

Introduction to Arduino

Mahesh Neelakanta
Director of ITS
FAU Libraries

July 2020

credit where credit is due..

Most material in this training program is based on the Sparkfun [Activity Guide for SparkFun Tinker Kit](#) and is licensed similarly [Creative Commons Attribution Share-Alike 4.0 International License.](#)

Who are you and why are you doing this?

- Mahesh Neelakanta
 - Director of ITS
 - IT Director at Engineering for 13 years and recently moved to Libraries in Fall 2019
- Library technology outreach initiatives to bring various areas of technology to the general student, faculty and staff population
 - 3D Printing (ITS Webinar by Hansy, Crystal and Yom)
 - Cloud Computing
 - Artificial Intelligence and Cloud Computing

Housekeeping

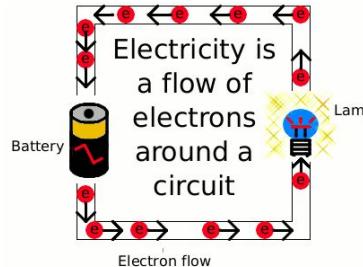
- Meeting is in webinar mode
 - Attendee mics will be muted (I can unmute if needed)
 - No video of attendees (only panelists)
 - Please ask your questions in Q&A and/or Chat
 - You can raise your hand as needed
- Polls
 - <https://meet.ps/mahesh>
- Slides and Documentation
 - <https://github.com/faumahesh/arduino>

Module 1 - Goals

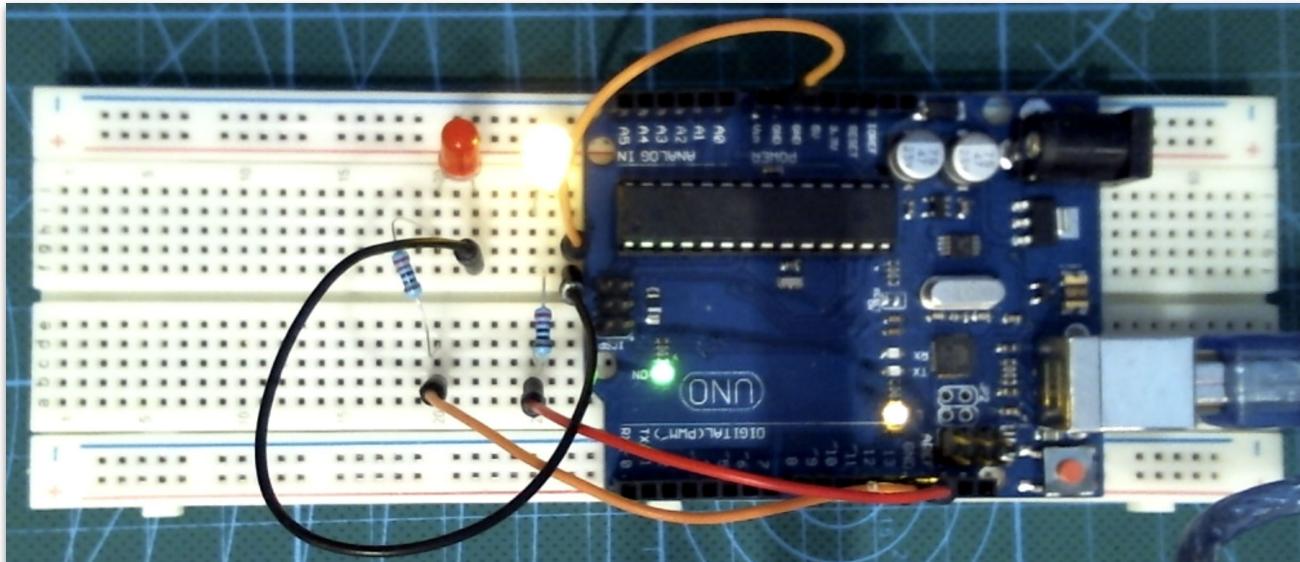
- Let's start with a demo!
- How is this series organized?
- What is Arduino and how to get started?
- What can you do with Arduino?
- The Sparkfun Tinkerkit Unboxing (<https://www.sparkfun.com/tinkerkit>)
- Let's Get Started!
 - Install Arduino Software
 - Circuit 1 - Blink an LED
- Module 2 Goals
- Additional Reading , Q&A

Let's start with a demo!

- Blinking LED

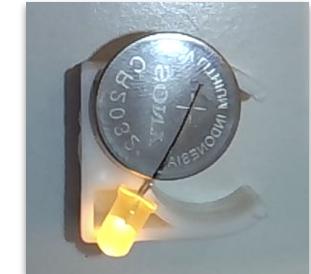


```
1 void setup() {  
2   pinMode(13, OUTPUT); |  
3   pinMode(12, OUTPUT);  
4 }  
5  
6  
7  
8 void loop() {  
9   digitalWrite(13, HIGH); // Turn on the LED  
10  digitalWrite(12, LOW); // Turn on the LED  
11  delay(1000); // Wait for two seconds  
12  digitalWrite(13, LOW); // Turn off the LED  
13  digitalWrite(12, HIGH); // Turn off the LED  
14  delay(1000); // Wait for two seconds  
15 }  
16 }
```



← Fancy Version

Cheap Version



How is this series organized?

- We will be covering the [Sparkfun Activity Guide](#) for the Tinker Kit over a period of 5-6 sessions
- Each session (except #2) will have 1-2 circuits from guide, ad-hoc circuit followed by a Q&A period
- Session 2 (next one) will focus on Arduino "C" language introduction
- This is a beginner course. At the end of the series, we will gauge interest for an intermediate class (WiFi, Shields) and an advanced class (IoT, Cloud)
- Sessions will be recorded for replay at FAU Libraries Youtube or Facebook
- Survey at the end of this session on what schedule works best for everyone
 - Weekly, twice a month, monthly?

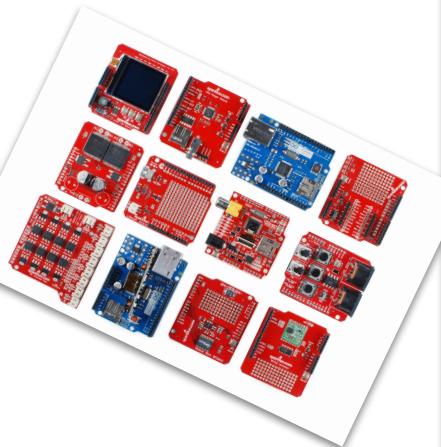
What is Arduino and how to get started?

- Easy to use hardware framework to learn (hands-on) electronics and coding
- Cheap enough to experiment-destroy-restart (\$10 - \$30)
- **Open source hardware** and lots of software to get started quickly
- Rich **ecosystem of "shields"** for easy extensibility - <https://store.arduino.cc/usa/arduino/shields>
- Common platform and jumping point to more advanced devices
- Program in "C" or Visually using Blockly (or Python with higher end units)
 - Arduino IDE (what we will be using) - <https://www.arduino.cc/en/main/software>
 - Otto Blockly - <https://github.com/OttoDIY/blockly>



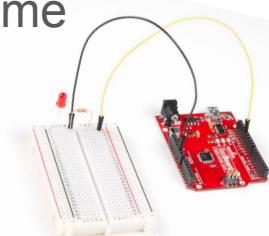
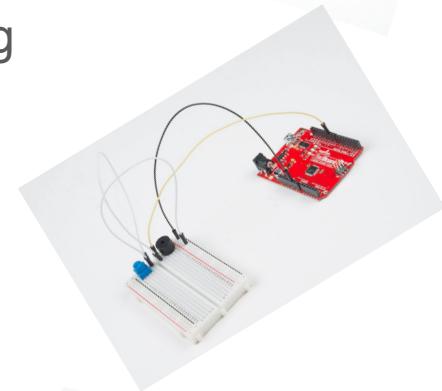
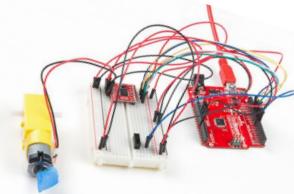
Arduino Boards (microcontrollers)

- Open Source hardware by Uno for entry level playground
 - Arduino Uno - \$23
 - SparkFun RedBoard - \$20
 - AdaFruit Metro - \$18
 - Amazon Generic Uno - \$13
 - Generic from China - \$4
- Due, Mega, Nano, Zero
 - Different form factors and capacities
 - https://www.sparkfun.com/standard_arduino_comparison_guide
 - <https://electropeak.com/learn/arduino-buying-guide-how-to-choose-right-arduino-board/>



What can you do with Arduino?

- Learn to code with an physical device (vs on desktop/web/app)
- Learn the basics of electronics and circuits by experimenting
- Build devices with sensors and displays
 - Check temperature, humidity, barometric pressure, light
 - Turn a switch on/off
 - Play a tune
 - Make a motor rotate
- Bond with your kids with hands on activities
- Enter into a much wider world of connected devices such as home automation and Internet-of-Things (IOT)
- What about you? <https://meet.ps/mahesh>



Sparkfun Tinker Kit Unboxing

- <https://www.sparkfun.com/products/14556>
- SparkFun RedBoard
- SparkFun Motor Driver (with Headers)
- Breadboard - Self-Adhesive (White)
- Servo -- Sub-Micro Size
- Hobby Gearmotor -- 200 RPM (Pair)
- Temperature Sensor -- TMP36
- Mini Photocell
- Piezoelectric Speaker
- SparkFun USB Mini-B Cable -- 6 Foot
- Jumper Wires -- 7" M/M 30 AWG (30 Pack)
- LED - RGB Diffused Common Cathode
- Red, Blue, Yellow, and Green LEDs
- Red, Blue, Yellow, and Green Buttons
- Mini Power Switch
- 10K Trimpot
- Battery Holder - 4xAA to Barrel Jack Connector
- 330 and 10K Resistors

Let's Get Started!

Arduino Software

1. Web based interface on <https://create.arduino.cc/>
2. Software simulator at <https://www.tinkercad.com/>
3. Download and install desktop client for Mac, Windows or Linux
 - a. <https://www.arduino.cc/en/Main/Software>
4. For this class, we will be using the desktop client

Install Arduino Software

<https://learn.sparkfun.com/tutorials/installing-arduino-ide>

The screenshot shows the Arduino IDE interface. The code in the editor is:

```
sketch_jul11a.ino
1 void setup() {
2   pinMode(13, OUTPUT);
3 }
4
5 void loop() {
6   digitalWrite(13, HIGH);
7   delay(1500);
8   digitalWrite(13, LOW);
9   delay(1500);
10 }
```

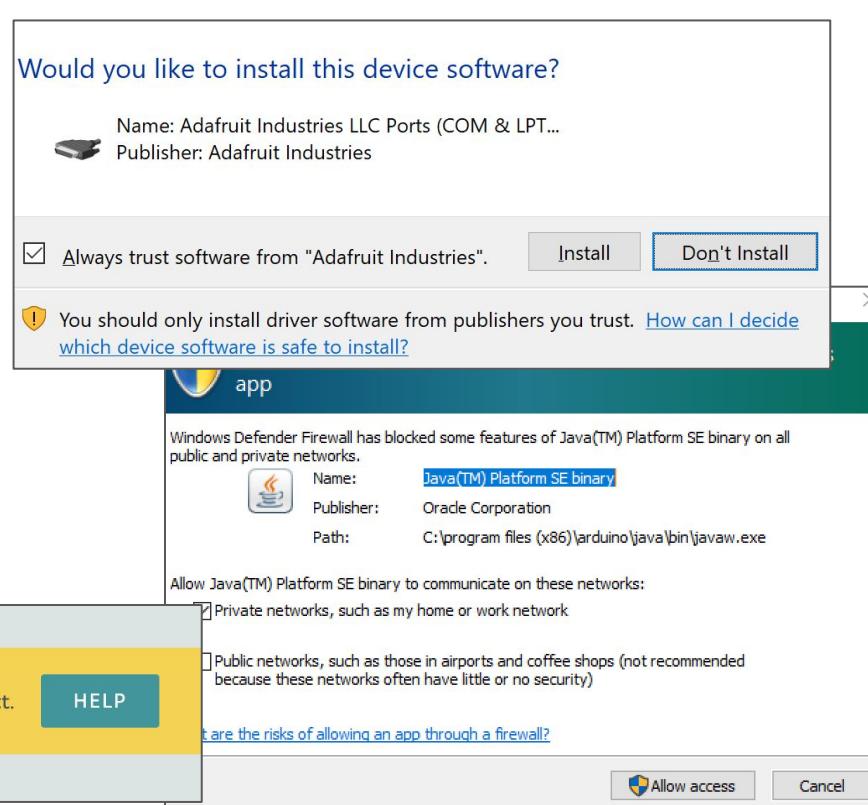
The serial monitor output shows the following:

```
sketch_jul11a | Arduino 1.8.13
sketch_jul11a:5
1 void setup() {
2   pinMode(13, OUTPUT);
3 }
4
5 void loop() {
6   digitalWrite(13, HIGH);
7   delay(1500);
8   digitalWrite(13, LOW);
9   delay(1500);
10 }

Library added to your libraries. Check "Include library" menu
SCK period      : 3.3 us

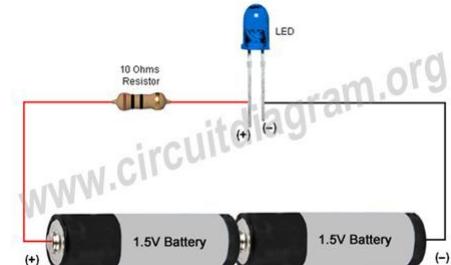
avrduude: AVR device initialized and ready to accept instructions
Reading | ##### | 100% 0.00s
avrduude: Device signature = 0x1e950f (probably m328p)
avrduude: reading input file '/var/folders/w0/08fxjx3l92t1gl8219s2jgffc000gn/T/arduino_sketch_jul11a.ino'
avrduude: writing flash (924 bytes):
Writing | ##### | 100% 0.16s
avrduude: 924 bytes of flash written
avrduude: verifying Flash memory against /var/folders/w0/08fxjx3l92t1gl8219s2jgffc000...
avrduude: load data Flash data from input file /var/folders/w0/08fxjx3l92t1gl8219s2jg

9
Arduino Uno on /dev/cu.usbmodem143101
```



Circuit 1 - Blink a LED

- <https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard>
- <https://learn.sparkfun.com/tutorials/redboard-hookup-guide>
- <https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/circuit-1-blink-an-led>
 - Build the circuit
 - Copy the code
 - Test the circuit
 - Modify the circuit to add a second LED
 - Modify the code and circuit to control the two LEDs separately
 - Modify the code to blink the two LEDs opposite of each other



Bonus! Arduino Simulator

<https://www.tinkercad.com/>

The screenshot shows the Tinkercad interface for an Arduino Uno R3 simulation. On the left, a breadboard diagram is displayed with a USB cable connected to the Arduino board. A simple circuit is built on the breadboard, featuring a red LED connected to pin 13 through a resistor. The Arduino board is shown with its various pins labeled. On the right, the code editor displays the following sketch:

```
/*
 * This program blinks pin 13 of the Arduino (the
 * built-in LED)
 */
void setup()
{
    pinMode(13, OUTPUT);
}
void loop()
{
    // turn the LED on (HIGH is the voltage level)
    digitalWrite(13, HIGH);
    delay(1000); // Wait for 1000 milliseconds
    // turn the LED off by making the voltage LOW
    digitalWrite(13, LOW);
    delay(1000); // Wait for 1000 milliseconds
}
```

The code uses the standard Arduino pin mapping where pin 13 corresponds to the built-in LED. The sketch alternates the digital output between HIGH and LOW states every second, effectively creating a continuous blink effect.

