MICROCONTROLLER LAB

1. INTRODUCING *ARDUINO*

Arduino is great. It's cheap, easy to use, and comes pre-assembled with a lot of useful tools for data acquisition and control. Let's get started…

TASK 1.1: INSTALL, CONNECT, AND COMMUNICATE WITH YOUR ARDUINO

You will first need to install Arduino development environment (i.e. software used to write Arduino programs and upload them to the device). This software will come with a "driver" that lets your computer know how to communicate with the Arduino.

Note: Different machines (PC, MAC, UNIX) require different drivers, please follow the appropriate installation instructions.

- Download and install the Arduino software/drivers: http://arduino.cc/en/Guide/HomePage
- Attach the Arduino Uno to your computer via a USB cable. The **green** power LED should go on.
- Run the Arduino software (Aruduino.exe), and set the following options:
  - Tools->Board: Arduino Uno (**Note**: if you have another Arduino Board, select appropriately)
  - Tools->Serial Port: (The COM port on which your Serial-USB driver was installed…try a few.)

Now you are ready to write, compile, and upload your first program!

```
// Lines preceded by "//" are ignored – use them for comments!

// the setup function runs once when you press reset or power up the board
void setup() {
        command1;
        command2;
        // Note: every command line must always end with a ";"
}

// the loop function runs over and over again, forever
void loop() {
        command3;
}
```

Every Arduino program has the structure shown above. A *setup* function for initial tasks and a *loop* function that runs continuously. There many different "commands/functions/tasks" that you can place between the { } of each of these functions to create the desired program. The easiest way, by far, to learn what these commands are *and* the syntax required to assemble them (i.e. to learn how to program your Arduino), is to start from the wonderful library of "Examples" included in the Arduino software.

EXERCISE 1.1.1: RUN YOUR FIRST PROGRAM ("BLINK") ON THE ARDUINO

- Go to: File->Examples->Basics->Blink
  - First "*verify*" (to check that the code is correct). Then "*upload*" the example.
  - Is the on-board LED in position 13 blinking? *If the LED is not blinking…find an instructor.*

Advanced Exercise 1.1.1: Can you make the LED blink at 5Hz?

Now let's try something a bit more challenging as a means of nurturing your fledgling Arduino skills…

EXERCISE 1.1.2: CHANGE THE *APPARENT* BRIGHTNESS OF THE LED BY CHANGING DUTY CYCLE

- Duty cycle is the ratio of (Time On)/(Total Time) expressed as a percentage.
  - For example, if the LED is on for 50 ms out every 200 ms, then the duty cycle is 25%.
  - *Write a program that sets the duty cycle to an arbitrary value between 0 and 100%.*

Note: If you don't want to notice the LED flickering On and Off, then it must flicker faster than your psychophysical "fusion frequency", the frequency beyond which you can no longer detect the blinking.

Advanced Exercise 1.1.2: what is your fusion frequency?

(Congratulations, you just did your first neuroscience experiment with an Arduino!)

2. DIGITAL INPUT

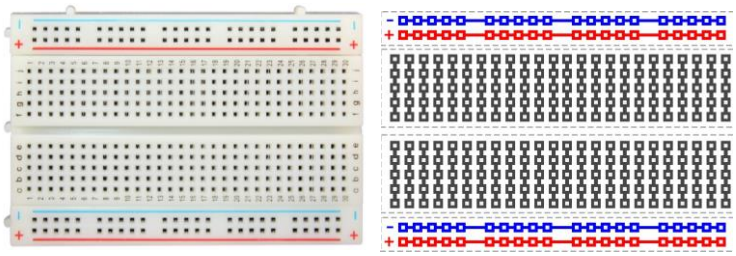
The most basic form of input to a computer is a binary value (0, 1) represented by a specific voltage levels (0 and 5 volts for Arduino Uno). Buttons, levers, beam breaks, etc. are all well suited for digital input. Let's monitor the state of a "push button" with your Arduino.

