UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE118(218)/L: INTRODUCTION TO MECHATRONICS

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:

This lab will introduce you to some of the CMPE-118 motor drive boards and high current driver boards that you will be using in your project. These are designed to be robust; however nothing we design can be fully protected against abuse. These boards will be used in your project to control your motors.

You will be using the Uno32 stack, with I/O board on top, Uno32 in the middle, and a power distribution board on the bottom. It is very important that you power all your circuits from the power distribution board as it is fused and will save your circuits if you have an error in them.

You will also be using ATX power supplies that have been set up to source 12V, 5V, and 3.3V rails. These are rugged and very clean power supplies. The ATX will source several amps on those rails—be careful with this (the ATX will save itself, but not likely your circuit).

Come into the lab with a plan: a plan for each part; a plan for the day; a plan for the week.

This lab is about preparation. If you try to do your prep work at the same time as you do the lab, you will take MUCH longer and break more things. Prepare first, and then do the labwork.

Test your hardware and software separately.

If you have spent more than one hour on any single task, you are probably not on the right path. STOP and ask a neighbor, tutor, or TA to take a look at what you are doing.

Make a pair of 14-wire ribbon cables. This is not required but it will save much time in the long run. Make a pair of power connectors to connect the board you are using to the power distribution board.

PRELAB:

Choose a partner to do the lab with, and join a group together in the “Lab 2 Group XXX” category on CANVAS. For instructions on how to do this, see the CMPE118_LabSubmission document on the website. If you have not chosen a partner by the day before the prelab is due, we will randomly assign one to you (most likely on the day the prelab is due). Note that Piazza is an excellent way to find partners.

Each part of the lab has prelab exercises. Complete these by yourself (you are welcome to collaborate with your teammate, but the work should be your own) and submit them using the assignment submission on the CANVAS website. The requirements for the prelab deliverables are detailed in each section, make sure you read the lab carefully (more than once) and answer them all.

Note that you need to electronically submit your prelab and lab report files in a very specific format. See the CMPE118_LabSubmission document on the class website for instructions on how to submit your files and how to verify that you have done so.

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:

- CMPE-118/L Uno32 IO Stack Documentation
- Header files in the C:/CMPE118 include directory
- Stepper.h header file in the C:/CMPE118 include directory

PRELAB:

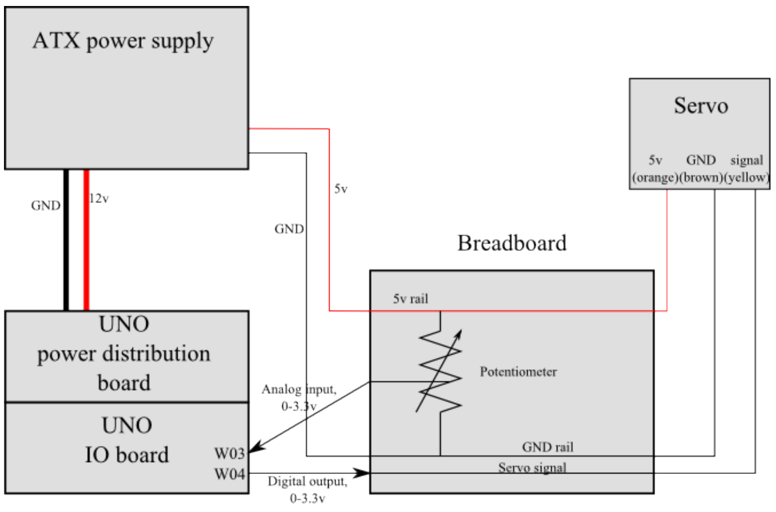
Create a Block Diagram for each part of the lab. Your block diagram should communicate the following information:

- 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).

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

Draw a detailed state diagram for a 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.

BUMPS AND ROAD HAZARDS:

This lab involves high currents. High currents are dangerous, both for you and for your hardware. 10 amps on the wrong wire can blow the traces off your board, fry any chip, and melt plastic. Silly mistakes and accidents can easily destroy days of hard work or cause days of delays. This not only endangers your schedule, it also produces pain, stress, sleeplessness, frustration, and dismay. Therefore, maintaining good "electronic hygiene" is **CRITICAL** to your success and sanity.