Additional Reading

- <https://learn.sparkfun.com/tutorials/what-is-an-arduino>
- <https://learn.sparkfun.com/tutorials/what-is-a-circuit>
- <https://www.arduino.cc/en/Guide/Introduction>
- https://files.seeedstudio.com/wiki/Book_and_stickers/A_Brief_Intro_to_Electronics.pdf
- <https://learn.sparkfun.com/tutorials/redboard-hookup-guide>

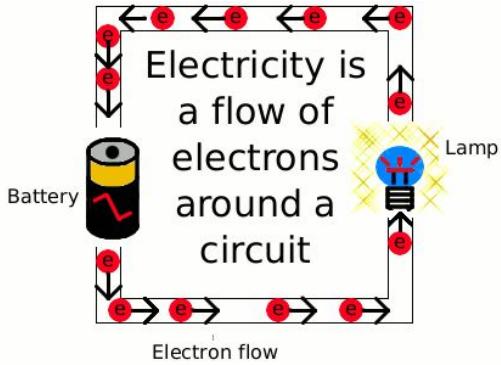
Q&A

Contact information : mahesh@fau.edu

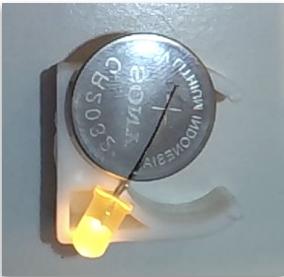
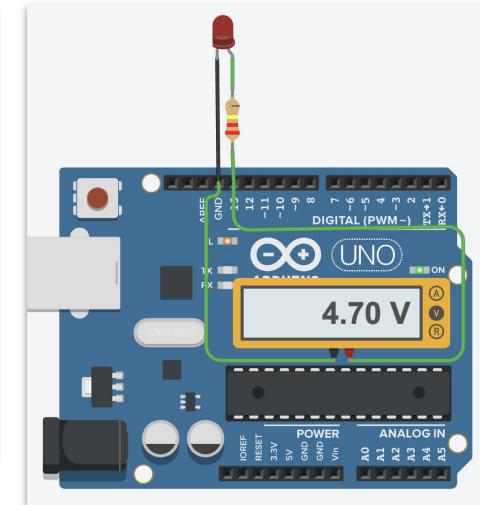
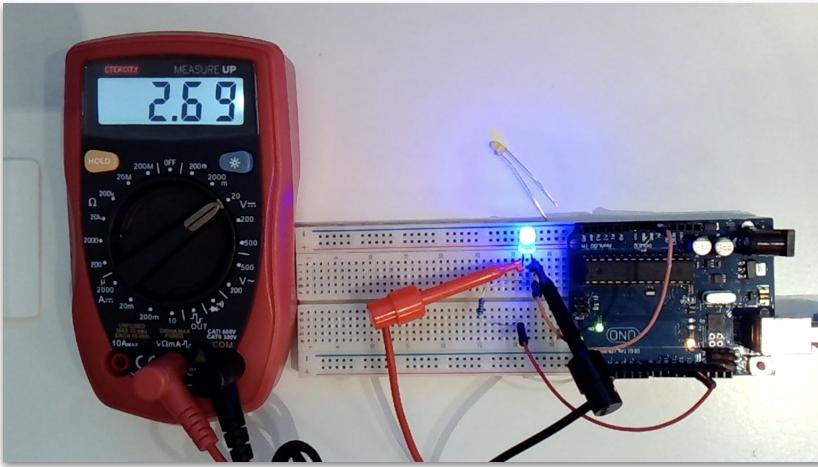
Introduction to Arduino - Module 2

- Questions from previous session?
- Blinking LED with a multimeter to measure the electric voltage
- Morse Code SOS example
- Arduino Language Tutorial
- Extending Blinking LEDs with some logic

A quick detour on electricity

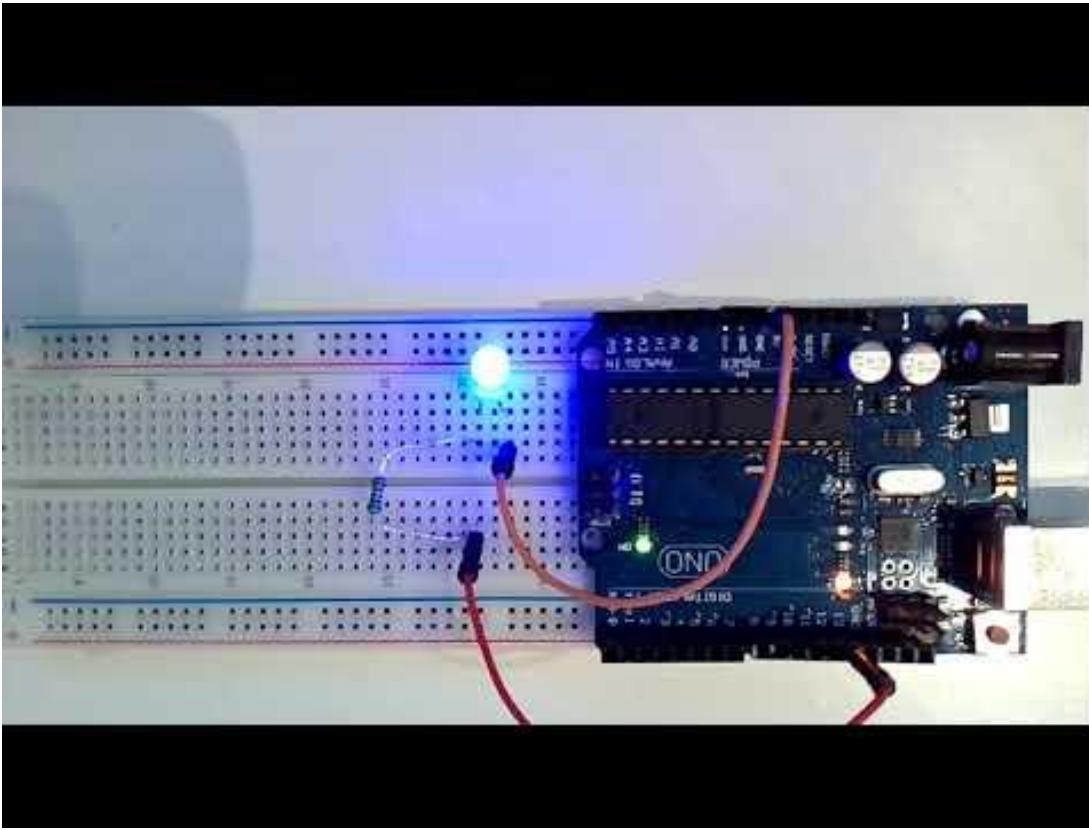


<https://rme.sharylandisd.org/cms/One.aspx?portalId=418090&pageId=2408938>



Blinking LED with a multimeter to measure the electric voltage

Morse Code SOS Example



https://github.com/faumahesh/arduino/tree/master/morse_simple

morse_simple

```
1 //  
2 // Simple Morse SOS demo  
3 // 7/2020 - Mahesh Neelakanta  
4 //  
5  
6 #define LEDPIN 13  
7 #define DOTDELAY 200 // How long to wait per dot  
8 #define DASHDELAY 350 // How long to wait per dash  
9 #define CHARDELAY 500 // How long to wait after end of each character  
10 #define EOMDELAY 1000 // How long to wait after end of message  
11  
12 void setup()  
13 {  
14   pinMode(LEDPIN, OUTPUT);  
15 }  
16  
17 void loop()  
18 {  
19   // SOS = ... --- ...  
20  
21   dotBlink();  
22   dotBlink();  
23   dotBlink();  
24   delay(CHARDELAY);  
25  
26   dashBlink();  
27   dashBlink();  
28   dashBlink();  
29   delay(CHARDELAY);  
30  
31   dotBlink();  
32   dotBlink();  
33   dotBlink();  
34   delay(EOMDELAY);  
35  
36 }  
37  
38 void dotBlink()  
39 {  
40   digitalWrite(LEDPIN, HIGH); // Turn on the LED  
41   delay(DOTDELAY); // Leave LED on for a short (DOT) delay  
42   digitalWrite(LEDPIN, LOW); // Turn off the LED  
43   delay(DOTDELAY); // Leave LED on for a short (DOT) delay  
44  
45 }  
46  
47 void dashBlink()  
48 {  
49   digitalWrite(LEDPIN, HIGH); // Turn on the LED  
50   delay(DASHDELAY); // Leave LED on for a long (DASH) delay  
51   digitalWrite(LEDPIN, LOW); // Turn off the LED  
52   delay(DASHDELAY); // Leave LED on for a long (DASH) delay  
53 }
```

Arduino Language

- Variation of the C/C++ Language.
- Core Language
 - Structure
 - Data Types, Variables and Constants
 - Functions
 - <https://www.arduino.cc/reference>
- Libraries (Extensions)
 - <https://www.arduino.cc/reference/en/libraries/>
 - Communication (715)
 - Data Processing (152)
 - Data Storage (90)
 - Device Control (510)
 - Display (305)
 - Other (275)
 - Sensors (627)
 - Signal Input/Output (249)
 - Timing (139)
 - Uncategorized (105)

STRUCTURE

The elements of Arduino (C++) code.

Sketch

loop()

setup()

Control Structure

break

continue

do...while

else

for

goto

if

return

switch...case

while

Further Syntax

#define (define)

#include (include)

/* */ (block comment)

// (single line comment)

; (semicolon)

{ (curly braces)

Arithmetic Operators

% (remainder)

* (multiplication)

+ (addition)

- (subtraction)

/ (division)

= (assignment operator)

Comparison Operators

!= (not equal to)

< (less than)

<= (less than or equal to)

== (equal to)

> (greater than)

>= (greater than or equal to)

Boolean Operators

! (logical not)

&& (logical and)

|| (logical or)

Pointer Access Operators

& (reference operator)

* (dereference operator)

Bitwise Operators

& (bitwise and)

<< (bitshift left)

>> (bitshift right)

^ (bitwise xor)

| (bitwise or)

~ (bitwise not)

Compound Operators

%= (compound remainder)

&= (compound bitwise and)

*= (compound multiplication)

++ (increment)

+= (compound addition)

-- (decrement)

-= (compound subtraction)

/= (compound division)

^= (compound bitwise xor)

|= (compound bitwise or)

VARIABLES

Arduino data types and constants.

Constants

HIGH | LOW

INPUT | OUTPUT | INPUT_PULLUP

LED_BUILTIN

true | false

Floating Point Constants

Integer Constants

Conversion

(unsigned int)

(unsigned long)

byte()

char()

float()

int()

long()

word()

Data Types

array

bool

boolean

byte

char

double

float

int

long

short

size_t

string

String()

unsigned char

unsigned int

unsigned long

void

word

Variable Scope & Qualifiers

const

scope

static

volatile

Utilities

PROGMEM

sizeof()

FUNCTIONS

For controlling the Arduino board and performing computations.

Digital I/O	Math	Random Numbers
<code>digitalRead()</code>	<code>abs()</code>	<code>random()</code>
<code>digitalWrite()</code>	<code>constrain()</code>	<code>randomSeed()</code>
<code>pinMode()</code>	<code>map()</code>	
	<code>max()</code>	
Analog I/O	<code>min()</code>	Bits and Bytes
<code>analogRead()</code>	<code>pow()</code>	<code>bit()</code>
<code>analogReference()</code>	<code>sq()</code>	<code>bitClear()</code>
<code>analogWrite()</code>	<code>sqrt()</code>	<code>bitRead()</code>
		<code>bitSet()</code>
		<code>bitWrite()</code>
Zero, Due & MKR Family	Trigonometry	
<code>analogReadResolution()</code>	<code>cos()</code>	<code>highByte()</code>
<code>analogWriteResolution()</code>	<code>sin()</code>	<code>lowByte()</code>
	<code>tan()</code>	
		External Interrupts
Advanced I/O	Characters	
<code>noTone()</code>	<code>isAlpha()</code>	<code>attachInterrupt()</code>
<code>pulseIn()</code>	<code>isAlphaNumeric()</code>	<code>detachInterrupt()</code>
<code>pulseInLong()</code>	<code>isAscii()</code>	Interrupts
<code>shiftIn()</code>	<code>isControl()</code>	<code>interrupts()</code>
<code>shiftOut()</code>	<code>isDigit()</code>	<code>noInterrupts()</code>
<code>tone()</code>	<code>isGraph()</code>	
	<code>isHexadecimalDigit()</code>	Communication
Time	<code>isLowerCase()</code>	<code>Serial</code>
<code>delay()</code>	<code>isPrintable()</code>	<code>Stream</code>
<code>delayMicroseconds()</code>	<code>isPunct()</code>	
<code>micros()</code>	<code>isSpace()</code>	
<code>millis()</code>	<code>isUpperCase()</code>	USB
	<code>isWhitespace()</code>	Keyboard
		Mouse

The next set of slides have significant content taken from the arduino language reference guide:

<https://www.arduino.cc/reference/en/>

Control Structure

Comments, Curly Braces and Semicolons

```
/* This is a valid comment */

/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.

This example code is in the public domain.
(Another valid comment)
*/

/*
if (gwb == 0) { // single line comment is OK inside a multi-line comment
    x = 3;          /* but not another multi-line comment - this is invalid */
}
// don't forget the "closing" comment - they have to be balanced!
*/
```

```
// pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;
digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
```

```
int a = 13;
```

Functions

```
void myfunction(datatype argument) {
    // any statement(s)
}
```

Loops

```
while (boolean expression) {
    // any statement(s)
}

do {
    // any statement(s)
} while (boolean expression);
```

```
for (initialisation; termination condition; incrementing expr) {
    // any statement(s)
}
```

Conditional Statements

```
if (boolean expression) {
    // any statement(s)
}

else if (boolean expression) {
    // any statement(s)
}

else {
    // any statement(s)
}
```

if

[Control Structure]

Description

The **if** statement checks for a condition and executes the proceeding statement or set of statements if the condition is 'true'.

