Physics STEM Lab: Appendix

Basics of Signals and Wires:

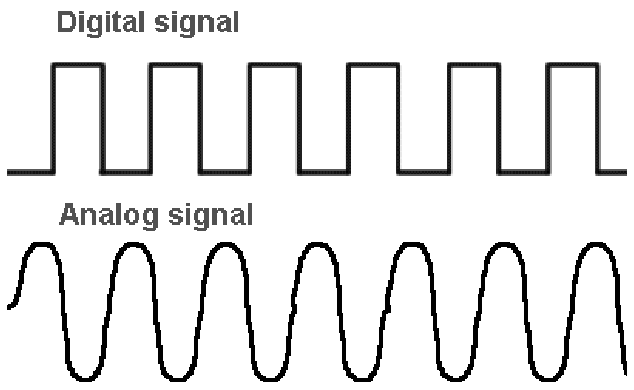
All communication in electrical engineering is handled through signals. It's how boards, devices, and any other related components talk and act upon each other. A signal at its most basic level is short bursts of voltage that can be read and interpreted by electronic devices. Wires and connectors are most commonly used for transmitting these signals. Connectors are commonly used to join two components together. A connector is either a metal protrusion (male pin), or a hole with contacts that fit a particular protrusion (female receiver). Only male and female connectors can join to form a proper connection. Wires are used to carry electricity and signals across a distance. Most wires are made out of bundled copper strands, but sometimes other metals are used, and some wires use solid metal instead of standard metal as well. As long as the two ends of the metal touch both of the connectors, the electricity is conducted throughout the wire, and the signal is carried between the two devices.

A jumper, shown above, is some wire with two connectors on either end. The jumper carries wire between two electronic components, typically from a board to a device the board uses. Jumpers come three type, depending on which 'gender' the connectors on the end are: Male-Male, Male-Female, and Female-Female. This jumper, for example, would be Male-Female, because there is a Male connector on one end, and a Female connector on the other end.

Input vs. Output:

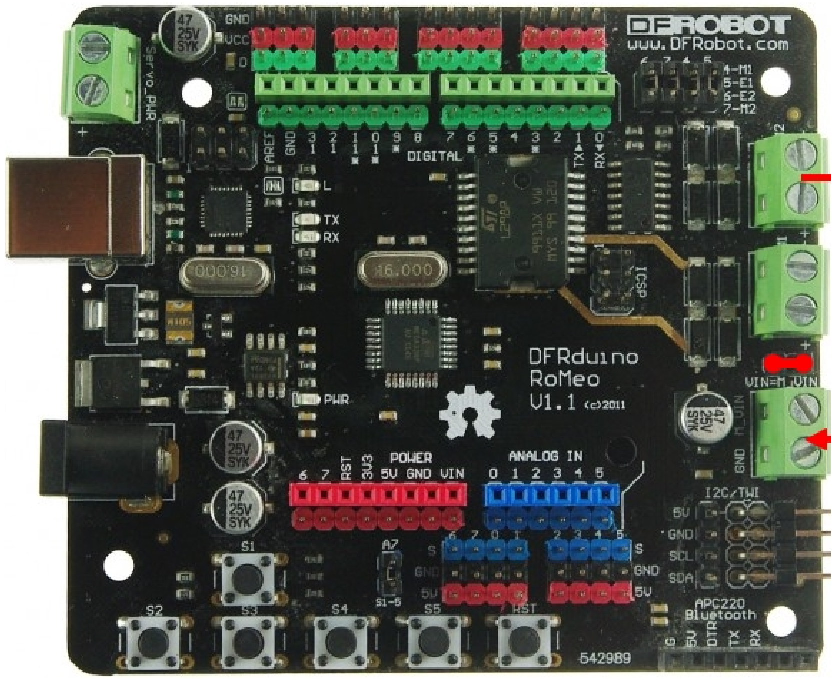
Electrical input and output (I / O) refer to the transfer of signals between the board and the devices connected to it. Input specifies the movement of a signal from a device into the board. The board is *receiving* information. Output specifies the movement of a signal from the board to a device. Thus, the board is *sending* information. Your input devices, the devices you will be reading data from, will primarily be the sensors you have plugged in. Your output devices, the devices you will be sending instructions to, will typically include LEDS, motors, or any other actuators you have attached to the board. You control all output signals that are sent, however, input signals will come into the board continuously, and you will need to actively read them in order to be able to use them.