TASK 2.1: ASSEMBLE THE PUSH BUTTON CIRCUIT

You will have one or more of the following buttons/levers/switches. Get one.



To make your life easier during this and subsequent exercises, we will use a prototyping board (a.k.a. solderless breadboard) to connect our external circuit components. Get one of these as well.



Solderless Breadboard Pad Diagram: *Note how the individual pin groups are connected to each other!*

The first, and vitally important, task is to determine how your particular switch/button/lever works. There will be a number of different wires/contacts coming out of the device. Pairs of these contacts are connected to each other: all of the time, only when the device is pressed, only when it is not pressed, or never.

EXERCISE 2.1.1: WHAT IS THE BEJAVIOUR OF YOUR BINARY INPUT DEVICE?

- Use a voltmeter, set to the resistance/connectivity setting, to determine which contacts are altered (connected or disconnected) when the device is activated (pressed, pushed, or toggled).

Now that you know how your switch behaves, we next need to use this behaviour to change a voltage level between 0 and 5 volts and then send this changing voltage into the Arduino.

Note: 0 and 5 volts is the convention most Arduinos uses, called TTL, it is a common, but by no means the only, standard. For example, the Arduino Due, a more sophisticated device, uses a lower voltage standard (+3.3 V) and 5V can damage that board.

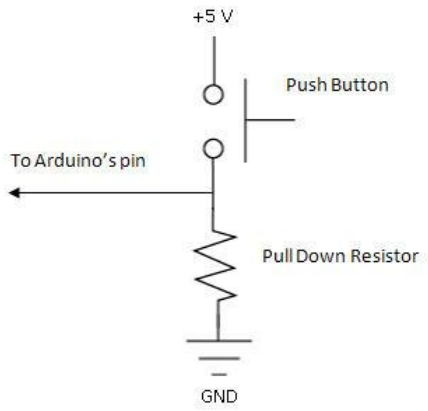
The seemingly obvious way to do this would be to connect +5V from the Arduino board, across the open switch contacts, and then send the other end to a digital input pin on the same Arduino. Thus, when the switch is pressed, the input pin is directly connected to the +5V. Great, but what happens when the switch is not pressed? Well…let's experiment…

EXERCISE 2.1.2: WHAT DOES A DIGITAL INPUT PIN READ WHEN NOT CONNECTED TO ANYTHING?

- Assemble the circuit described above.
- Go to: File->Examples->Digital->Button
  - This program will turn on the onboard LED (pin 13) when the input to pin 2 is high (+5V).

You might notice something strange. When you let go of the button, the LED does not turn off immediately…or maybe it does…sometimes, but not others. WTF?

When the button is *not* pressed, the digital input pin is connected to *nothing*…which is not the same as 0 volts! An input pin (digital or analog) that is not connected to anything is called "floating". It will often report random values, which may even appear to depend on noise in the local environment. If we want our button to behave, then we need to connect to 0 volts when *not* connected to 5 volts. Take a look at the following circuit:



Using a pull-down resistor to connect the Arduino's input pin to 0 volts when the button is released.

EXERCISE 2.1.3: PUSH BUTTON SWITCH WITH A PULL-DOWN RESISTOR

- Assemble the circuit shown above. What should the value of the pull-down resistor be?

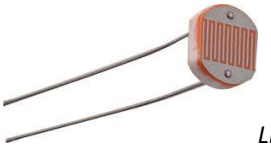
- **Note:** You want less than 1 mA of current flowing when the switch is pressed.
  - *(Hint: use Ohm's Law)*

3. ANALOG INPUT

Any device that converts a real world quantity into a voltage can be used as a sensor (transducer). In the following example we will be using light-dependent resistors (LDRs), but any such sensor will suffice.

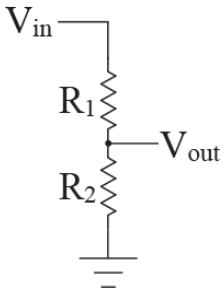
TASK 3.1: MEASURE LIGHT (OR ANYTHING) WITH A COMPUTER

A simple light sensor can be made from any material that varies its internal resistance based on the intensity of light absorbed, e.g. an LDR.