Syntax

```
if (condition) {  
    //statement(s)  
}
```

Parameters

condition: a boolean expression (i.e., can be **true** or **false**).

Example Code

The brackets may be omitted after an if statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement.

```
if (x > 120) digitalWrite(LEDpin, HIGH);  
  
if (x > 120)  
digitalWrite(LEDpin, HIGH);  
  
if (x > 120) {digitalWrite(LEDpin, HIGH);}  
  
if (x > 120) {  
    digitalWrite(LEDpin1, HIGH);  
    digitalWrite(LEDpin2, HIGH);  
}  
// all are correct
```

else

[Control Structure]

Description

The **if...else** allows greater control over the flow of code than the basic **if** statement, by allowing multiple tests to be grouped. An **else** clause (if at all exists) will be executed if the condition in the **if** statement results in **false**. The **else** can proceed another **if** test, so that multiple, mutually exclusive tests can be run at the same time.

Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire if/else construction. If no test proves to be true, the default **else** block is executed, if one is present, and sets the default behavior.

Note that an **else if** block may be used with or without a terminating **else** block and vice versa. An unlimited number of such **else if** branches are allowed.

Syntax

```
if (condition1) {  
    // do Thing A  
}  
else if (condition2) {  
    // do Thing B  
}  
else {  
    // do Thing C  
}
```

Example Code

Below is an extract from a code for temperature sensor system

```
if (temperature >= 70) {  
    // Danger! Shut down the system.  
}  
else if (temperature >= 60) { // 60 <= temperature < 70  
    // Warning! User attention required.  
}  
else { // temperature < 60  
    // Safe! Continue usual tasks.  
}
```

while

[Control Structure]

Description

A **while** loop will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

Syntax

```
while (condition) {  
    // statement(s)  
}
```

Parameters

condition: a boolean expression that evaluates to **true** or **false**.

Example Code

```
var = 0;  
while (var < 200) {  
    // do something repetitive 200 times  
    var++;  
}
```

do...while

[Control Structure]

Description

The `do...while` loop works in the same manner as the `while` loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

Syntax

```
do {  
    // statement block  
} while (condition);
```

Parameters

`condition`: a boolean expression that evaluates to `true` or `false`.

Example Code

```
int x = 0;  
do {  
    delay(50);          // wait for sensors to stabilize  
    x = readSensors();  // check the sensors  
} while (x < 100);
```

for

[Control Structure]

Description

The **for** statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The **for** statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

Syntax

```
for (initialization; condition; increment) {  
    // statement(s);  
}
```

Parameters

initialization: happens first and exactly once.

condition: each time through the loop, **condition** is tested; if it's **true**, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes **false**, the loop ends.

increment: executed each time through the loop when **condition** is **true**.

Example Code

```
// Dim an LED using a PWM pin  
int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10  
  
void setup() {  
    // no setup needed  
}  
  
void loop() {  
    for (int i = 0; i <= 255; i++) {  
        analogWrite(PWMpin, i);  
        delay(10);  
    }  
}
```

break

[Control Structure]

Description

break is used to exit from a **for**, **while** or **do...while** loop, bypassing the normal loop condition. It is also used to exit from a **switch case** statement.

Example Code

In the following code, the control exits the **for** loop when the sensor value exceeds the threshold.

```
int threshold = 40;
for (int x = 0; x < 255; x++) {
    analogWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold) {      // bail out on sensor detect
        x = 0;
        break;
    }
    delay(50);
}
```

continue

[Control Structure]

Description

The **continue** statement skips the rest of the current iteration of a loop (**for**, **while**, or **do...while**). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iterations.

Example Code

The following code writes the value of 0 to 255 to the **PWMpin**, but skips the values in the range of 41 to 119.

```
for (int x = 0; x <= 255; x++) {
  if (x > 40 && x < 120) { // create jump in values
    continue;
  }

  analogWrite(PWMpin, x);
  delay(50);
}
```

Comparison Operators

Comparison Operators:

```
x == y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)
```

```
if (x > y) { // tests if x is greater (bigger) than y  
  
    // do something only if the comparison result is true  
  
}  
  
if (x <= y) { // tests if x is less (smaller) than or equal to y  
  
    // do something only if the comparison result is true  
  
}  
  
if (x == y) { // tests if x is equal to y  
  
    // do something only if the comparison result is true  
  
}  
  
if (x != y) { // tests if x is not equal to y  
  
    // do something only if the comparison result is true  
  
}
```

Math Operators

Operator Symbol	Description	Example
=	Store the value of the right hand side to the "variable" on the left hand side	<pre>int temp; temp = analogRead(A1);</pre>
+	Add the left and right hand side values	<pre>int a = 10; int b = 20; int c; c = a + b ;</pre>
-	Subtract the right hand side value from left hand side value	<pre>int a = 10; int b = 20; int c; c = a - b ;</pre>
*	Multiply the left and right hand side values	<pre>float tF = 75.0; float tC; tC = tF * 1.8 + 32.0;</pre>
/	Divide the right hand side value from left hand side value	<pre>float tC = 23.0; float tF; tF = (tC - 32.0) / 1.8;</pre>

Boolean Operators

Boolean Operators

```
if (!x) { // if x is not true  
  // statements  
}
```

```
if (digitalRead(2) == HIGH && digitalRead(3) == HIGH) { // if BOTH the switches read HIGH  
  // statements  
}
```

```
if (x > 0 || y > 0) { // if either x or y is greater than zero  
  // statements  
}
```

Boolean Operators

! (logical not)

&& (logical and)

|| (logical or)

Compound Operators

Compound Operators

`%=` (compound remainder)

`&=` (compound bitwise and)

`*=` (compound multiplication)

`++` (increment)

`+=` (compound addition)

`--` (decrement)

`-=` (compound subtraction)

`/=` (compound division)

`^=` (compound bitwise xor)

`|=` (compound bitwise or)

```
x = 1; x++; // x will be 2

y = 7 ; y--; // y will be 6

x += 5; // x will be 7

y -= 3; // y will be 3

z = 10; z /= 5; // z will be 2

if (x == 7) { x++; } // x will be 8

if (x == 6 || y == 3) { y += 7 } // y will be 10

if (x == 6 && y == 3) { y += 7 } // y will remain 10
```

Data Types, Variables and Constants

Data Types

Data types are used to define the type of information (data) that is to be computed, acquired, modified, stored, sent, etc. Different data types use up different amounts of storage (memory). The most common data types include:

- byte : integer values between 0 and 255 (unsigned)
- char : ASCII character values. Ex: 'a', 'A', '1', '2', 'Z', '&...'
- int : integer values between -32768 to +32767 (signed)
- long : integer values between -2,147,483,648 to 2,147,483,647 (signed)
- boolean : true or false values
- float : decimal values between 3.4028235E+38 and -3.4028235E+38
- string : a list (array) of characters
- unsigned : prefix to make a int, long, etc. become just 0 and positive values

Variables

Variables are values of a certain data type that is used to store values that can change over time. In Arduino C, the variables can be defined using a data type prefix and can be initialized to a value or assigned a value later in the program.
Examples:

```
int a = 10;

float b = 3.14159;

String name[] = "fau";

long bignum = 0;
bignum = 32000 * 32000;

ledON = false;

char sos[] = "SOS";
```

```
int a = 5;
int b = 10;
int c = a * b;

int r;
float pi = 3.14159;

r = a + b + c;

area = pi * r * r;
```

Constants

Constants are just like variables except that their values are fixed (constant) and cannot be changed. Constant

```
int r = 10;
const float pi = 3.14159;

area = pi * r * r;

pi = 2.7182 ; // this will give a compiler error
```

```
// Pre-defined constants

pinMode(13, OUTPUT);
digitalWrite(13, HIGH);

boolean ledOn = true;
boolean motorOn = false;
```

Constants take up memory space (ram). If you are just trying to have a word representation for a value, you can also use `#define`

```
#define ledPin 3
// The compiler will replace any mention of ledPin with the value 3 at compile time.
```

Array Variables

Arrays are variables that have a list of values where each value is accessed using an index (normally an integer)

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

```
int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};
// myArray[9]      contains 11
// myArray[10]     is invalid and contains random information (other memory address)
```

```
mySensVals[0] = 10;
```

```
x = mySensVals[4];
```

```
for (byte i = 0; i < 5; i = i + 1) {
    Serial.println(myPins[i]);
}
```

Functions

What are functions?

Functions are used to improve programming style by segmenting different tasks (or tasks that need to be repeated) into their own sections. This allows for breaking a larger problem down into sub-problems. Functions are used in arduino in a variety of ways including system functions like ***setup()*** and ***loop()*** for initialization and processing, user-written functions to split a larger task into subtasks and libraries of 3rd party written functions (methods) that extend the arduino ecosystem.

setup() and loop()

```
blink_two_leds
1
2 void setup() {
3   pinMode(13, OUTPUT);
4   pinMode(12, OUTPUT);
5 }
6
7
8 void loop() {
9   digitalWrite(13, HIGH);    // Turn on the LED
10  digitalWrite(12, LOW);    // Turn on the LED
11  delay(1000);            // Wait for two seconds
12  digitalWrite(13, LOW);   // Turn off the LED
13  digitalWrite(12, HIGH);  // Turn off the LED
14  delay(1000);            // Wait for two seconds
15 }
```

setup() is a function which is run (called) when the arduino first starts up or is reset (via reset button). It is run only once for each startup or reset.

loop() is a function which is run (called), It is called repeatedly forever, until the arduino is turned off or an exception (crash) occurs.

User functions

```
17 void loop()
18 {
19 // SOS = ... --- ...
20
21 dotBlink();
22 dotBlink();
23 dotBlink();
24 delay(CHARDELAY);
25
26 dashBlink();
27 dashBlink();
28 dashBlink();
29 delay(CHARDELAY);
30
31 dotBlink();|
32 dotBlink();
33 dotBlink();
34 delay(EOMDELAY);
35
36 }
37
38 void dotBlink()
39 {
40 digitalWrite(LEDPIN, HIGH); // Turn on the LED
41 delay(DOTDELAY); // Leave LED on for a short (DOT) delay
42 digitalWrite(LEDPIN, LOW); // Turn off the LED
43 delay(DOTDELAY); // Leave LED on for a short (DOT) delay
44
45 }
46
47 void dashBlink()
48 {
49 digitalWrite(LEDPIN, HIGH); // Turn on the LED
50 delay(DASHDELAY); // Leave LED on for a long (DASH) delay
51 digitalWrite(LEDPIN, LOW); // Turn off the LED
52 delay(DASHDELAY); // Leave LED on for a long (DASH) delay
53 }
```

Arduino Core Library

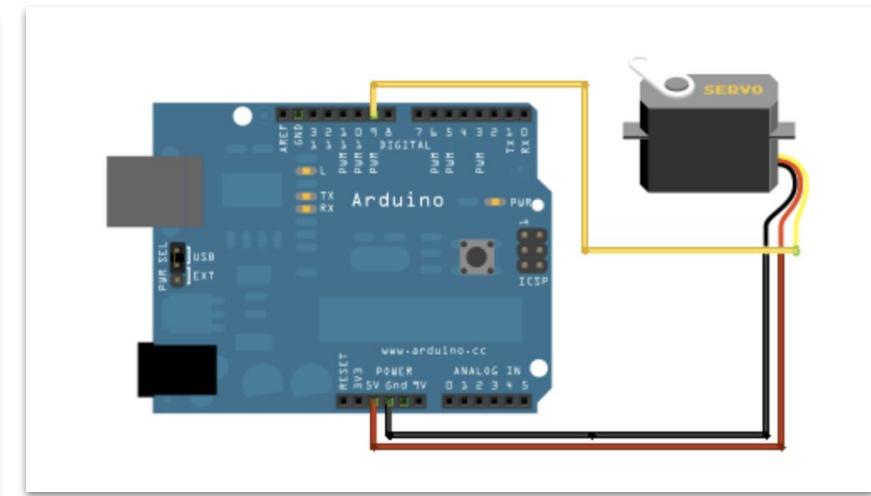
Digital I/O	Math	Random Numbers
digitalRead()	abs()	random()
digitalWrite()	constrain()	randomSeed()
pinMode()	map()	
	max()	
	min()	
Analog I/O	pow()	Bits and Bytes
analogRead()	sq()	bit()
analogReference()	sqrt()	bitClear()
analogWrite()		bitRead()
		bitSet()
		bitWrite()
		highByte()
Zero, Due & MKR Family	Trigonometry	lowByte()
	cos()	
analogReadResolution()	sin()	
analogWriteResolution()	tan()	
		External Interrupts
Advanced I/O	Characters	
noTone()	isAlpha()	
pulseIn()	isAlphaNumeric()	
pulseInLong()	isAscii()	Interrupts
shiftIn()	isControl()	interrupts()
shiftOut()	isDigit()	noInterrupts()
tone()	isGraph()	
	isHexadecimalDigit()	
Time	isLowerCase()	Communication
delay()	isPrintable()	
delayMicroseconds()	isPunct()	
micros()	isSpace()	Serial
millis()	isUpperCase()	Stream
	isWhitespace()	
		USB
		Keyboard
		Mouse

Libraries

Sweep.ino

```
1 /* Sweep
2 by BARRAGAN <http://barraganstudio.com>
3 This example code is in the public domain.
4
5 modified 8 Nov 2013
6 by Scott Fitzgerald
7 http://www.arduino.cc/en/Tutorial/Sweep
8 */
9
10 #include <Servo.h>
11
12 Servo myservo; // create servo object to control a servo
13 // twelve servo objects can be created on most boards
14
15 int pos = 0; // variable to store the servo position
16
17 void setup() {
18   myservo.attach(9); // attaches the servo on pin 9 to the servo object
19 }
20
21 void loop() {
22   for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
23     // in steps of 1 degree
24     myservo.write(pos); // tell servo to go to position in variable 'pos'
25     delay(15); // waits 15ms for the servo to reach the position
26   }
27   for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
28     myservo.write(pos); // tell servo to go to position in variable 'pos'
29     delay(15); // waits 15ms for the servo to reach the position
30   }
31 }
```

<https://www.arduino.cc/reference/en/libraries/servo/>



Methods

- `attach()`
- `write()`
- `writeMicroseconds()`
- `read()`
- `attached()`
- `detach()`

Let's put it all together!