Good electronic hygiene practices include:

- **NEVER, NEVER, NEVER** plug things in or pull things out of your board with the power supply turned on.
- Use banana plugs to connect power. They are stable and secure—there's a reason all power supplies use banana connectors! Do not use alligator clips.
- Check your polarity **AT LEAST THREE** times before hooking up power to things. Color-code everything to facilitate this process. You'll have four rails in this lab (GND, 3.3v, 5v, and 12v). Give each one a different color.
- Be very careful about wires dangling from power supplies or O-scope leads shorting things out. Every bit of bare metal is a risk, so take the time to tuck away lose connectors.
- Always inspect your boards for solder bridges and the like. Little flecks of solder or wire can have big consequences.
- Take the time to make your connectors well.
    - Make sure they're a comfortable length.
    - Make sure you have heat shrink covering every joint—bare metal is a recipe for sparks and shorts.

    - Test your wires and connections with a multimeter on continuity test mode (it is usually depicted with a diode and a sound wave)
    - Don't forget to cross test—the black lead on one end should connect to the other black lead, AND should not connect to the red lead.
- Diagnose errors before continuing.
- Avoid frustration. Frustration can impair your decision-making. If you are having difficulty, ask for help. Breathe. Take a walk outside. Communicate thoroughly and kindly with your lab partner.
- Anticipate user error. You and your partner will be tired, frustrated, or rushing at some point during this lab; ensure you have a way to know what you are supposed to be doing.

It is often tempting to skip some of these practices. If you find yourself feeling such temptation, imagine what you will feel when something important explodes, fries, or melts. It is worth taking an extra 15 minutes now to lower the odds of that happening later.

OUTLINE:

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

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:

- CMPE118_Uno32_IO_board documentation
- CKO Ch. 27 (especially 27.3)
- RC Servo Tutorial from Society of Robotics
- AD.h from the C:\CMPE118 include directory
- LED.h from the C:\CMPE118 include directory
- RC_Servo.h from the C:\CMPE118 include directory
- Appendix A on LDOs from Lab 2

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 as noted in Part 0 with all things labelled carefully.

BUMPS AND ROAD HAZARDS:

The servo has three wires. One of these should connect to ground, one should connect to a **MAXIMUM** of 5v, and the third wire carries a control signal. **DO NOT CONNECT THE SERVO TO A HIGHER VOLTAGE, YOU WILL DESTROY THE SERVO**.

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 SERVO. You should see a square pulse of no more than 2ms width. If you do not see this, DO NOT connect the servo to that signal!

Your Uno32 stack and the servo 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.

OUTLINE:

You are going to drive the RC Servo under software control from your Uno32. You will have a potentiometer that feeds a voltage into an analog pin, and based on that voltage you will scale the output of your RC Servo pulse. In addition, you will display a scaled version of the value of your input from the potentiometer on the LED bars on the IO Stack.

You may connect the ATX power supply's 5v output directly to your breadboard. Alternatively, you may use a 5v regulator from the 12v line and make your own 5v rail.

You will not use any of the daughter boards for this part of the lab. Make a new MPLABX project following the instructions from Lab 0, and examine the RC_Servo.h library header.

The RC Servo is a simple device with a motor, a gearbox, and a feedback mechanism. It gets power (5V ONLY) and ground, and a command signal that is a pulse width modulation (1.5ms high time is centered). The servo will slew to the commanded position, and hold position.

You will be creating an MPLABX project to do so (refer back to Lab 0 if you have forgotten how to do this). The input from the potentiometer should be displayed on the LEDs of the stack, the servo should move from one end of its range to the other at a minimum to maximum value on the potentiometer. The servo should move +/- 40-60 degrees, do **NOT** alter the MINPULSE or MAXPULSE values in the header file.

Explore how to control the Servo and what you can do with it. What is the minimum change of the pulse high time that results in motion of the servo head? (The module allows you to control the pulse width to 1uS resolution). In degrees, what is the range of the servo? What is its maximum angular velocity? What is its minimum angular velocity for smooth motion? (You may not want to have the servo under potentiometer control while performing these tests).

Demonstrate to the TA/tutors your control over the servo, the LEDs tracking the potentiometer, and your code for doing this for a checkoff. In the final report include annotated scope traces of the output of the servo control pin, interesting snippets of your code, block diagrams, and a writeup of what you observed.

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:

- CMPE118_Uno32_IO_board documentation

- CMPE118_DS3658 documentation
- CKO Ch. 8
- AD.h from the C:\CMPE118 include directory
- LED.h from the C:\CMPE118 include directory
- PWM.h from the C:\CMPE118 include directory
- Appendix A on LDOs from Lab 2

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 vary with your pot. 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.

OUTLINE:

You are going to control a DC motor using a DS3658 High Current Peripheral Driver that can sink up to 600mA per output. You will be controlling the motor speed (unidirectional control) by reading the input from a potentiometer directly into the Uno32, and then driving a PWM signal from the Uno32 to drive the DS3658.