LDR

However, converting this change in resistance into a change in voltage will require a simple measurement circuit…the famous "voltage divider".



Voltage Divider

$$Vout = Vin * \frac{R2}{(R1+R2)}$$

The voltage appearing at V_{out} depends on the ratio of R1 and R2. Therefore, if either of these resistors is replaced by an LDR, then a light-dependent change in resistance will result in a change in the voltage measured at V_{out}, i.e. you will now have a voltage signal that depends on light intensity.

EXERCISE 3.1.1: BUILD A LIGHT SENSING CIRCUIT

- Assemble the circuit shown above. Use the +5V coming from Arduino and measure the V_{out} with a voltmeter. Is the circuit light sensitive?

Advanced Exercise 3.1.1: What is the "optimal" value for the other, fixed resistor?

Ultimately, we will want to read this varying voltage level into a computer (and then do all sorts of cool things with that data). For this, we will use the analog-to-digital converter (ADC) of the Arduino. There are four pins available for analog input, and when the function *analogRead(*pin#*)* is called, it will represent ("digitize") the voltage appearing at the specified pin as a 10-bit binary number.

EXERCISE 3.1.2: READ ANALOG VALUES WITH ARDUINO

- Go to: File->Examples->Analog->AnalogInput
  - This program will read the analog voltage at the indicated pin# and use this value to alter the flickering rate of the onboard LED.

Advanced Exercise 3.1.2: Change the program to turn off the LED when the light sensor exceeds a certain value (i.e. design a "smart home" lighting controller)

This is already pretty great, but it would be even better to know exactly the values measured by the ADC. Indeed, the Arduino does know these values, but it doesn't have a "display" with which it can tell you what they are. (**Note**: one can buy an extension board for Arduino (a "shield") that has a simple text display, these can be very useful for certain projects). Therefore, we would now like Arduino to communicate the values it digitizes back to our PC, which does have a display, for visualization (or for saving the data to a file). In order to send data between the PC and Arduino, a connection must be established using a standard "protocol" which both devices understand. We will use the very common "Serial Communication Protocol" (or UART/USART).

Note: this is the same connection you made/used to originally program the Arduino.

In order to establish a connection to dynamically send (or receive) data with your Arduino, then you must do the following:

- Open a connection: specifying the speed (baud rate) in Bits/sec – "Serial.begin(baud rate);"
- Write to that open "port": "Serial.println(*whatYouWantToPrint*);"
- Wait a bit: (if you send data too quickly, the serial port can get "clogged")

The Arduino code that does this, and only this, is as follows:

```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
  int value = 42;
  Serial.println(value);
  delay(1);         // delay (1 ms) in between writes for stability
}
```

EXERCISE 3.1.3: SEND MEASUREMENTS FROM ARDUINO TO YOUR PC VIA THE SERIAL CONNECTION

- Go to: File->Examples->Basic->AnalogReadSerial
  - This program will read the analog voltage at the indicated pin# and then "print" it as a stream of text characters (ASCII-bytes) that are sent back to the PC via the serial port connection.
  - In order to visualize these incoming values, use the "Serial Monitor" found as a tool (the small magnifying glass) in the upper right corner of the Arduino software.

Note: The serial monitor just receives all incoming Bytes that arrive on the same COM port as the connected Arduino and prints them directly into a text window….nothing fancy. (Actually, if you consider how a modern video display works…it's crazy fancy.)

Advanced Exercise 3.1.3: How quickly can you take a measurement with Arduino? Can you measure this time lag and report it via the Serial Monitor?