Extending Blinking LEDs with some logic

```
1 //  
2 // Medium Morse SOS demo  
3 // 7/2020 - Mahesh Neelakanta  
4 //  
5 /*  
6     Added const, loops, if-else, comparison operators  
7     math operators, compound operators, data types  
8 */  
9 */  
10  
11 #define DOTDELAY 200    // How long to wait per dot  
12 #define DASHDELAY 350   // How long to wait per dash  
13 #define CHARDELAY 500   // How long to wait after end of each character  
14 #define EOMDELAY 1000   // How long to wait after end of message  
15  
16 const int LEDPIN = 13;  
17 const char msg[] = "AB";  
18  
19  
20 void dot()  
21 {  
22     digitalWrite(LEDPIN, HIGH); // Turn on the LED  
23     delay(DOTDELAY);        // Leave LED on for a short (DOT) delay  
24     digitalWrite(LEDPIN, LOW); // Turn off the LED  
25     delay(DOTDELAY);        // Leave LED on for a short (DOT) delay  
26 }  
27  
28 void dash()  
29 {  
30     digitalWrite(LEDPIN, HIGH); // Turn on the LED  
31     delay(DASHDELAY);        // Leave LED on for a long (DASH) delay  
32     digitalWrite(LEDPIN, LOW); // Turn off the LED  
33     delay(DASHDELAY);        // Leave LED on for a long (DASH) delay  
34 }
```

```
37 void setup()  
38 {  
39     Serial.begin(115200);  
40  
41     pinMode(LEDPIN, OUTPUT);  
42 }  
43  
44 void loop()  
45 {  
46     int index;  
47     char letter;  
48  
49     Serial.print("Printing: ");  
50     Serial.print(msg);  
51     Serial.print(" Length=");  
52     Serial.println(sizeof(msg)-1);  
53  
54     for(index = 0; index < sizeof(msg)-1 ; index++) {  
55         letter = msg[index];  
56         Serial.println(letter);  
57  
58         if (letter == 'A' || letter == 'a') {  
59             dot(); dash();  
60         }  
61         else if (letter == 'B' || letter == 'b') {  
62             dash(); dot(); dot(); dash();  
63         }  
64         else if (letter == 'S' || letter == 's') {  
65             dot(); dot(); dot();  
66         }  
67         else if (letter == 'O' || letter == 'o') {  
68             dash(); dash(); dash();  
69         }  
70         else {  
71             // Unknown value so skip it  
72         }  
73         delay(CHARDELAY);  
74     }  
75     delay(EOMDELAY);  
76 }
```

Q&A

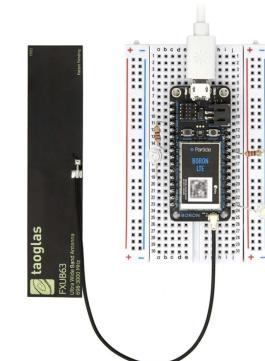
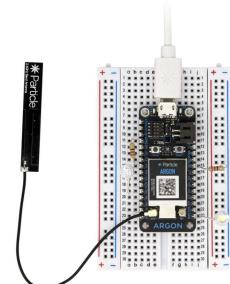
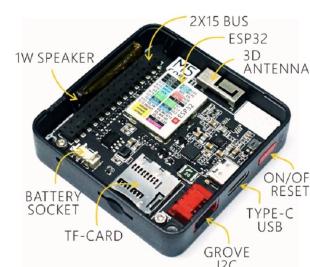
Contact information : mahesh@fau.edu

Introduction to Arduino - Module 3

- Questions from previous session?
- What are some other embedded devices?
- Electronics - resistors, schematics and Ohm's law
- Circuit 2 - Potentiometer
- Analog vs Digital
- Circuit 3 - Photoresistor
- Q&A

Questions ?

What are some other embedded devices? WiFi / Cell



ESP8266 (NodeMCU)

ESP32 (ex: m5stack.com)

particle.io

What is a resistor?

- A resistor reduces (or resists) the flow of electrical energy across it.
- "slows" the flow of electrons by causing it to flow across different materials or longer distance.
- Usually implemented as a small part that has a lot of wire wound tightly around it.





Electronics - resistors

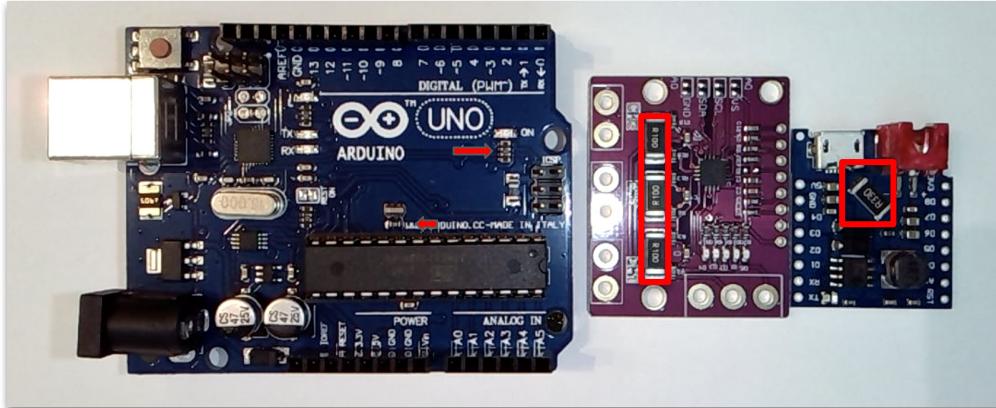
- Contrary to the BORG, resistance is extremely important in the digital world!
- Resistors are used
 - To protect circuits by reducing electrical flow
 - To dissipate energy
 - To control electrical flow to control devices (ex: light bulbs and motors)
 - In sensors to detect or measure light, temperature, humidity, force/direction
- Resistors are commonplace in most electrical circuits along with other components such as capacitors, diodes, transistors and inductors and are part of the core building blocks of any circuit.
- Other components include microcontrollers (MCU), integrated circuits (IC), transformers, relays, switches, motors, fuses/breakers, crystals and batteries
(<https://www.alliedcomponents.com/blog/essential-electronic-components-functions/>)

Types of resistors

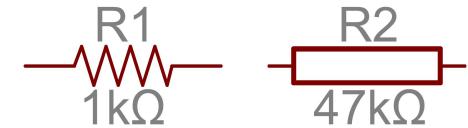
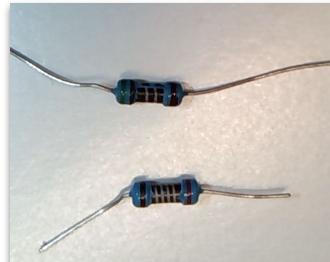


Plate-Through-Hole (PTH) Resistors

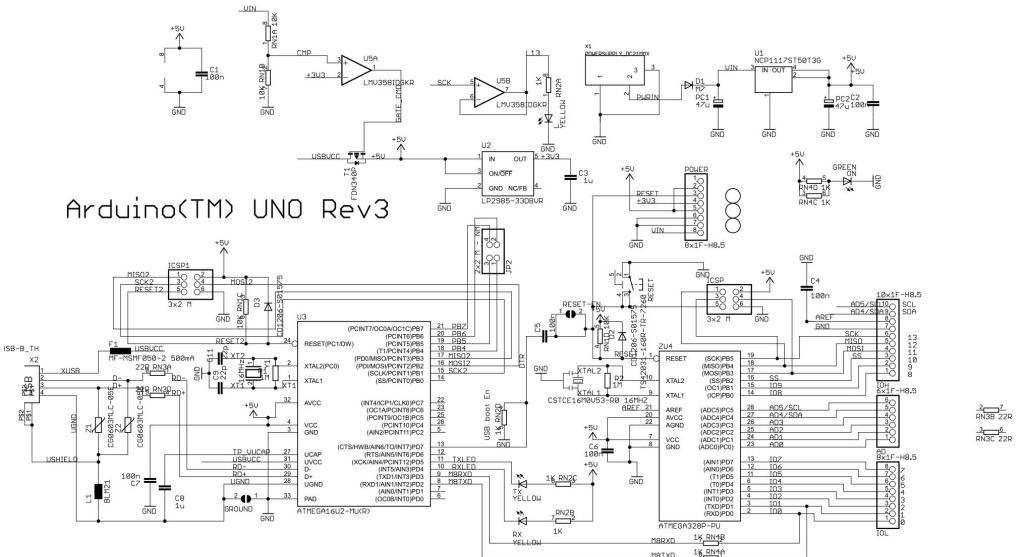
<https://learn.sparkfun.com/tutorials/resistors>



Surface Mount Device (SMD) Resistors



Electronics Schematics

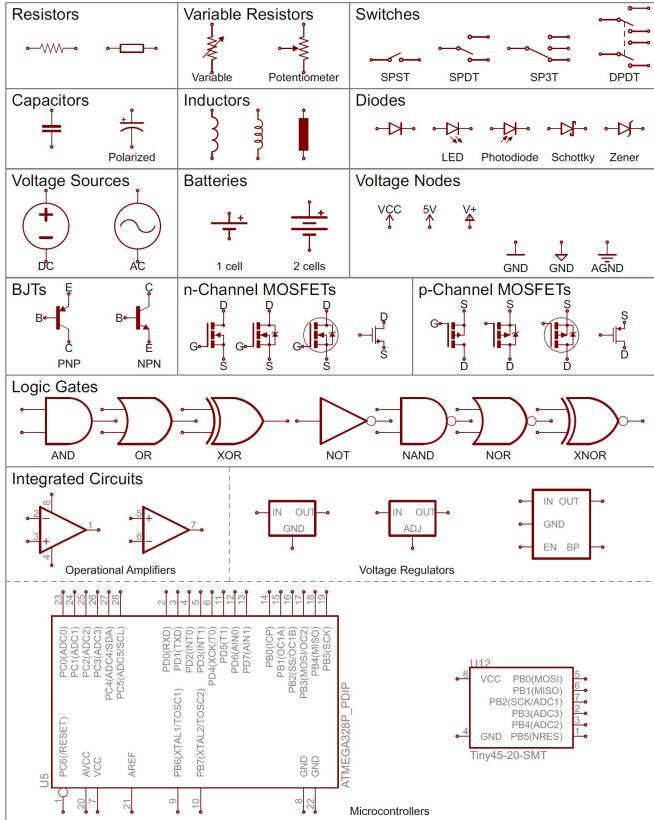


https://www.arduino.cc/en/uploads/Main/Arduino_Uino_Rev3-schematic.pdf

https://www.allaboutcircuits.com/uploads/articles/Arduino_schematic.png

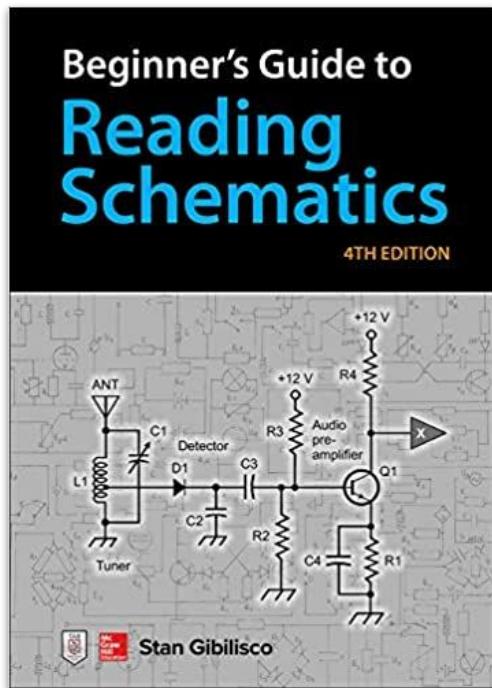
Want to know more about arduino uno's components?

<https://blog.make2explore.com/overview-of-arduino-uno-hardware-1/>

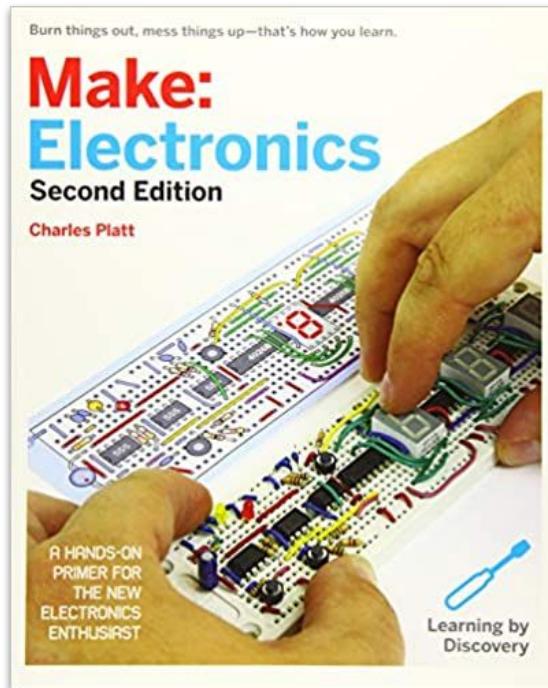


<https://learn.sparkfun.com/tutorials/how-to-read-a-schematic>

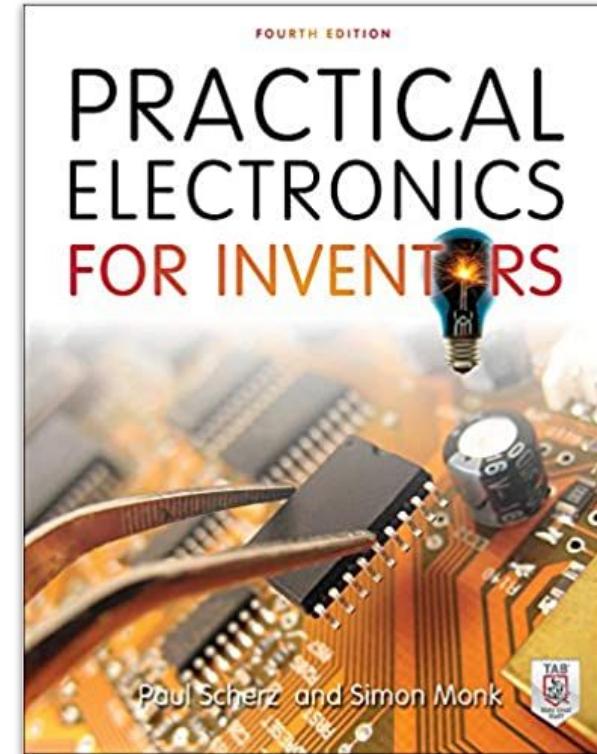
Books on electronics



<https://www.amazon.com/Beginners-Guide-Reading-Schematics-Fourth/dp/1260031101>



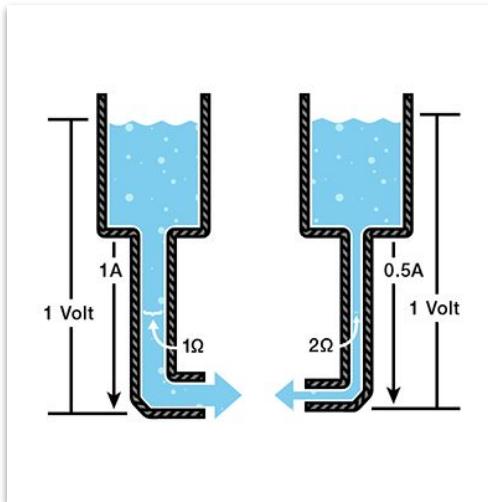
<https://www.amazon.com/Make-Electronics-Learning-Through-Discovery/dp/1680450263/>