Digital vs. Analog:



Digital and Analog signals are used to send information between electronic devices. The primary difference is digital signals represent a binary switch - it can be either on or off - whereas analog signals can be a range of values.This can be seen by observing their signal waves. Digital signals generate rectangular, on-off waves, while analog signals generate smooth, continuous sine waves.Mathematically, digital outputs can only be 255 (the max value), or 0 (the minimum, or off value). An analog output can be anywhere between 0 to 255. Ultimately, this means we have more control over analog signals, as we can adjust how much 'power' or voltage to send to the device we are sending it to. We could set an LED to only half brightness, or have a motor run at 30% of its maximum output for example. With digital signals, we lose that control. The brake lights on a car for example, may only turn on or off: there is no need to have it run at only a fraction of its brightness. Digital signals do have their advantages, however. They require less power and are more reliable than analog signals, and simply make more sense for switches and other similar values.

Diagram of an Arduino Board:



- From the top of the board, going clockwise:
- The green pins with the numbers under them, both male and female, are digital pins. They are typically the most used pins on any given Arduino project. The male and female ends of the pin are connected, so either can be used, depending on your preference.
    - The pins with squares underneath their numbers (3, 5, 6, 9, 10, and 11) are PWM-enabled digital pins. You can call an analogWrite on these to invoke PWM and simulate an analog signal.
    - Note: the 0 and 1 pins are used for serial communication, and thus cannot be used for this project. Also note: pins 4, 5, 6, and 7 are in use by the motors, and thus also cannot be used.
- The columns of green, red, and black pins above the digital pins are used for conveniently connecting digital I/O devices.
- The first two green protrusions (with the screwdriver heads) on the right side are the motor connectors. When the top connector is plugged in, the board uses pins 4 and 5 to control the motor, and when the bottom connector is plugged in, the board uses pins 6 and 7 to control it. For this project, 4 and 5 will control the "left" wheel, and 6 and 7 will control the "right" wheel.
- The blue pins under "Analog In" are analog input pins. These, as their name suggests, are used exclusively for *reading* analog signals. Once again, the male and female pins are connected, and may be used interchangeably.

- The columns of blue, black, and red pins under the Analog Inputs are used for conveniently connecting certain analog I/O devices. We will connect the photoresistors using these.
- The red pins under “Power” do a variety of things: the pin marked vin is used for inputting voltage into the board, The two GND pins are grounds, the 5V and 3V3 pins provide constant 5 Volts and 3.3 Volts respectively, and the RST pin may be used to “reset” the board using software.
- The switches marked S1 - S5 under the analog and power pins are buttons - they can be read and used to provide mechanical input from the user. The button marked RST (all the way to the right), will restart the software anytime it is pressed.
- Finally, the black connector on the left is used to plug in the battery, and the white connector is where the USB cable plugs in.
- The other components on the board are rarely needed, and will not be used for this project.

Physics STEM Lab: Programming Primer

A program is a set of instructions given to a machine for execution. Whereas the hardware of a machine can be referred to as its body, the software can be thought of as the 'brain' of the machine: it takes in the data received by the body and tells it how to respond. Programming is used today to operate almost anything technological you see today, from the simple google search on your computer, to the cutting edge robots waiting to be released. For robots in particular, who must engage and respond to their environment in real time, software is especially important. The robot must be programmed to read its inputs, and output appropriate responses, or it will likely get into undesirable situations, e.g. driving off a cliff. The board we're using, the Arduino, is a microcontroller designed to easily do just that; we can program our Arduino to read data from our sensors, make appropriate decisions, and write them to our output devices.