EXERCISE 3.1.4: SAVE SOME COOL DATA

- Acquire some interesting data from your sensor and then copy it to a text file from the Serial monitor window (i.e. cut and paste).
- You will need this data in a later "Data Analysis" section.
- ☺ Enjoy ☺

4. DIGITAL OUTPUT

It is clearly useful for computers to not only *measure* digital voltage levels (0 to 5 V), but to also *generate* these same distinct voltages. Actually, you have already done this with the "Blink" example. Pin 13 of the Arduino Uno is directly soldered to an onboard LED that, conveniently, reports the binary state of that pin. You likely noticed that the function "digitalWrite(pin#, state)" was used to set the specified pin to the specified state. Let's use this "digital output" functionality to control a few different external devices.

TASK 4.1: CONTORL AN EXTERNAL LED

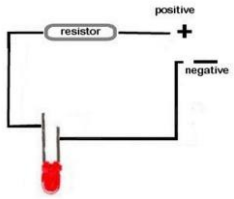
This may, at first, seem rather straightforward. Grab an LED, connect to a digital output pin and a ground pin of the Arduino and "blink away". However, there a few very important things to consider.

LED
(**Note**: one leg is longer than the other)

LEDs are diodes, meaning current will only flow in one direction. If you manage to get current to flow the other way, then you have damaged your LED (most of the time).

- **Important:** Always check the correct polarity of the LED: The *long leg* connects the positive side of the circuit

Furthermore, LEDs are somewhat sensitive. If too much current flows, even in the correct direction, then it will be damaged. The current flowing through an LED is not limited along across the Diode, so we *must* include some device for limiting the current to an appropriate level (which will be different for different LEDs). The simplest such device is a "current limiting resistor".



EXERCISE 4.1.1: BUILD AN LED CONTROL CIRCUIT

- Build a circuit, using the solderless breadboard, such that the current flowing though the LED is roughly limited to 10 milliAmps.
- "Blink" your external LED from pin 12 of the Arduino.

There are many, many devices that can be controlled (switched ON and OFF) in this manner, but not all. Some external devices simply require too much "power" (Power = Voltage * Current, P = VI). In other words, these devices draw too much current, or require too high voltage, to be *driven* by the Arduino's (i.e. USB's) tiny power source. Another terminology used for describing how much power a device requires is its *load* on the driver circuit.

Note: Direct digital output via the Arduino is normally limited to device require less than <40 mAmps.

TASK 4.2: CONTORL A PIEZO BUZZER

You can also use a sequence of digital outputs to generate (ugly) sounds. A pizeo is a crystalline material that changes its shape, slightly, when exposed to an electric field. If exposed to rapidly varying electric field, then this will cause rapid changes in shape, and if this happens inside of a small plastic case with a tiny hole, then rapid changes in air pressure are produced, i.e. sound.

Piezobuzzer

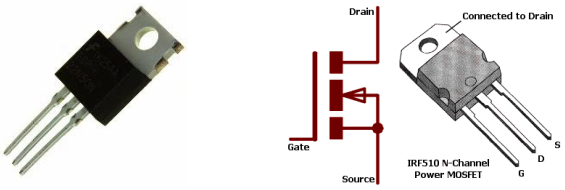
EXERCISE 4.2.1: CONTROL BUZZER FREQUENCY

- Most piezo buzzers can be directly driven by the digital output of the Arduino. Therefore, your task is to simply generate changes in the High-Low values, a square wave, at the output pin connected to your buzzer.
    - Modify the "Blink" program to run on pin 12 and to "flicker at 500Hz".
    - Connect your buzzer between pin 12 and GND. Lovely, eh?
    - Investigate the "tone(pin, frequency)" function.

Advanced Exercise 4.2.1: Use an analog measurement from your LDR circuit to change the frequency of "tone" (frequency of square wave) digital output. That is measure the voltage level, which should be light dependent, using analogRead() and then use this number to control the output frequency. This is sometimes called a "Theremin", it is other times called "Annoying."