<https://www.amazon.com/Practical-Electronics-Inventors-Fourth-Scherz/dp/1259587541/>

Ohm's Law

- Voltage = Current × Resistance
 - $V = I \times R$



$$1V = 1A \cdot 1\Omega$$

$$1V = ?A \cdot 2\Omega$$

$$1V = 0.5A \cdot 2\Omega$$

- Full details at <https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/ohms-law>

Protecting a LED using resistors

- A normal LED can only run on 18-20 mA (milliamps) of current. So if we connect a LED directly to a 9v battery, the current would be:
 - $9v = I \times R$ (in this case, without a resistor, R would be very very small.
 - $I = 9v \div 0.0000000001 \Rightarrow$ a very large number \Rightarrow destroyed LED
- To protect the LED, we should put a resistor in the circuit. Since we know the recommended current of a LED (I), we can solve for R
 - $R = V \div I$
 - $R = 9v \div 0.0018 \Rightarrow 500\Omega$ (Ohms)
 - So use a 560Ω (closest usable one). Examples here will use 330Ω
- <https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law/an-ohms-law-experiment>

Circuit 2 - Potentiometer

- A potentiometer (aka POT) is a *variable* resistor.
- By turning a knob, you can control (vary) the amount of resistance a POT delivers to a circuit.
- This in turn can control (vary) the amount of "flow" of electrons, and thus the voltage on the circuit.
- <https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/circuit-2-potentiometer>

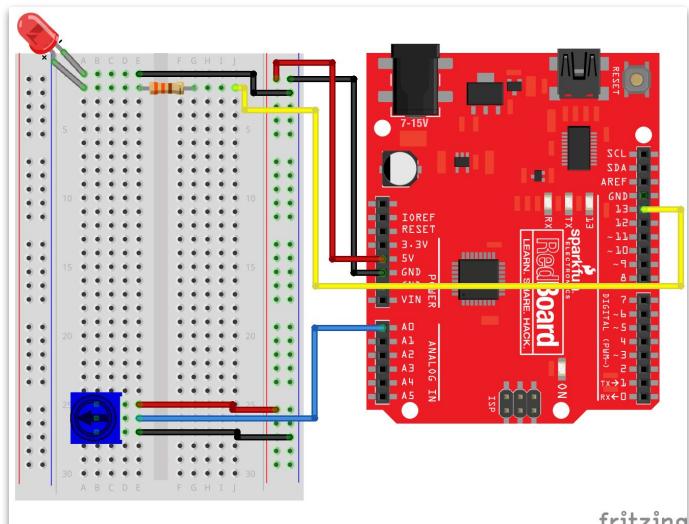
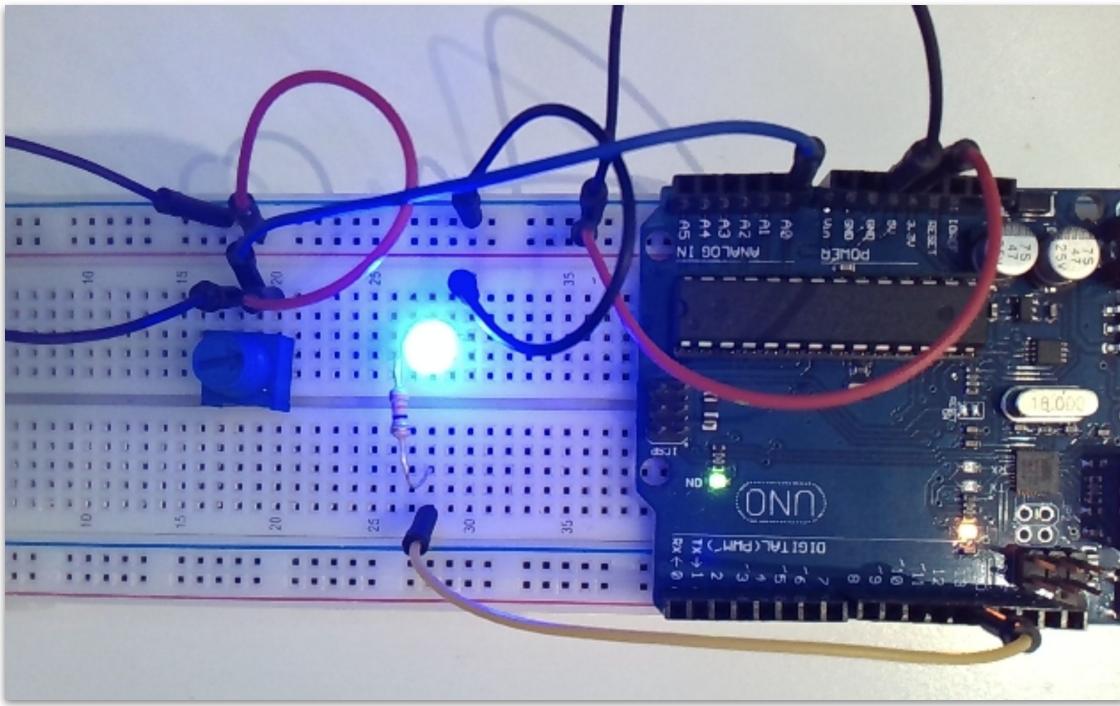


<https://www.sparkfun.com/products/9806>



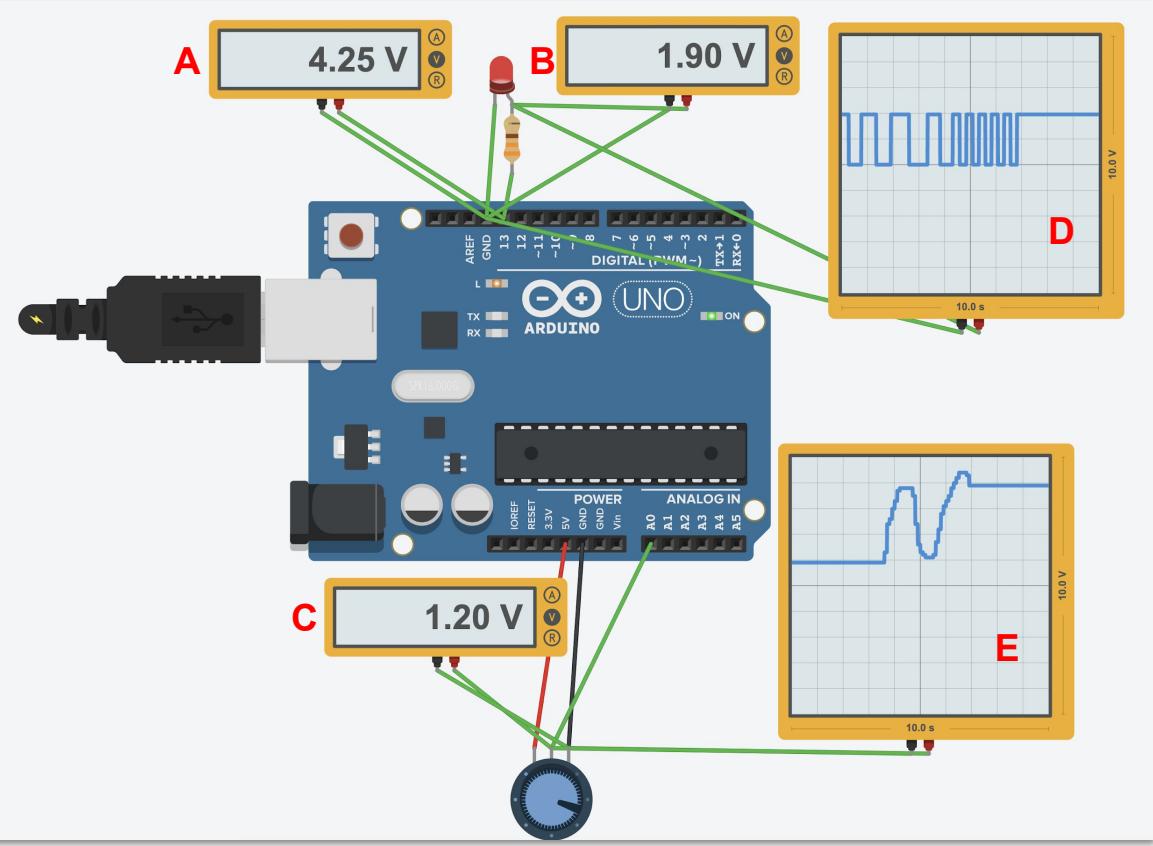
<https://www.sparkfun.com/products/14624>

Lets build it!



fritzing

```
1 /*
2 SparkFun Tinker Kit
3 Circuit 2: Potentiometer
4
5 Changes how fast an LED connected to pin 13 blinks, based on a potentiometer connected to pin A0
6
7 This sketch was written by SparkFun Electronics, with lots of help from the Arduino community.
8 This code is completely free for any use.
9
10 View circuit diagram and instructions at: https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/
11 Download code at: https://github.com/sparkfun/SparkFun\_Tinker\_Kit\_Code
12 */
13
14 int potPosition;      //this variable will hold a value based on the position of the potentiometer
15
16 void setup()
17 {
18     Serial.begin(9600);      //start a serial connection with the computer
19
20     pinMode(13, OUTPUT);    //set pin 13 as an output that can be set to HIGH or LOW
21 }
22
23 void loop()
24 {
25     //read the position of the pot
26     potPosition = analogRead(A0);    //set potPosition to a number between 0 and 1023 based on how far the knob is turned
27     Serial.println(potPosition);    //print the value of potPosition in the serial monitor on the computer
28
29     //change the LED blink speed based on the trimpot value
30     digitalWrite(13, HIGH);        // Turn on the LED
31     delay(potPosition);          // delay for as many miliseconds as potPosition (0-1023)
32
33     digitalWrite(13, LOW);        // Turn off the LED
34     delay(potPosition);          // delay for as many miliseconds as potPosition (0-1023)
35 }
```



POT Demo with Multimeter to measure point in time voltage and oscilloscope to measure voltage over time.

A = Voltage being sent to the LED

B = Voltage being sent to the LED after resistor. Higher the resistor, the lower the voltage

C = Voltage being sent out from the potentiometer. Variable based on knob

D = Voltage over time to LED

E = Voltage over time from POT



Analog vs Digital in Electronics

Analog signals have an infinite set of possible values.

The world around us is filled with examples such as temperature, speed, sound, light/color, TV signals until 2009, etc.

The values can range across a theoretical infinite range.

Temperature can be 98.1, 98.11, 98.112, 98.1134, 99.4.

Digital signals are "discrete", meaning they have a fixed set of values (or states).

Most often, in the field of electronics and computers, the values are 0 and 1 (binary).

These values are represented as voltage. For example 0V = 0 and 5V = 1.

Usually this is a little more flexible. Ex: 0V-1.5V = 0 and anything above 1.5V is 1

Real world examples include on/off switches, locks, stop lights

Analog vs Digital in Arduino

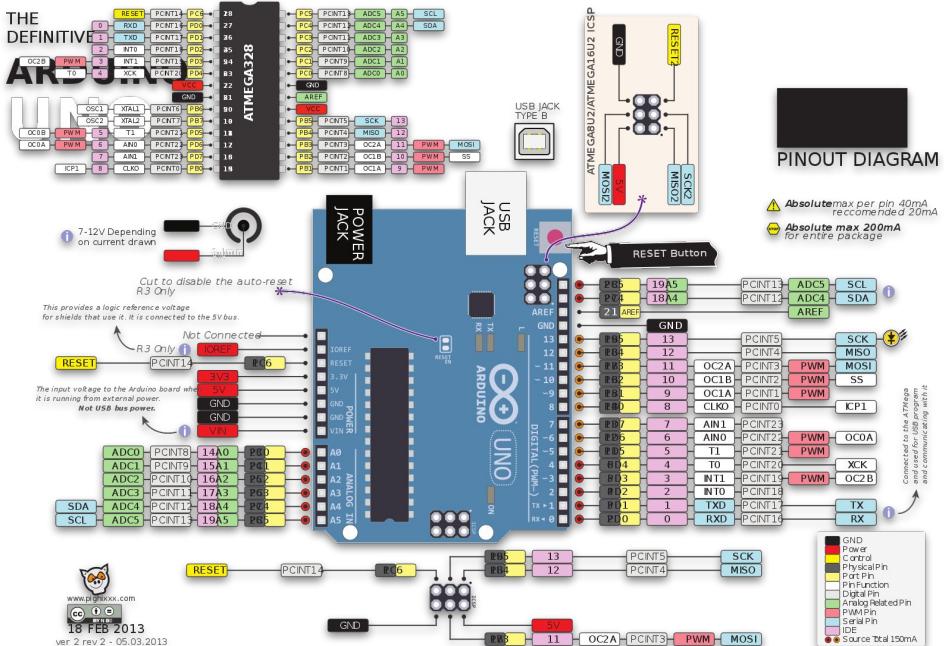
The Arduino has pins for sending/receiving digital signals in D0-D13 and for receiving (no sending) analog signals in pins A0-A5.

All voltages on the arduino are 0V - 5V.

