EE 201 Intro to Arduino

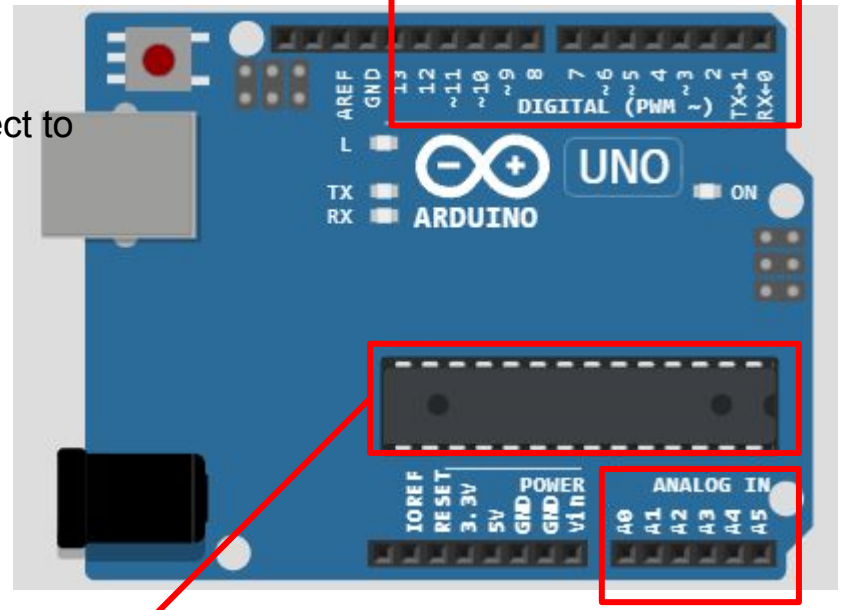
Basic Pinouts

Although there are many Arduino types, we will be referring to the most popular type, the Arduino Uno.

Uno is essentially a microcontroller with very useful pins/peripherals broken out to easily interface with it. The Arduometer uses the same microcontroller but with different pins.

Digital Pins
(can read or write 0 or 1 as 0V or 5V)

USB-B to connect to computer and provide power



ATMEGA microcontroller; the same chip that will be on our Arduometer!

Analog Input Pins
(can read 0V to 5V as 0 to 1024)

Digital/Analog Components

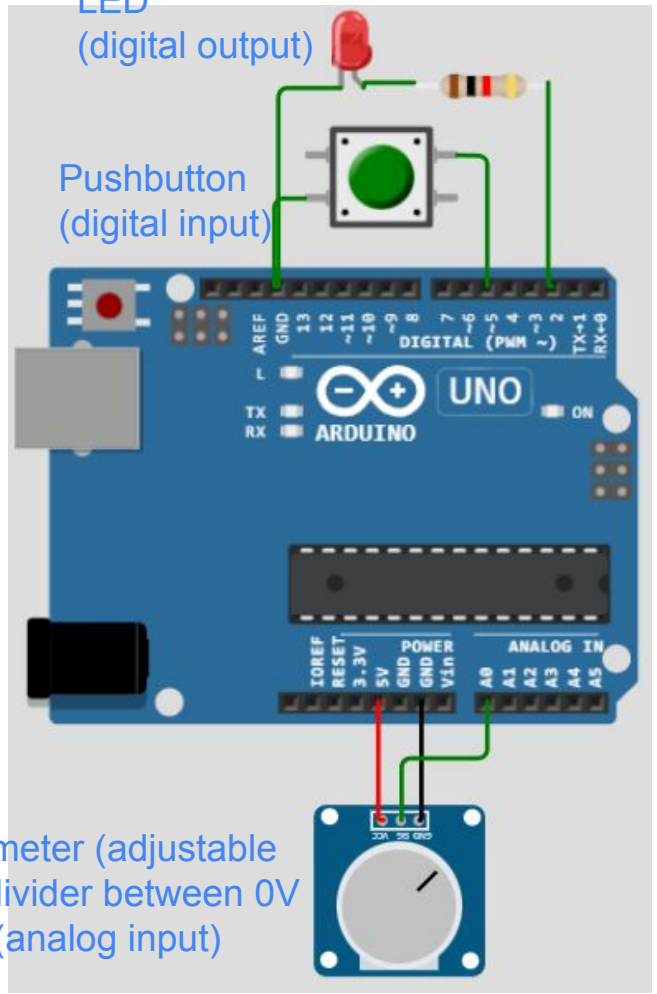
Generally, **digital** refers to binary values (0 or 1), and **analog** refers to continuously variable values (0.1, 2.33, 4.02).

