the polarity.
- The stepper motor has four wires, one for each end of two coils. Use a multimeter or find the datasheet to determine how they are paired.
- **THE UNO PINS CAN BE DAMAGED BY MORE THAN 3.3V**. Be sure that your voltage divider (and anything else you ever connect to the Uno32) doesn't exceed 3.3V in any situation (anticipate user error). From the 5V rail, use a regulator or pull the 3.3V rail from the ATX directly.
- **DO A SCOPE TRACE OF THE OUTPUT FROM THE CONTROL PINS ON THE UNO BEFORE YOU CONNECT THE DRV8811.** You should see a stepping signal. If you do not see this, DO NOT connect the DRV8811 to that signal!
- It is extremely important to only make changes to your circuits with the **POWER OFF**! Electronic hygiene is very important here.
- Your Uno32 stack and the DRV8811 need to have a common ground connection back to the power supply. This can be directly to the ATX, or through the power distribution board.

Tasks:

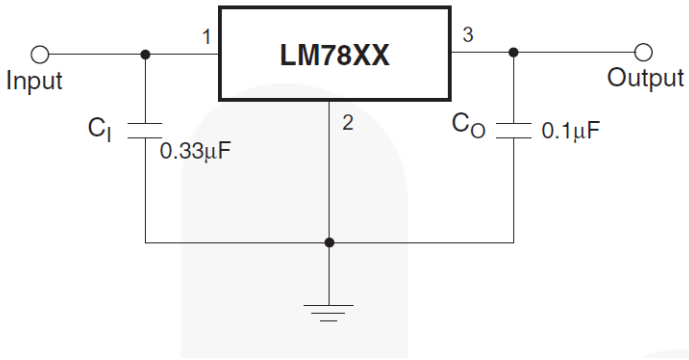
- Control a stepper motor using software and the DRV8811 Stepper Driver Module.
- Use a dedicated board to sequence the coils, requiring only steps, direction, and enable signals.
- Begin with the Stepper.c/h module and #define DRV8811_DRIVE.
- Use a multimeter or datasheet to determine how the stepper motor's four wires (two coil ends) are paired, and connect them to the DRV8811.
- Determine the maximum step rate you can start and the maximum rate at which you can step without losing steps:
  - Mark the shaft and rotate forwards and backwards 360 degrees to check for marker creep.
- Investigate whether these rates are affected by:
  - Loading the stepper (e.g., applying fingers to the shaft).
  - Drive method (adjust jumpers on the DRV8811 for current limits and step modes).
- Demonstrate your software and stepper driver functionality to TA/tutors for checkoff:
  - Replicate how you determined the maximum step rate.
  - Hard code the steps in your software for demonstration.
- Include in the report:
  - Interesting snippets of code.
  - A description of findings about stepper limits when using the dedicated board.

Appendix A: LDO Regulators

The basic regulator used in most circuits is a fixed output LDO (low drop-out). This is a three terminal device with an input, ground, and output. The input is the supply voltage, which must be above the output voltage by the dropout voltage. As long as the voltage on the input is above the output + dropout voltage, the output will remain stable at the fixed voltage. Typically a diode is added inline to the input to ensure that if you reverse polarity on the input and ground, no current will flow (reverse polarity protection). See below for the pinouts for the TO-220 package of the 7805 regulator.



The regulators in your kits are 7805 (Fixed output at 5V) and can supply up to 1A of current. Note that they have a power rating, and will heat up approximately 70°C for every watt you pull through (they have an over current and thermal shutdown built in). If you are drawing a lot of current through, you will need to add a heatsink.

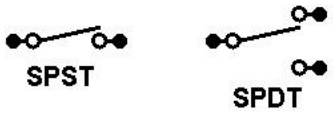


For stability, the regulator requires both an input filtering capacitor as well as an output filtering capacitor. Typically a 0.22uF cap on the input and a 0.1uF cap on the output will ensure that the regulated voltage stays good even with changes in the load.

Adding a regulator and reverse polarity protection to your circuit makes your circuit robust to variable inputs, and errors in hooking up power and ground. Use regulators with reverse polarity protection in every circuit module you make.

Appendix B: Hooking Up A Switch

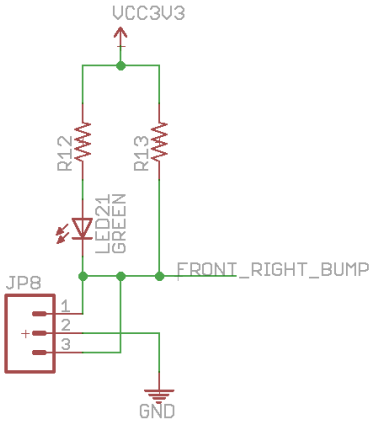
The basic switch is used almost everywhere (bumpers, etc.) to indicate to the software that it has been pushed. When determining if the switch has been pushed, a clean signal needs to be sent to the pin on the microcontroller. For a single switch, we will simplify it to a single pole, single throw (SPST) or a single pole, dual throw (SPDT) switch.



A SPST switch is a two terminal device, which is open when the switch is open, and continuous when the switch is closed (that is, the switch makes or breaks continuity). These are, for instance, momentary push switches, or a normal toggle switches with only two wires.

A SPDT switch has three terminals, one marked common, one normally open, and one normally connected. The switch connects common to normally connected, and when it is thrown, it connects common to normally open.

In both cases, the challenge is to drive a signal into the IO pin that moves from 0V to 3.3V in order to detect the switch has changed states. Note that if you are simply trying to determine that the switch has been pressed, then both kinds of switches are hooked up the same way:



The common pin is hooked to ground, and the other side (of a two terminal switch) or the normally open (in a three terminal switch) which is then run directly into the pin. A pullup resistor is connected to the pin to 3.3V (with an optional LED and resistor to indicate the switch is pushed).

When the switch is open, there is no current flow and the LED stays off and the pin reads a "HIGH" state at 3.3V. When the switch is closed, the LED will illuminate and the pin will read a "LOW" state at 0V. Note that the LED is not required, and can be omitted from the circuit, but is often quite useful to indicate what is going on.

Checkoff and Time Tracking

Student Name:_____ CruzID_____@ucsc.edu

Time Spent out of Lab	Time Spent in Lab	Lab Part - Description
		Part 0 – Preparation for Lab
		Part 1 – Driving an RC Servo
		Part 2 – Unidirectional Drive of a DC Motor
		Part 3 – Subbing the Inductive Kickback
		Part 4 – Bidirectional Control of a DC Motor
		Part 5 – Control of a Stepper Motor
		Part 6 – Stepper Motor Using Dedicated Board

Checkoff: TA/Tutor Initials	Lab Part - Description
	Part 1 – Driving an RC Servo
	Part 2 – Unidirectional Drive of a DC Motor
	Part 3 – Subbing the Inductive Kickback
	Part 4 – Bidirectional Control of a DC Motor
	Part 5 – Control of a Stepper Motor
	Part 6 – Stepper Motor Using Dedicated Board