Analog values are converted to digital values using an Analog to Digital Converter (ADC)

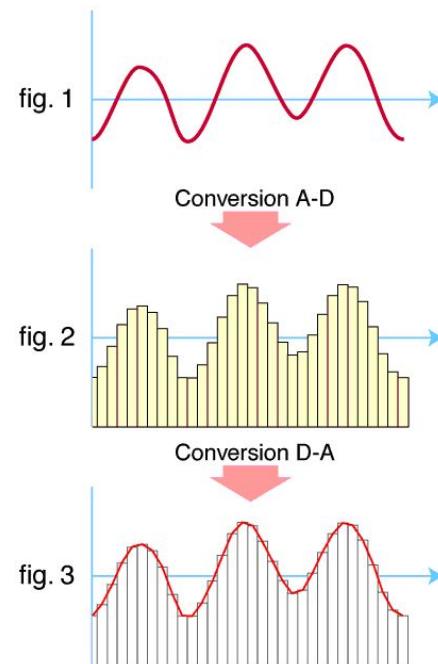
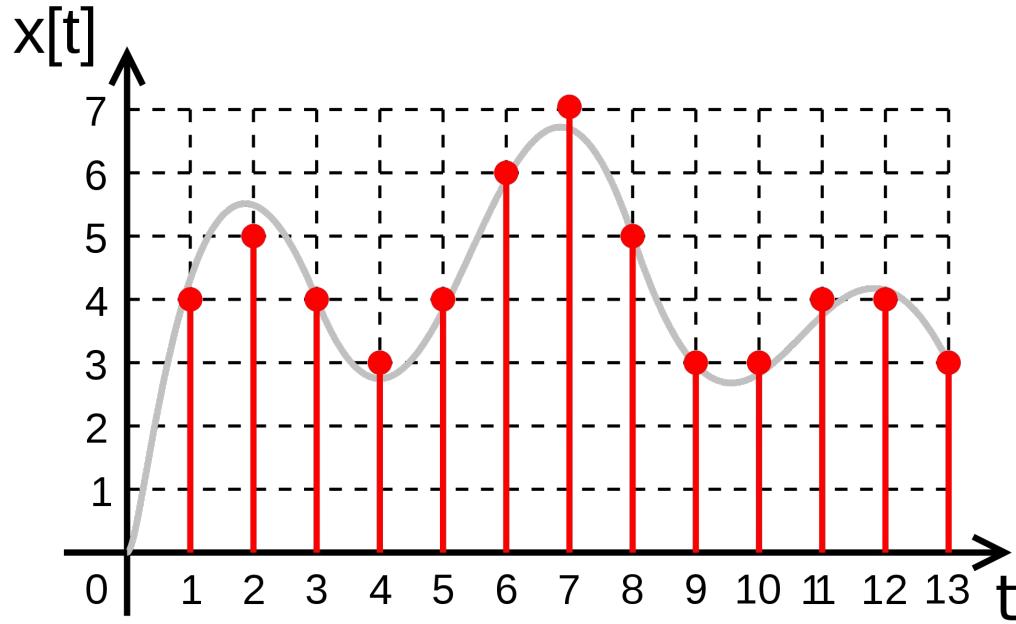
The Arduino ADC has 10 bits. $2^{10} = 1024$ so the arduino can map values of 0V to 5V to a range of 0 to 1023

Digital values are read as simply LOW for 0 and HIGH for 1



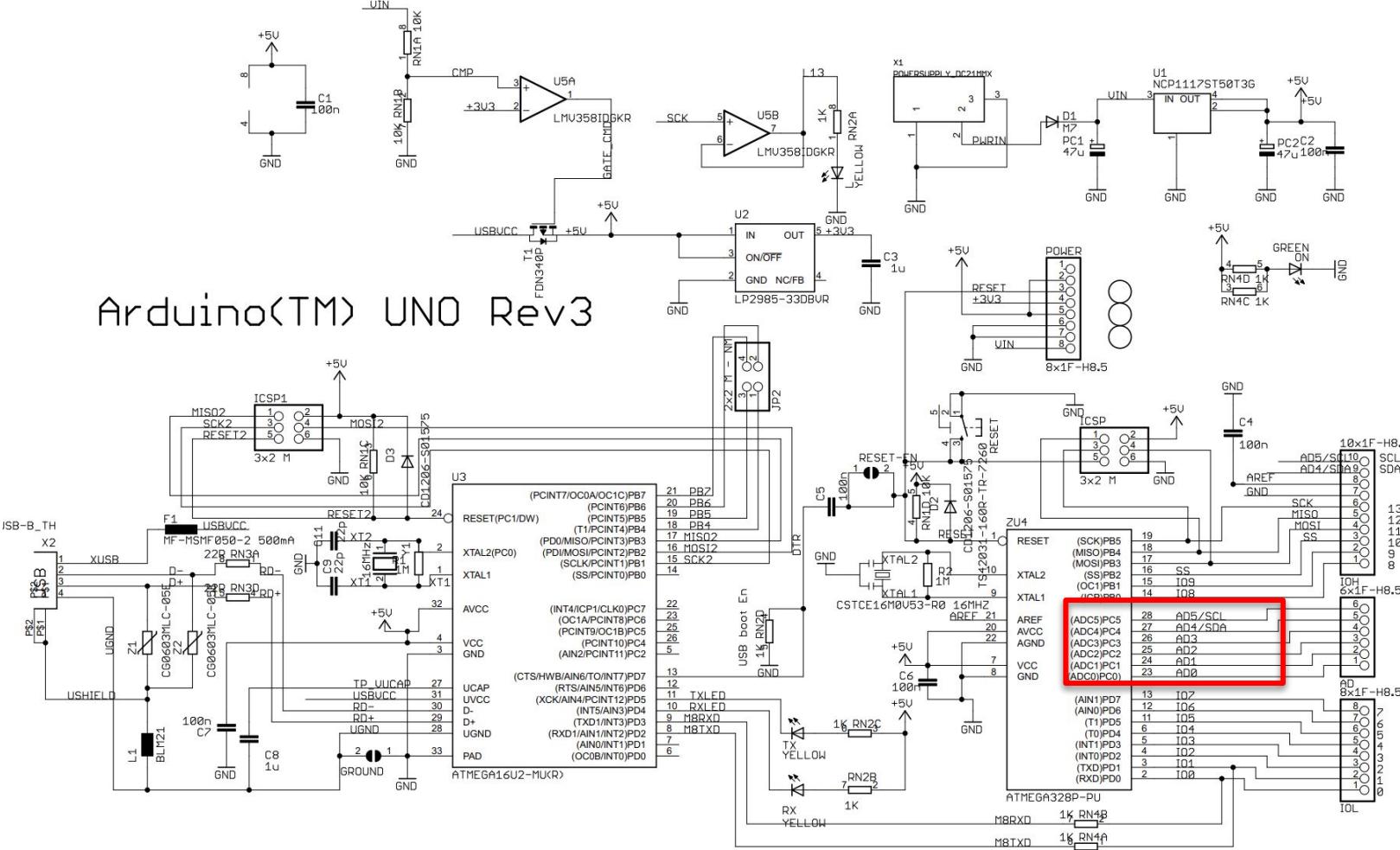
https://commons.wikimedia.org/wiki/File:Pinout_of_ARDUINO_Board_and_ATMega328PU.svg

ADC or how to convert analog to digital (sampling)



https://en.wikipedia.org/wiki/Analog-to-digital_converter

Arduino(TM) UNO Rev3



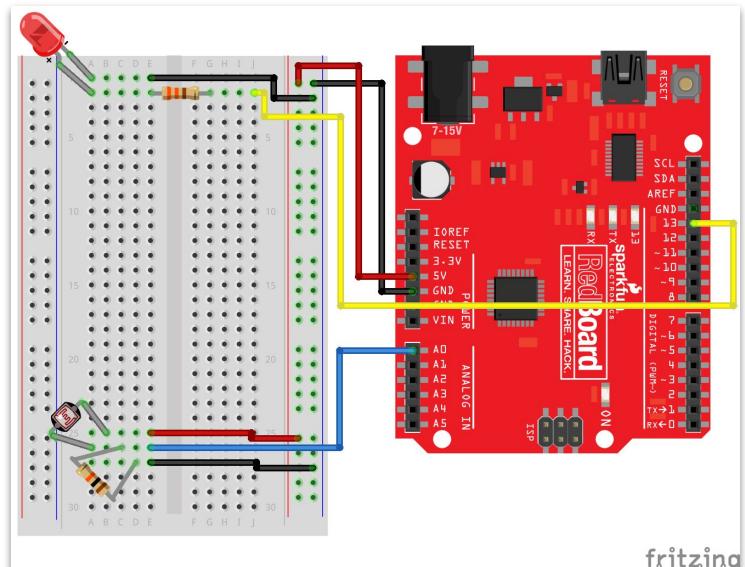
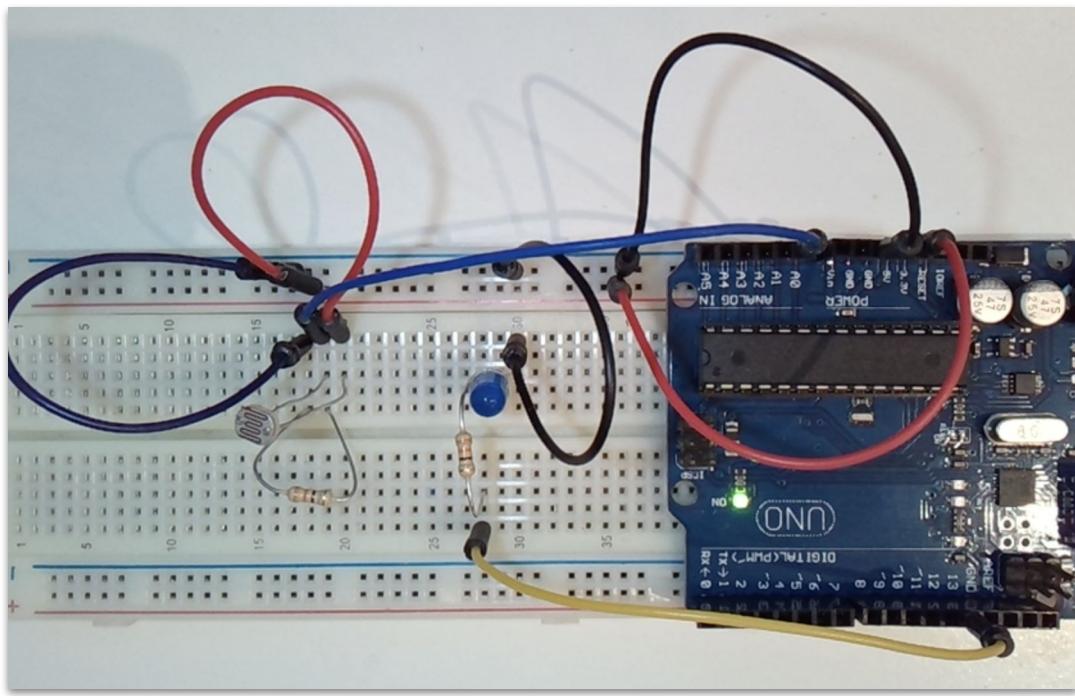
2 7
RN3B 22R
3 6
RN3C 22R

Circuit 3 - Photoresistor

- A photo resistor is also a "variable" resistor.
- The photoresistor uses a light sensor to control the amount of resistance.
- <https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/circuit-3-photoresistor>



Lets build it!



fritzing

```
1 /*
2 SparkFun Tinker Kit
3 Circuit 3: Photoresistor
4
5 Use a photoresistor to monitor how bright a room is, and turn an LED on when it gets dark.
6
7 This sketch was written by SparkFun Electronics, with lots of help from the Arduino community.
8 This code is completely free for any use.
9
10 View circuit diagram and instructions at: https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/
11 Download drawings and code at: https://github.com/sparkfun/SparkFun\_Tinker\_Kit\_Code
12 */
13
14 int photoresistor = 0;           //this variable will hold a value based on the position of the knob
15 int threshold = 150;            //if the photoresistor reading is below this value the light will turn on
16
17 void setup()
18 {
19   Serial.begin(9600);           //start a serial connection with the computer
20
21   pinMode(13, OUTPUT);          //set pin 13 as an output that can be set to HIGH or LOW
22 }
23
24 void loop()
25 {
26   //read the position of the knob
27   photoresistor = analogRead(A0); //set photoresistor to a number between 0 and 1023 based on how far the knob is turned
28   Serial.println(photoresistor);  //print the value of photoresistor in the serial monitor on the computer
29
30   //if the photoresistor value is below the threshold turn the light on, otherwise turn it off
31   if (photoresistor < threshold){
32     digitalWrite(13, HIGH);      // Turn on the LED
33   } else{
34     digitalWrite(13, LOW);       // Turn off the LED
35   }
36
37   delay(100);                  //short delay to make the printout easier to read
38 }
```

Module 3 - Surprise !!!! You have homework !!!

1. Build circuits 2 and 3
2. Study the Arduino Pinout Guide
 - a. <https://www.circuito.io/blog/arduino-uno-pinout/>
3. Read the Sparkfun engineering essentials guide
 - a. https://www.sparkfun.com/engineering_essentials
4. Extra Credit
 - a. Use the potentiometer to connect to also connect to a digital pin instead to see what analog value (0-1023) makes the digital value go from 0 to 1. Extrapolate what the analog value would be in terms of voltage.

Q&A

Github Page : <https://github.com/faumahesh/arduino>

Contact information : mahesh@fau.edu

Introduction to Arduino - Module 4

- Questions from previous session?
- Circuit 4 - RGB Night-light
- Circuit 5 - Buzzer
- Q&A

Questions ?

RGB LED

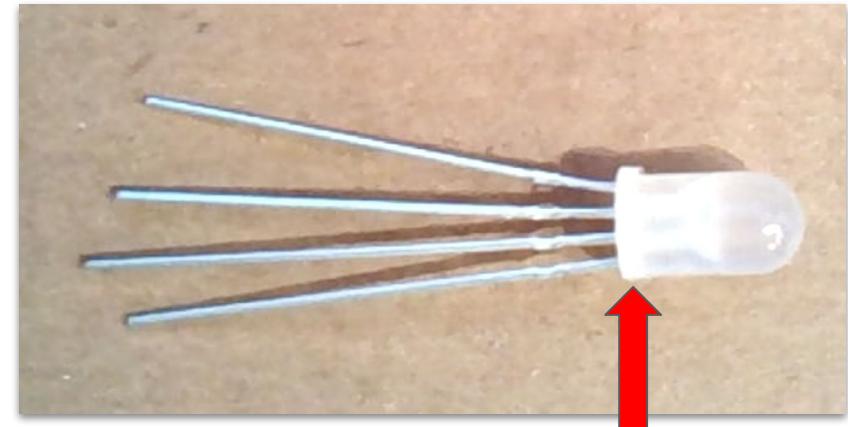
Similar to a regular LED but instead has the three primary colors used in electronic devices (red, green, blue).