Digital input & output - pins 0-13

- Can read/write 0 or 1 (0V or 5V) to these pins
- For outputs (like LEDs): connect positive side to digital pin; negative side to ground
- Max ~40 mA from digital output pins combined. This is totally fine for powering light-medium use (discrete LEDs, buzzers, etc.) but for high current applications (like LED matrices, motors), use these pins only for switching/signaling.

Analog input - pins A0-A5

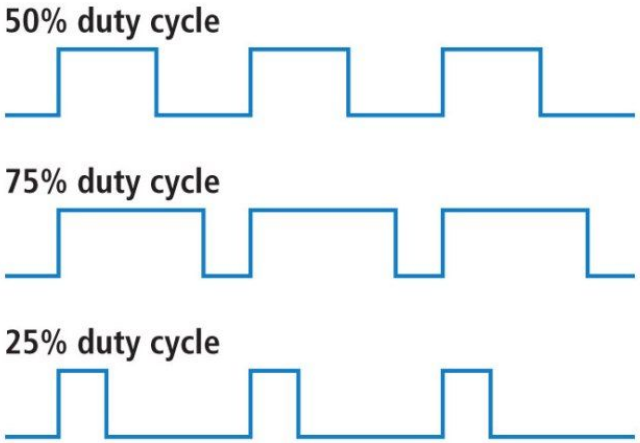
- If component requires power, wire to Power pins. Then feed the analog input signal to an analog input pin.
- Reads 0V to 5V as 0 to 1024



What about analog output?

The Arduino Uno does not natively support *true* analog output (e.g., it cannot output 1.24V, 3.55 V etc.), but it does support digital pulse-width modulation (PWM) which is a pulse of 0V-5V at an adjustable frequency on pins 3, 5, 6, 9, 10, and 11.

For many applications this is similar enough to an analog output. For example, if you wanted to dim an LED by 50%, you could drive it with a 50% PWM signal to achieve a similar end result.



Sample PWM signals

Programming Basics Part 1 - mandatory functions

Arduino requires two functions in your .ino program file:

`void setup()` is code that runs only once when you start the program.

`void loop()` is code that continuously runs after `setup()` is done. It will loop as quickly as it can but you can slow it down manually by including a `delay()` function inside.

When you power up an Arduino, it will automatically run `setup()` once and then run `loop()` continuously.

```
1   void setup() {
2     // put your setup code here, to run once:
3
4   }
5
6   void loop() {
7     // put your main code here, to run repeatedly:
8
9   }
```

Arduino API link:
https://www.arduino.cc/reference/en/

Programming Basics Part 2 - setup

`void setup()` contains code that is run first on the Arduino. The function runs only once.

Common things that go in `setup()`:

- Arduino pins require being defined as input or output before being used: `pinMode(pin, mode)`
- If you want to use prints, you should also start the Serial monitor here: `Serial.begin(baudrate)`
- If you have any constant display or output (for example an On status LED), you can set them here.

```
1  void setup() {
2    // put your setup code here, to run once:
3    pinMode(13, OUTPUT); // set up pin 13 as output
4    pinMode(6, INPUT); // set up pin 6 as input
5
6    digitalWrite(13, HIGH)); // turn on pin 13
7
8    Serial.begin(9600); // set up Serial monitor
9
10 }
```

Arduino API link:
https://www.arduino.cc/reference/en/

Programming Basics Part 3 - analog and digital interfacing

After you have defined `pinMode(pin, mode)` for all the analog/digital pins you are using, you can then read or write from them:

`digitalRead(pin)` -> returns 0 or 1

- Reads digital value of pin (returns 0 or 1 based of voltage reading of 0V or 5V)

`digitalWrite(pin, value: HIGH or LOW)`

- Writes digital value of pin (outputs 0V or 5V)

`analogRead(pin)` -> returns 0 to 1024

- Reads analog value from pin (returns 0 to 1024 scaled off of voltage reading of 0V to 5V)