TASK 4.3: CONTORL A "HEAVY LOAD"

Let's say you wanted turn on the room lights with your Arduino, or start your car. These devices require a significant amount of power to run, which cannot be provided directly by the Arduino. However, you can certainly use the Arduino to "gate" the flow of power to these heavy load devices. There are many ways to build such computer-controlled gates (e.g. relays), but out favorite is the absurdly wonderful *transistor*. Specifically, the high power MOSEFT (Metal-Oxide-Semiconductor-Field-Effect-Transistor).

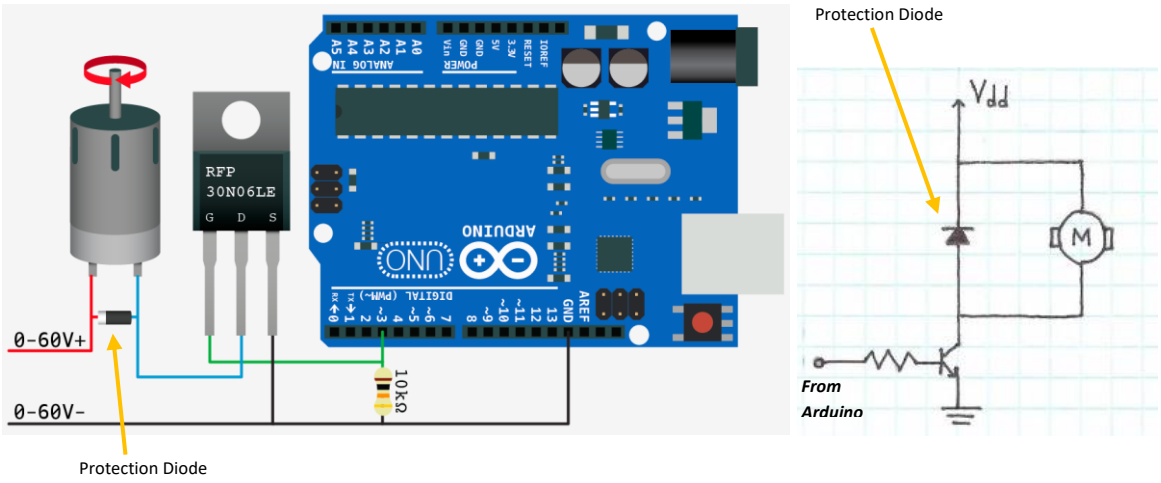


MOSEFT: Photo and Schematic of IRF510

Transistors are beautiful devices and you would do well to avoid having me explain the gory details of their operation (I will get excited and waste all of the time allotted for this worksheet. However, do ask me later over beer!). In short, the flow of current between the *Drain* and *Source* (for an N-Channel MOSFET, current flows D→S, it is the opposite for a P-Channel device) is controlled by the voltage present at the *Gate*. Low Gate voltage, no current flow, high Gate voltage current flow. The loveliness is that the current required to open and close the Gate can be very small, while the current flowing between D and S can be, potentially, huge…over 5 Amps for the IRF510 shown above. Therefore, MOSFETs allow a tiny Arduino digital output to turn on a big electric motor. This is very useful….let's try.

EXERCISE 4.3.1: TURN ON A DC MOTOR WITH YOUR ARDUINO

- Find a motor (a small one). Try driving it directly with the "Digital Output" pins. It might move a bit, but it might also draw so much power that your Arduino resets!
- Assemble the circuit shown below.



Arduino Motor Control via MOSFET: *Cartoon and Schematic*

There is ***A LOT*** going on here! Let's unpack.

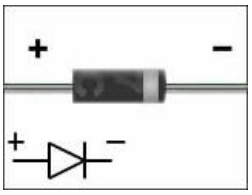
1. The DC Motor requires a separate power source (0-60V+ in the picture, V_{dd} in the scheme). This will supply the extra power needed to actually drive the motor. You can use some batteries, a DC transformer, or even your benchtop power supply. Start with only a few (+3 V) volts.