Each "color" has a separate wire and they all share a common ground (GND)

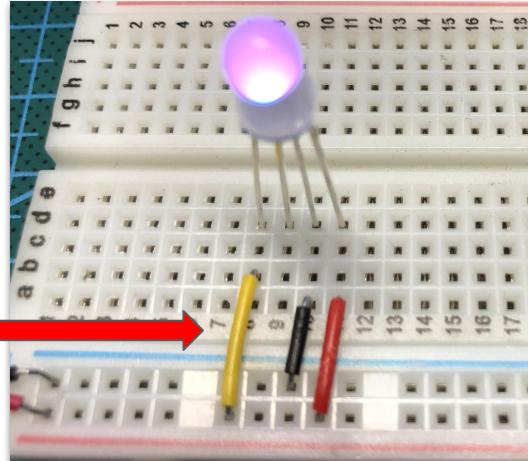
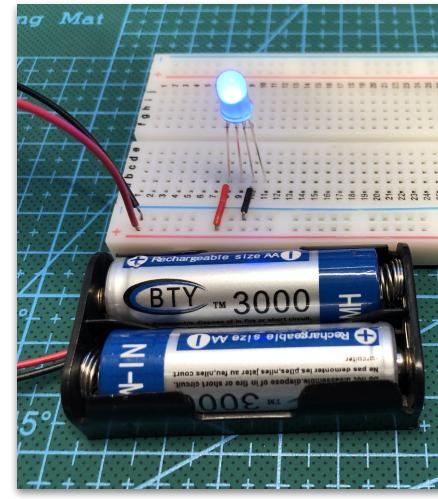
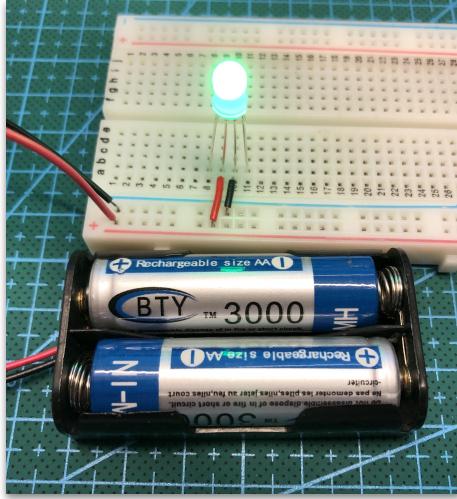
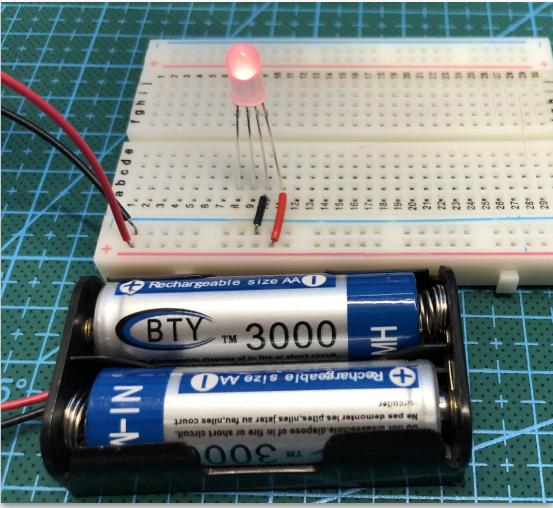
Longest wire here is GND (opposite of regular LED), followed by green, blue and red.

If you cut the wires, use the flat edge of the LED to denote the "red" side.

RYB and CMY are also used in color theory (google it).



Red side (flat)



Pre-cut wires! Super
clean for prototyping!

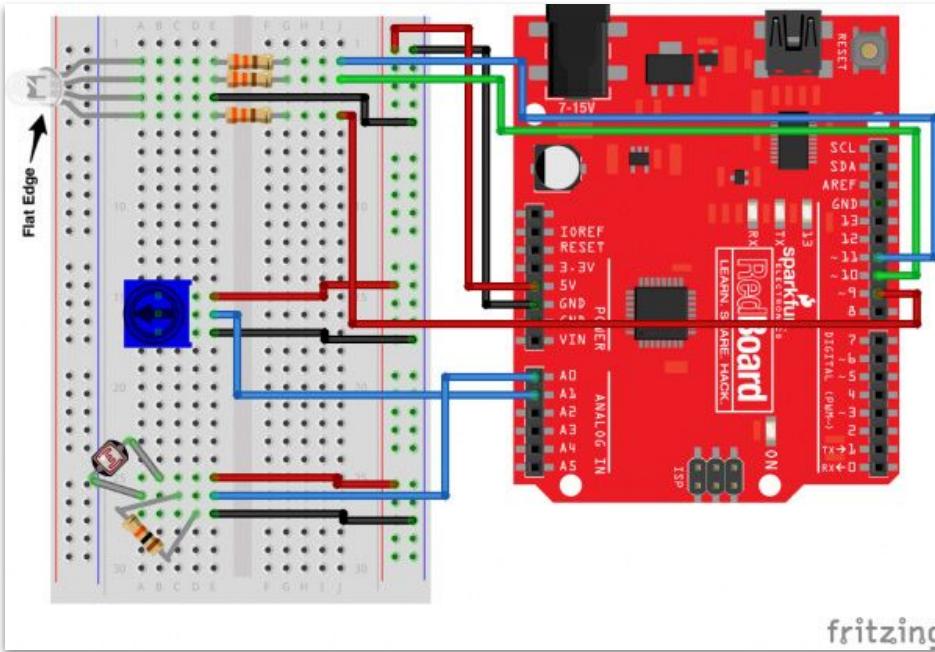
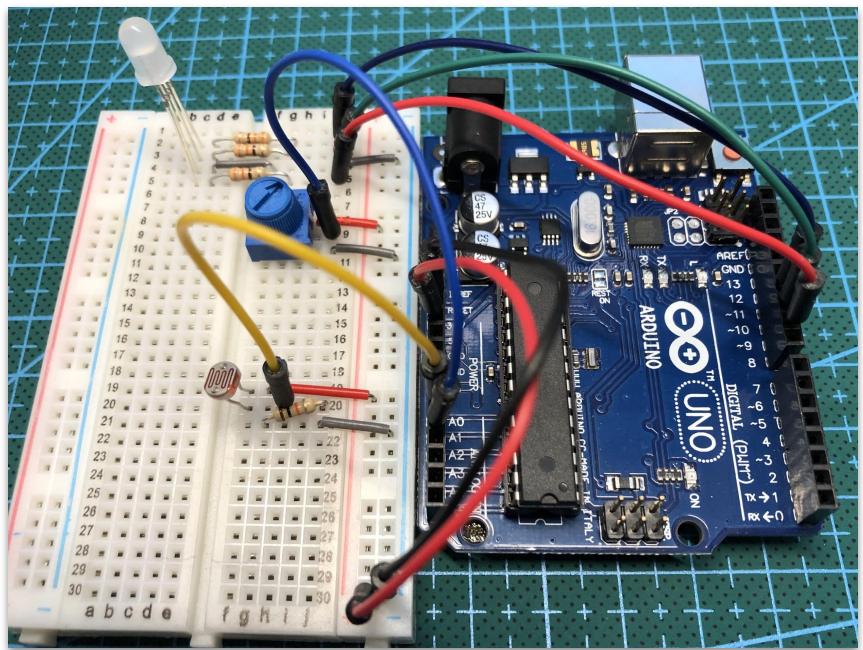
Red and Blue make Purple!

Circuit 4 - RGB Night-light

Program Overview

1. Store the light level from pin A0 in the variable `photoresistor` .
2. Store the potentiometer value from pin A1 in the variable `potentiometer` .
3. If the light level variable is above the `threshold` , call the function that turns the RGB LED off.
4. If the light level variable is below the `threshold` , call one of the color functions to turn the RGB LED on.
5. If `potentiometer` is between 0 and 150, turn the RGB LED on red.
6. If `potentiometer` is between 151 and 300, turn the RGB LED on orange.
7. If `potentiometer` is between 301 and 450, turn the RGB LED on yellow.
8. If `potentiometer` is between 451 and 600, turn the RGB LED on green.
9. If `potentiometer` is between 601 and 750, turn the RGB LED on cyan.
10. If `potentiometer` is between 751 and 900, turn the RGB LED on blue.
11. If `potentiometer` is greater than 900, turn the RGB LED on magenta.

Lets build it!



fritzing

Buzzer

The buzzer uses a small magnetic coil to vibrate a metal disc inside a plastic housing. By pulsing electricity through the coil at different rates, different frequencies (pitches) of sound can be produced. Attaching a potentiometer to the output allows you to limit the amount of current moving through the buzzer and lower its volume.

We will be using the [tone\(\)](#) library function that is included with the arduino to generate the frequency needed (as electrical pulses) to create the sound in the buzzer.



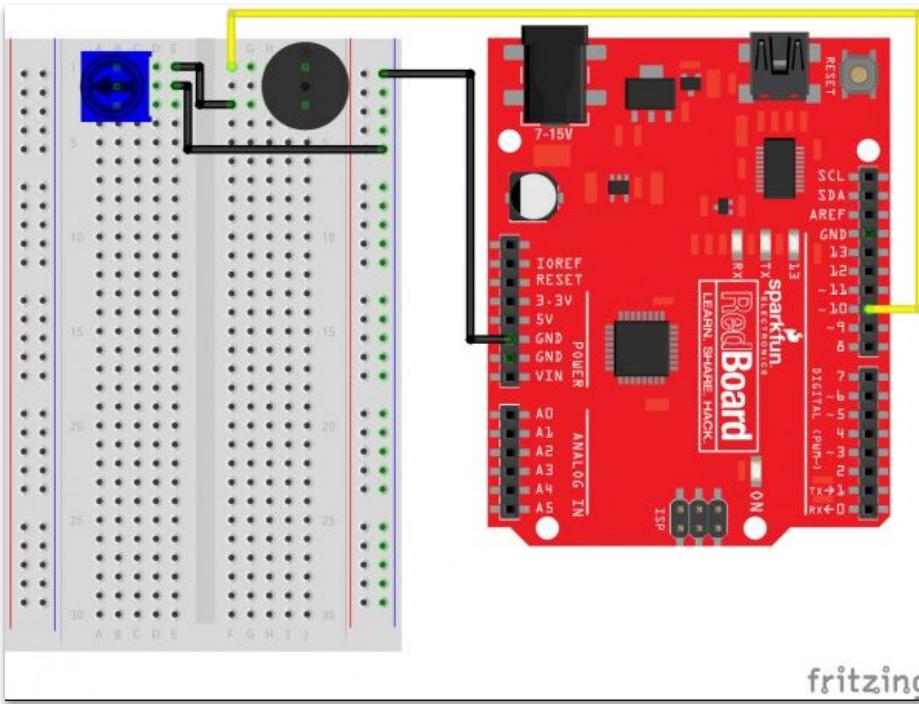
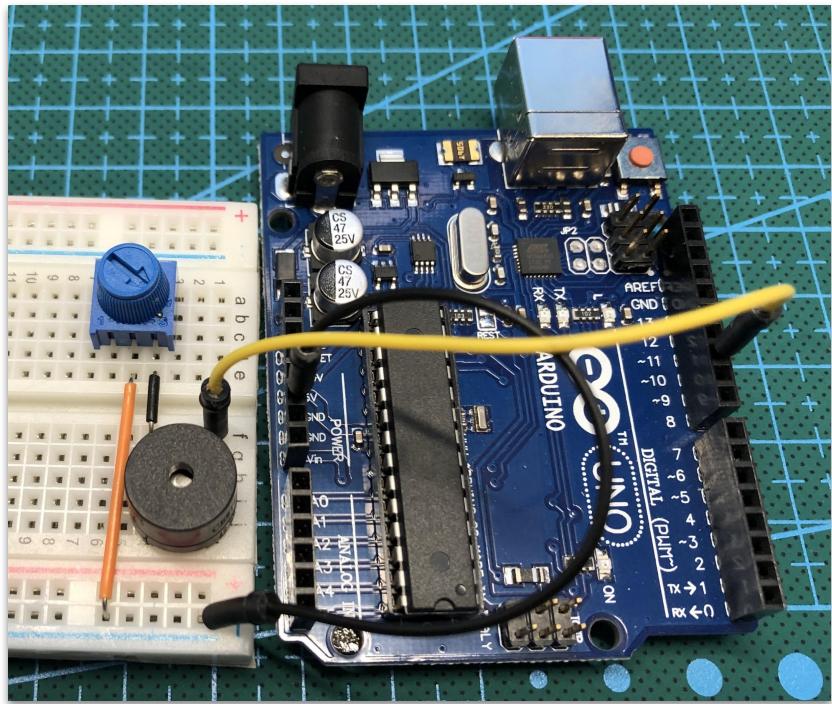
Circuit 5 - Buzzer

Program Overview

Inside the main loop:

1. Play the first note for x number of beats using the play function.
 - a. (Inside the play function:) Take the note passed to the play function and compare it to each letter in the notes array. When you find a note that matches, remember the index position of that note (e.g., 6th entry in the notes array).
 - b. Get a frequency from the frequency array that has the same index as the note that matched (e.g., the 6th frequency).
 - c. Play that frequency for the number of beats passed to the play function.
2. Play the second note using the play function
...and so on.

Lets build it!



fritzing

Q&A

Github Page : <https://github.com/faumahesh/arduino>

Contact information : mahesh@fau.edu

Introduction to Arduino - Module 5

- Questions from previous session?
- Circuit 6 - Digital Trumpet
- Oscilloscopes
- Circuit 7 - Simon Says
- Q&A

Questions ?

Circuit 6 : Digital Trumpet



Push Buttons are "momentary switches" are used for intermittent access (reset switches, keypads, etc). See the [DSO138](#) oscilloscope later in this section for a use-case.

We use the [tone\(\)](#) function to generate a signal at a specific frequency.

Reference > Language > Functions > Advanced io > Tone

tone()

[Advanced I/O]

Description

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to `noTone()`. The pin can be connected to a piezo buzzer or other speaker to play tones.

Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to `tone()` will have no effect. If the tone is playing on the same pin, the call will set its frequency.

Use of the `tone()` function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

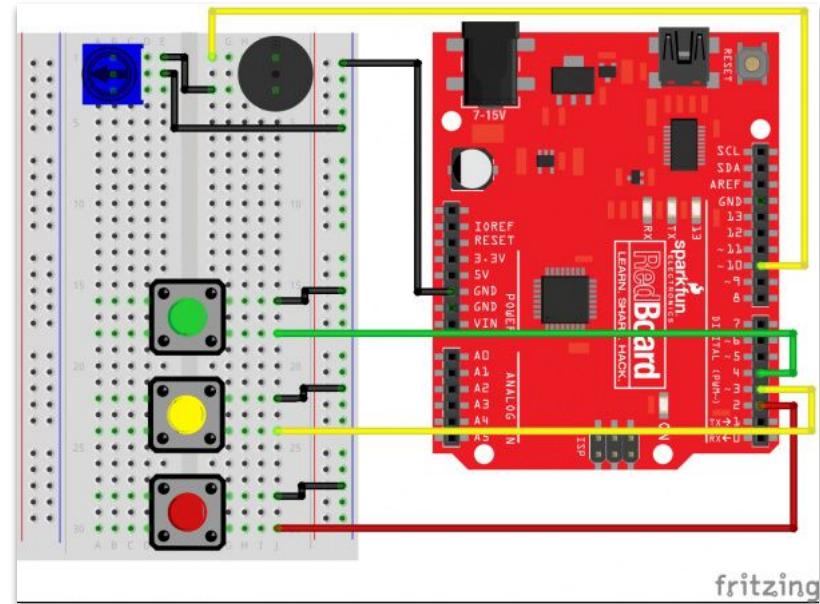
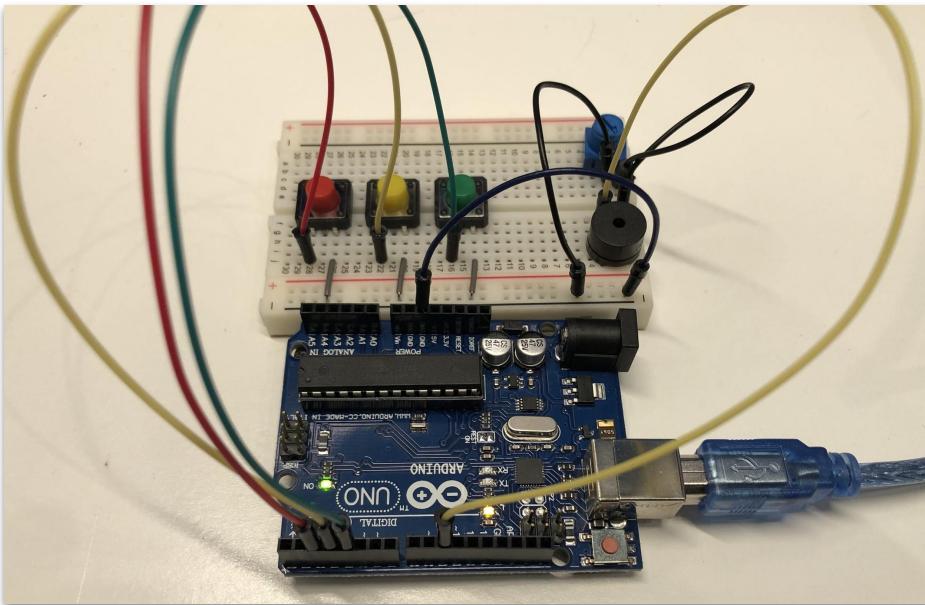
It is not possible to generate tones lower than 31Hz. For technical details, see Brett Hagman's notes.