`analogWrite(pin, value: 0 to 255)`

- Outputs PWM signal on pin; value given represents duty cycle (0-100% duty cycle scaled off of given value of 0 to 255)

```
1   void setup() {
2     // put your setup code here, to run once:
3     pinMode(1, OUTPUT); // set up pin 1 as (digital) output
4     pinMode(2, INPUT); // set up pin 2 as (digital) input
5     pinMode(A0, INPUT); // set up pin A0 as (analog) input
6     pinMode(3, OUTPUT); // set up pin 3 as (digital/PWM) output
7
8     Serial.begin(9600); // set up Serial monitor
9
10    digitalWrite(1, HIGH); // turn on pin 1
11    Serial.println(digitalRead(1)); // read & print pin 1
12    Serial.println(analogRead(A0)); // read & print pin A0
13    analogWrite(3, 128); // output 50% duty cycle PWM on pin 3
14  }
```

Arduino API link:
https://www.arduino.cc/reference/en/

Programming Basics Part 3 - loop

`void loop()` contains code that continuously runs on the Arduino after `void setup()` is run once.

The loop will run as fast as it can, but you can introduce a manual delay using standard delay functions:

`delay(num_milliseconds)`

`delayMicroseconds(num_microseconds)`

Causes pin 13 to turn off/on about twice per second (e.g., flashes an LED twice per second)

```
17  void loop() {
18    // put your main code here, to run repeatedly:
19    digitalWrite(13, HIGH);
20    delay(250);
21    digitalWrite(13, LOW);
22    delay(250);
23  }
```

Arduino API link:
https://www.arduino.cc/reference/en/

Programming Basics Part 4 - printing

First, set up the Serial monitor In `void setup():`

```
Serial.begin(9600);
```

- 9600 is the baud rate (serial device speed), which is fine for printing/monitoring.

Then to print:

```
Serial.print("text" or variable)

Serial.println("text" or variable)

Serial.print("Arduino\r\n")
```

is the same as `Serial.println("Arduino")`

```
17  int i = 0;
18  void loop() {
19    // put your main code here, to run repeatedly:
20    digitalWrite(13, HIGH);
21    delay(250);
22    digitalWrite(13, LOW);
23    delay(250);
24
25    i++;
26    Serial.print("Number of blinks:");
27    Serial.println(i);
28  }
```

Serial Monitor output

```
Number of blinks:8
Number of blinks:9
Number of blinks:10
Number of blinks:11
Number of blinks:12
Number of blinks:13
Number of blinks:14
```

Arduino API link:
https://www.arduino.cc/reference/en/

Programming Basics Part 4 - other constructs

Arduino is similar to C++ so you can also use standard language features, most commonly:

- Create variables
- Make your own custom functions
- If/else if/else statements
- For loops, while loops

```
// CUSTOM VOID FUNCTION (no return value)
void custom_void_function() {
  Serial.println("Executing custom void function!");
}

// CUSTOM FUNCTION (returns int)
int square_value(int value) {
  int result;
  Serial.println("Executing square_value!");
  result = value*value;
  return result;
}
```

```
// COMMON VARIABLE DATATYPES
int value = analogRead(A0); // integer
float test_float = 0.02; // 32-bit precision
double test_double = 0.03; // 64-bit precision

// IF/ELSE STATEMENT
if (value > 50){
  custom_void_function();
}
else {
  Serial.println(square_value(value));
}

// FOR LOOP
for (int i = 0; i < 10; i++){
  Serial.println(i); // executes this loop 10 times
}
```