At the core of any program are three things: variables constants, and functions. Variables are values that you expect to change over the course of a program. In Arduino, when defining a value for the first time, you must declare its type, followed by its name. For instance, int sensorValue; means to create a variable of type int (a number), and name it sensorValue. I can then use sensorValue in my program. There are two things we do with variables, for the most part: assign them, or read their value. We assign variables a value using an equal sign. For example sensorValue = 0; stores a zero value in sensorValue. I can then read the value of sensorValue, by simply using its name in the program. print(sensorValue) will print a value of 0 to the monitor (assuming that was the last value that was stored).

Constants are values that you don't expect to change. For example, we can define pi as a constant, since that will never change no matter what we do. In Arduino, we do this by writing #define, followed by the constant's name (in capital letters) then its value. Such as #define PI 3.14. Note: we do not need to declare its type. We can then read the constant, the same we do a variable, by simply using its name. Unlike a variable, you cannot change a constant's value once it has already been declared

Functions are a set of instructions for the program to carry out. You will perform operations on your variables and constants in these to make things happen. An example function may be:

```
int sum(int numOne, int numTwo) {
   return numOne + numTwo;
}
```

which takes two numbers and returns their sum for use. The returned value may be stored in a variable or used in some other operation in another function.

You can also perform logic statements inside a function. if and else statements perform an instruction only if a condition is true. For example, the function:

```
int biggerNumber(int numOne, int numTwo) {
   if (numOne > numTwo) {
      return numOne;
   } else {
      return numTwo;
   }
}
```

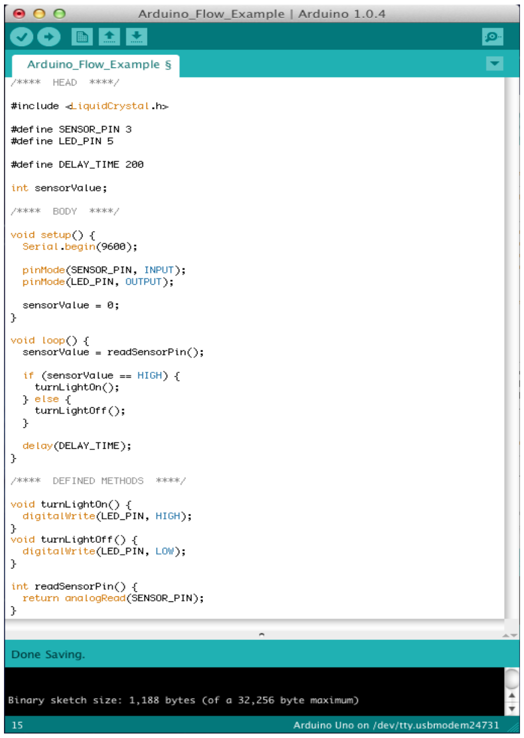
returns the bigger of the two numbers inputted.

Finally, you can also call other functions inside of functions. For example, the function:

```
int sumThreeNumbers(int numOne, int numTwo, int numThree) {
   int sumOfOneAndTwo = sum(numOne, numTwo);
   int total = sum(sumOfOneAndTwo, numThree);
   return total;
}
```

returns the sum of all three numbers, using the sum function above.