You will output the scaled value of the potentiometer on the LEDs of the Uno32 stack, and drive the motor accordingly (0V -> 0 LEDs -> 0% duty cycle), (3.3V -> 12 LEDs -> 100% duty cycle). You will want to use a ribbon connector or wires soldered to your male-male pins to connect the IO ports on the Uno32 to the DS3658 inputs.

The DC3658 only sinks current, so it needs to be connected to the low voltage side of the DC motor. 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 for this purpose.

You will also 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.

Check your potentiometer output with a multimeter to ensure it never exceeds 3.3v BEFORE connecting it to the UNO IO board. Always check the voltage range on a wire before you connect it to the UNO IO board! Students have destroyed many input pins in this class.

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.

Run the motor at 20% and 80% duty cycles and take o-scope traces of the motor voltage. Identify the different areas of the drive, inductive kickback, snubbing, and other features of interest on the trace (you may want to include the PWM as another signal on the trace as well).

Demonstrate to the TA/tutors your control over the DC Motor, the LEDs tracking the potentiometer, and your code for doing this for a checkoff. In the final report include annotated scope traces of the output of the PWM control pin, traces of your 20%/80% tests, interesting snippets of your code, block diagrams, and a writeup of what you observed.

PART 3 – SNUBBING THE INDUCTIVE KICKBACK

OVERVIEW:

In this assignment, you will explore the use of diodes and Zener diodes in different configuration 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:

- CMPE118_Uno32_IO_board documentation
- CMPE118_DS3658 documentation
- CKO Ch. 22, 23, 24, 26, and 27
- AD.h from the C:\CMPE118 include directory
- LED.h from the C:\CMPE118 include directory
- PWM.h from the C:\CMPE118 include 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 vary with your pot. 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.

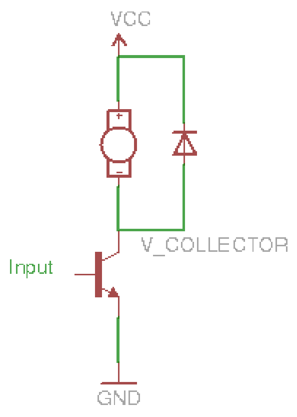
Your Uno32 stack and the DS3658 need to have a common ground connection back to the power supply. This can be directly to the ATX, or through the power distribution board.

OUTLINE:

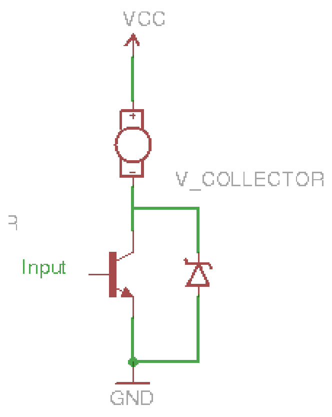
You are going to explore inductive kickback in a motor and how to snub it using various diode placements in the drive path of the motor. The hardware and software setup should be identical to the previous part, when you controlled the DC motor in a unidirectional manner.

You will also explore the limits of the PWM drive to the DC motor, and how the snubbing arrangement will affect those limits. Details of the inherent limitations (both maximum frequency and limits of resolution of the duty cycle) of the PWM module can be found in the header file PWM.h in the C:\CMPE118\Include directory. You will be using a Zener diode in schematic C, and a normal diode in schematic A. Make sure you can identify the two different diodes and use the right one in the right place. Verify this before applying power.

Study the waveforms that are generated when you hook the DC motor with the following snubbing arrangements. You will need to make an O-scope trace of the motor's waveform at an intermediate duty cycle, a very high duty cycle, and a very low duty cycle. What differences do you observe between these circuits? Explain these differences. For each circuit, identify and quantify the kickback peak and the decay time.



A



C

The transistor in the schematics are the DS3658. 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. Instead, connect the o-scope's ground lead to ground, and the high lead to $V_{collector}$. This is an indirect measurement of the motor's waveform—you will have to do a little math to recover the actual waveform.

When comparing the different diode circuits, pay close attention to the decay time of each circuit. You may need to adjust your timescale to get a good measurement.

Find limits on the frequency and duty cycle for each circuit. Under what conditions is the motor's speed a linear function of duty cycle?

Add resistive torque to the motor (grip the shaft with your fingers, then with a pair of pliers). What is the effect on the waveform? Why?

Demonstrate the various waveforms (live) on the o-scope to the TA/tutors for a checkoff. They will ask you and your labmate questions to determine if you can explain what you are seeing. In your lab report, include pictures of your waveforms, very well labelled, and a detailed discussion of what you encountered. Include insights what is going on.