Arduino API link:
https://www.arduino.cc/reference/en/

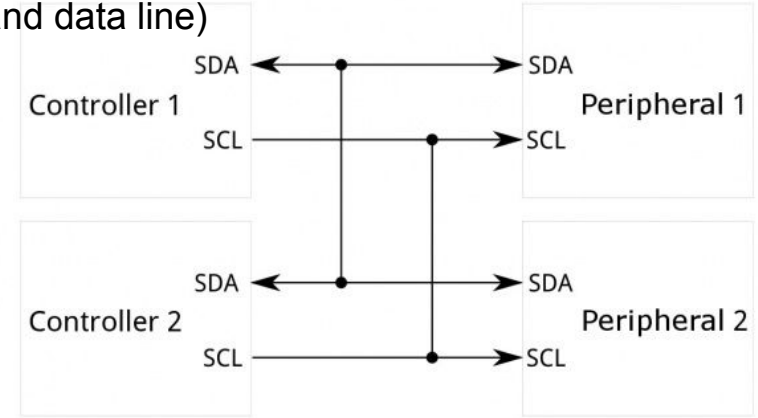
Working with Sensors

Arduino is very powerful for interfacing with sensor breakouts (distance, temperature, force, motion, touch, color, etc.). - browse some here: https://www.sparkfun.com/categories/23

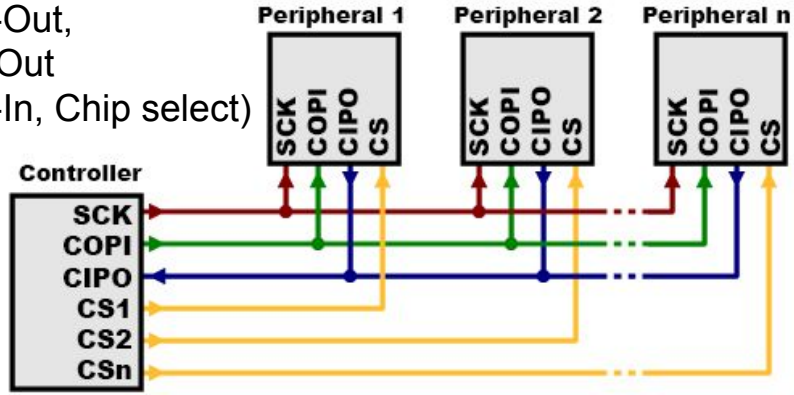
Some sensors simply output a digital or analog signal and can be read via the Arduino analog/digital pins.

You will find that many peripherals, especially those that require a high data rate, don't actually connect to the analog/digital pins. Instead, they use high-speed digital interfaces, the most common being **I2C** and **SPI.** The added benefit of these interfaces is that we can use multiple sensors on the same set of pins. I2C in particular can in theory support 127 peripherals on just two wires!

I2C Interface
(clock and data line)



SPI Interface
(Clock, Controller-In Peripheral-Out, Controller-Out Peripheral-In, Chip select)



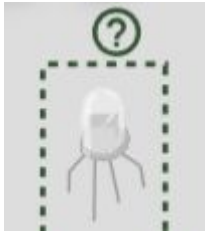
Using the Wokwi Arduino simulator

In EE 201, we are essentially making a modified “cloned” Arduino via the Arduometer. So we don’t have many actual Arduinos on hand.

The Wokwi Arduino simulator (https://wowki.com) is a powerful tool that can emulate many Arduino functions and includes models for a lot of popular components and sensors (purple + button to add new part)

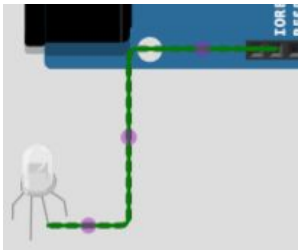
Real Arduino Unos are about $30 with cloned Unos <$20 if you want to have your own!

Click the (?) on a component to view its documentation/details

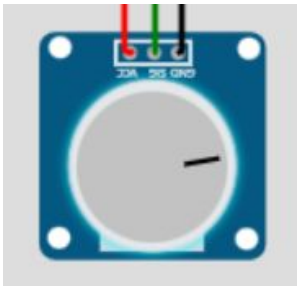


Hover on any pin to display its name

Click on wire sections to reshape them



While the simulation is running, simply click/drag on input components to interact with them



Samples

Here are some examples to get you started:

1. Basic example showing digital input/output, analog input, and serial monitor:

   https://wokwi.com/projects/345847592124416596

2. Basic example showing LED blink + counting LED blinks:

   https://wokwi.com/projects/345856220118123092

3. Example showing interfacing with the HC-SR04 distance sensor (click the (?) for the sensor to review its documentation). When running the simulation, click on the sensor to adjust the simulated distance:

   https://wokwi.com/projects/345858494329520722

Arduino API link:
https://www.arduino.cc/reference/en/