The basic flow of an Arduino program looks something like this:



```
/****  HEAD  ****/

#include <LiquidCrystal.h>

#define SENSOR_PIN 3
#define LED_PIN 5

#define DELAY_TIME 200

int sensorValue;

/****  BODY  ****/

void setup() {
  Serial.begin(9600);

  pinMode(SENSOR_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);

  sensorValue = 0;
}

void loop() {
  sensorValue = readSensorPin();

  if (sensorValue == HIGH) {
    turnLightOn();
  } else {
    turnLightOff();
  }

  delay(DELAY_TIME);
}

/****  DEFINED METHODS  ****/

void turnLightOn() {
  digitalWrite(LED_PIN, HIGH);
}
void turnLightOff() {
  digitalWrite(LED_PIN, LOW);
}

int readSensorPin() {
  return analogRead(SENSOR_PIN);
}
```

```
Done Saving.

Binary sketch size: 1,188 bytes (of a 32,256 byte maximum)
15                                  Arduino Uno on /dev/tty.usbmodem24731
```

In the head of the program, we have our include statements, define statements, and variable declarations. The include statements say we want to use data from another file, and tells the board to import them into our program. In this example, we are importing a file called "LiquidCrystal.h". The define statements essentially allow a constant value to be replaced by the text preceding it. For instance, instead of using "3" each time we refer to the pin the sensor is plugged into, we can define SENSOR_PIN to equal 3 and use that instead, for clarity. It is common practice to define where each of your input / output devices are plugged into in this manner, as well as any constants that you don't expect to change, such as the delay time between each loop.

You also declare the variables you will be using in the program's "head". In this case we update the sensor value each time through loop. In order to use this variable, you have to formally declare its type (int - which stands for integer), followed by its name (sensorValue). Occasionally, these variables may be defined in the body.

The body of the program contains two essential functions: setup() and loop(). Both of these functions must be defined within the program, otherwise the code will not run. The setup() function is used to prepare the board and ready the software to be run. In here, you define the mode (Input or Output) for each pin you will be using, as well as initialize any objects or variables. Anything else that makes sense to do before the robot actually begins to "work" should be done in setup also. The loop() function, as its name suggests, loops continuously, and is the main function of the program. In here, you perform all the logic and steps for the robot to run. This typically involves reading data from your sensors, using that data to perform any decisions or calculations, and finally using those decisions to produce outputs. In the example loop, we read the value of our sensor, then, depending on whether the sensor is activated (HIGH) or not (LOW), we write a value to our LED as our output. We also include a brief delay between loops, to conserve battery power.

Below the body lies the definition for any functions you created and used throughout setup() and loop(). In this case, we invented the turnLightOn(), turnLightOff(), and readSensorPin() functions for the main loop, so we have to tell the program what they do somewhere in the code.

Essential Arduino Functions:

With that, there are some essential Arduino functions and syntax that you should become familiar with.

- #include - imports data from another file so that its functions and definitions may be used in the current file.
- #define - assigns a value to written word, so that word may be used in place of the value (for clarity)
- int - stands for integer. Essentially tells the compiler that the value in question is a number.

- void - Indicates that the function in question returns nothing.
- HIGH - an integer constant that stands for 255, the on value for most digital devices
- LOW - an integer constant that stands for 0, the off value for most digital devices
- >, ==, < - Greater than, equal to, and less than. They are used to compare two numbers together. (Note: when you want to find out if two numbers are equal to each other, you use two equal signs "==". If you just use one, you are assigning something).
- setup() - A mandatory function that must be defined. It is called the moment you turn on the Arduino board, and is meant to ready the board and software for use.
- loop() - A mandatory function that must be defined. It loops continously after setup is called and should contain most of the logic needed for the robot to run.

These are some functions are predefined functions that can be used at will:

- pinMode(*pin#* , *mode*) - Readies the pin (determined by *pin#*) to be either an input pin, or an output pin (determined by *mode*)
- digitalWrite(*pin#* , *value*) - Writes a *value* (HIGH or LOW) to the specified pin (determined by *pin#*)
- digitalRead(*pin#*) - Reads the integer value (HIGH or LOW) of the specified pin (determined by *pin#*)
- analogWrite(*pin#* , *value*) - Writes a *value* (from 0 - 255) to the specified pin
- analogRead(*pin#*) - Reads the integer value (from 0 - 255) of the specified pin
- Serial.begin(*number*) - Used to ready communication between the board and the Serial Monitor (found under Tools => Serial Monitor while board is plugged in)