External Power Supplies

2. The "protection diode" is vital. Why?
   a. DC motors are made from coils of wire, that when current flows them, create the magnetic field this is used to "move" the permanent magnet attached to the motor's central shaft. When current ceases (i.e. the transistor switch is closed), the magnetic field disappears and the motor no longer has power. However, the motor still has momentum and will keep spinning for a little while after the power (drive current) is removed. During this period, a permanent magnet (attached to the central axis) will be moving past coils of wire. When a magnetic field changes across a conductor, it *induces* a current to flow in that conductor. (This is how an electric generator works!) However, this *induced* current is flowing in

the opposite direction as the current used to drive the motor…in fact, this *induction* is the main reason the motor slows down!

b. Now, recalling the previous discussion of LEDs, like diodes transistors are also polarized! The MOSEFT is happy to gate current flow one direction (from D to S for an N-channel device), but very unhappy when it encounters current going the *other way.* Therefore, we must protect the transistor from these induced *negative currents*. Hence the protection diode. This diode is also polarized (the silver tip is on the side of the diode to should point in the direction of current flow: + to – by convention) and in the orientation indicated, will not pass any positive current (the current used to drive the motor) under normal operation. However, when the motor power is stopped, i.e. the transistor closes, and the motor induces a negative current. This induced current can flow back through the diode to the +V power supply and not through the transistor. Yeah!



Diode: Picture and Schematic (with polarity)

c. The protection diode is something people often leave out, because it "doesn't seem necessary". In these such labs, we *always* break some transistors because someone forgets the protection diode. It is necessary.

3. Make sure the G, D, and S pins are identified and connected correctly.
4. Have fun!

EXERCISE 4.3.2: WRITE A PROGRAM TO TURN YOUR MOTOR ON WHEN THE LIGHT CHANGES

- Use the AnalogInput circuit with your LDR and the digitalOutput + MOSFET circuit from the previous exercise.

Advanced Exercise 4.2.2: Build a speed controller: vary the Duty Cycle of a flickering digital output to motor switch MOSFET in a light dependent manner.

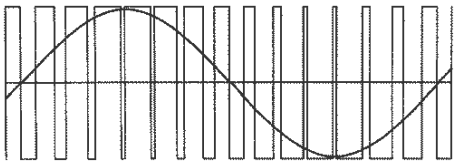
Super Advanced Exercise 4.2.2: Build a speed servo controller. (Hint: you'll need to detect each full/partial rotation)

5. (PESUDO) ANALOG OUTPUT

To come full circle. We will also briefly discuss generating "continuous", analog voltages. This is very much the inverse of measuring analog voltages with an ADC, yet, unfortunately, the Arduino Uno does not have a Digital-to-Analog Converter (DAC).

Note: The Arduino Due does, and it is great fun generating high precision sound waveforms that can vary from 1 microsecond to the next...I encourage you to play around!

Given that many devices require more than 1-bit of control resolution, yet computers (and the Arduino) are best at generating only two voltage levels (0 and 5V). Many external devices will make use of a pseudo-analog control signal, such as Pulse-Width-Modulation (PWM). You have already used PWM to vary the "apparent" intensity of an LED, but you can also do the same with many devices.



PWM: Digital approximation of a sine wave

TASK 5.1: CONTORL A "HOBBY" SERVO

Realizing that many simple digital computers would prefer to generate a digital signal, devices requiring analog control were built that automatically interpret a PWM input as an analog value (i.e. they have an internal circuit that measures Duty Cycle and uses that as the continuous control signal).

EXERCISE 5.1.1: CONTROL A SERVO POSITION USING PWM

- Arduino has special digital output pins that can be used for high precision PWM. These pins are marked with a (~) symbol. Use one such pin to control the position of your hobby servo.
- The example "File->Examples->Serv0->Sweep" is an excellent place to start.

Advanced Exercise 5.1.1: Attach your LDR to the servo. Write a program to point in the brightest direction.

Do anything.

CORE CONCEPTS

TTL – Transistor-to-Transistor Logic

Pull-down Resistor

LDR

Duty Cycle

UART/USART

COM Port

Voltage Divider

Baud Rate

ASCII-Code

ADC

Debounce