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. You will use the CMPE118 DRV8814 Dual H-Bridge Driver Module board. 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:

- CMPE118_Uno32_IO_board documentation
- DRV8814 Dual H-Bridge documentation
- CKO Ch. 22, 23, 24, 26, and 27
- AD.h from the C:\CMPE118 include directory
- LED.h from the C:\CMPE118 include directory
- PWM.h from the C:\CMPE118 include directory
- Appendix A: 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 DRV8814. 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 DRV8814. You should see a PWM signal, with the duty cycle vary with your pot and switch. 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 DRV8814 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.

OUTLINE:

You are going to 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. You will use the CMPE118 DRV8814 Dual H-Bridge Driver Module board. 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. Appendix A details how to hook up a switch to the Uno32.

Requirements:

- 0V corresponds to 0% duty cycle PWM
- 3.3V corresponds to 100% duty cycle PWM
- Switch determines direction
- LEDs indicate speed and direction

Note that for the LEDs, you should be doing this with bit shifting and masking. If you have a very convoluted switch statement or lots of “ifs” you are doing it wrong.

Determine the frequency limits of operation for your setup. If they differ from those found in parts 1 or 2, explain why. Remember to add a resistive load to your motor and see what happens.

Demonstrate your motor drive software/hardware to TA/tutors for checkoff. It should meet all the requirements. In the final lab report, include a picture of the setup, any snippets of interesting code, and a detailed explanation of what you did and discovered.

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:

- CMPE118_Uno32_IO_board documentation
- DRV8814 Dual H-Bridge documentation
- CKO Ch. 26

- Stepper.h from the in the C:/CMPE118 include 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 DRV8814. 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 DRV8814. 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 DRV8814 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.

OUTLINE:

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 DRV8814 H-bridge. That is, you are going to manually control the coils individually in order to switch the stepper coils to make to go forwards and backwards in Full Step, Wave, and Half-Step drives.

You will 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 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).

Are these rates changed by loading the stepper (fingers on the shaft)? How about by 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 (read the CKO chapter on steppers if you don’t know what that means).

Demonstrate your software and stepper drive to the TA/tutors for checkoff; that is, change your #define and recompile and show the stepper working on all modes of drive. Note that you can 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. In the report, include the interesting snippets of code, and 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:

- CMPE118_Uno32_IO_board documentation
- DRV8811 Stepper Board documentation
- CKO Ch. 26
- Stepper.h from the Lab 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 DRV8814. 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.

OUTLINE:

You are going to control a stepper motor using software and the DRV8811 Stepper Driver Module. That is, you are going to use a dedicated board that will sequence the coils for you, so you need to send only steps and direction (and enable).

You will begin with the Stepper.c/h module, and #define DRV8811_DRIVE. 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, and hook it up to the DRV8811.

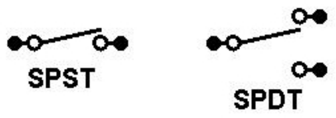
Determine the maximum step rate you can start, and the maximum rate at which you can step without losing any steps (try marking the shaft and rotating forwards and backwards 360 degrees to see if the marker creeps).

Are these rates changed by loading the stepper (fingers on the shaft)? How about by the drive method (See the jumpers on the DRV8811 for changes in both current limits and step modes)?

Demonstrate your software and stepper driver working to the TA/tutors for checkoff. Replicate how you determined the maximum step rate for them. Again, here you can hard code the steps you take in your software. In the report, include the interesting snippets of code, and a description of what you discovered about your stepper limits when driving the stepper with a dedicated board.

APPENDIX A: HOOKING UP A SWITCH

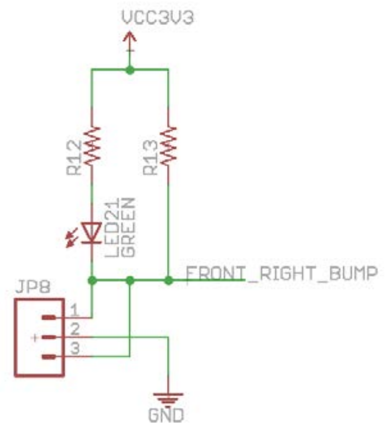
The basic switch is used almost everywhere (bumpers, etc.) to indicate to the software that it has been pushed. When determining if the switch has been pushed, a clean signal needs to be sent to the pin on the microcontroller. For a single switch, we will simplify it to a single pole, single throw (SPST)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

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.

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