INPUT_PULLUP

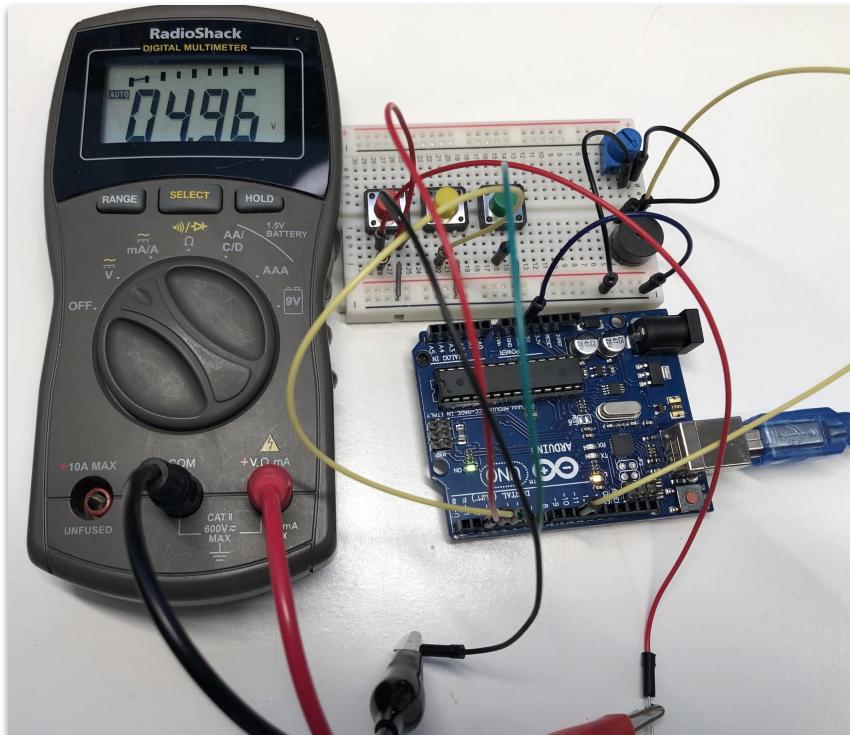
In order to read the value of the push button, we will use a "digital" pin. Digital pins will have either a ON or OFF (1 or 0, HIGH or LOW) value. However to be electrically correct, they need to be wired correctly. This wiring is done with an internal "pullup" resistor. For simplicity, we will assume this is part of what makes arduino's easy to use. However it is an important topic and is explained well here:

<https://www.baldengineer.com/arduino-internal-pull-up-resistor-tutorial.html>

Lets Build It!



INPUT_PULLUP In Action



When no button is pressed, the multimeter reads ~5 volts. When the button is pressed, the value drops to 0 volts.

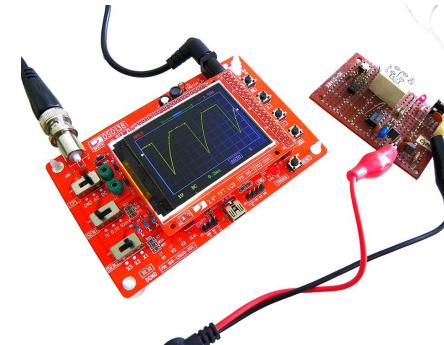
Oscilloscopes



Classic Analog Oscilloscope



Modern Digital Oscilloscope



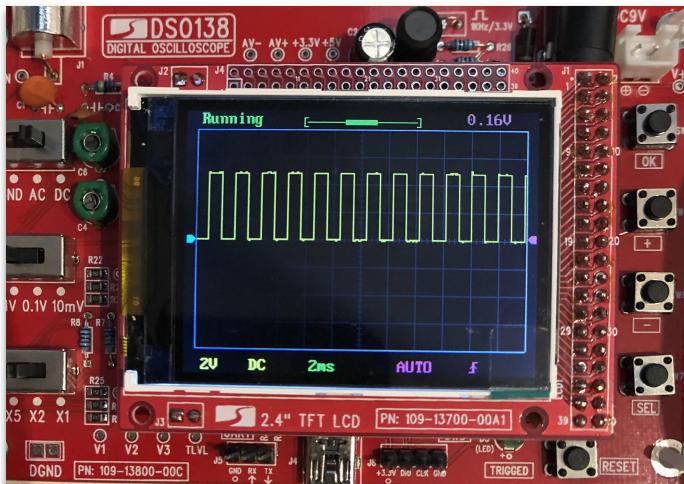
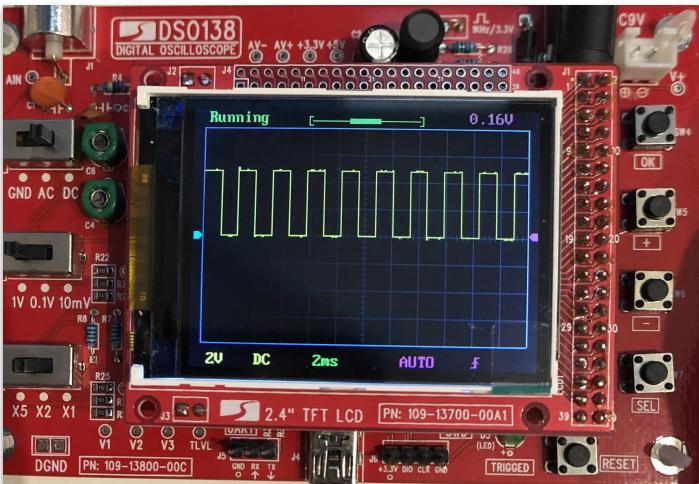
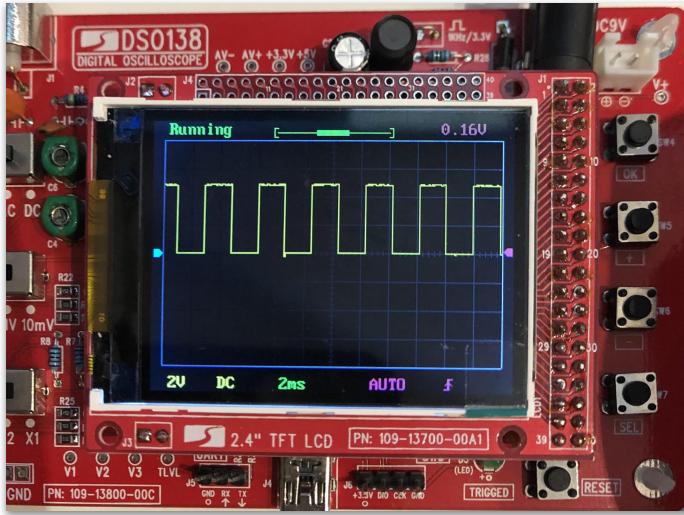
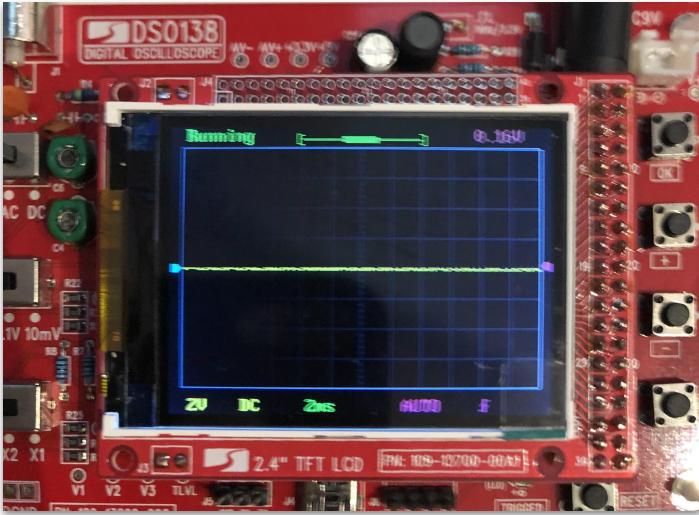
Low-End DIY Oscilloscope

Oscilloscopes are measurement devices that display voltage over time. They can be used to look at signal waveforms to analyze properties such as amplitude, frequency, rise time, time interval, distortion, and others.

Visualizing Frequencies

Oscilloscope
measuring
frequencies at 0 Hz,
262 Hz, 392 Hz and
523 Hz from output
of Arduino

Can you find
the
pushbuttons?



Circuit 7 - Simon Says

- Our first "game" using an arduino
- Mimic the [classic memory game from Milton Bradley](#) (now Hasbro)
- Use arrays, loops, time delays, random numbers, functions, buzzers and LEDs together
- [millis\(\)](#), [random\(\)](#), [tone\(\)](#),

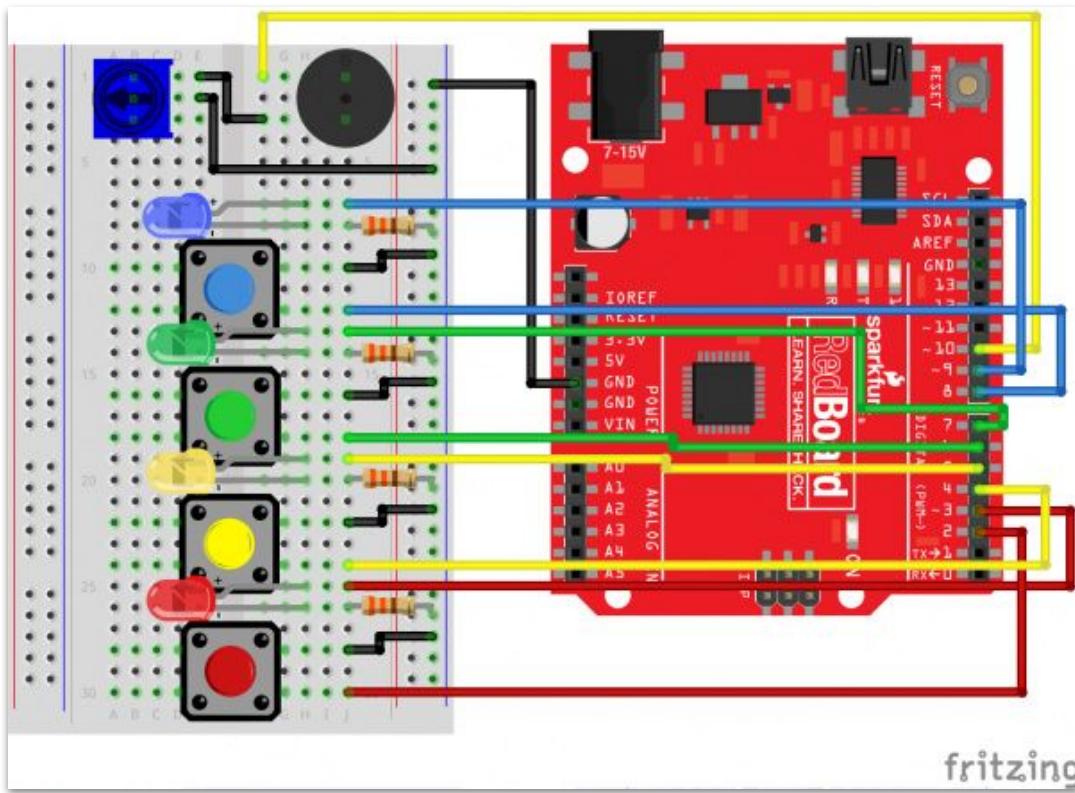
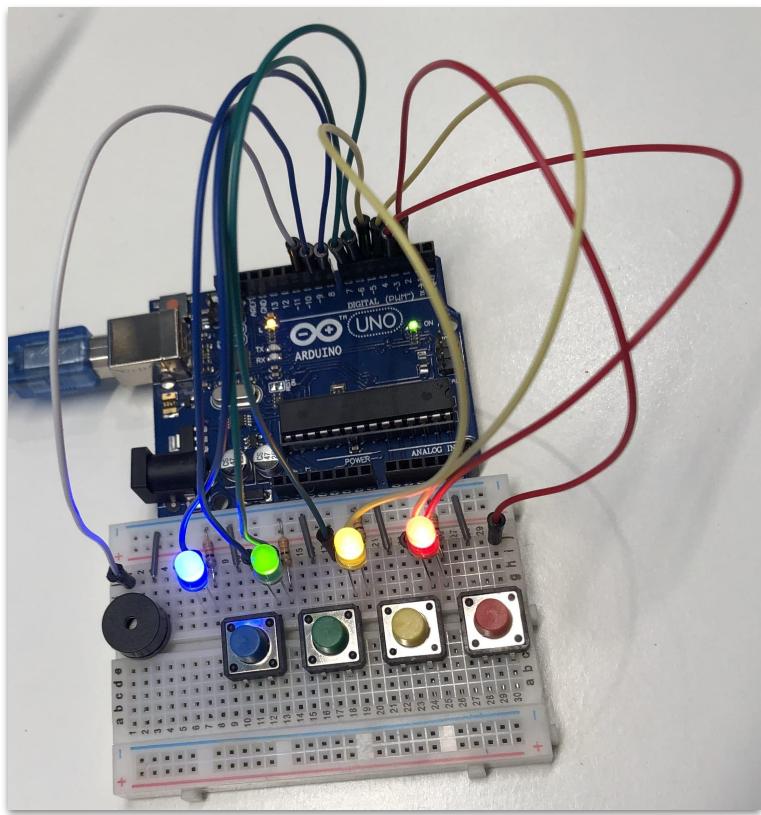


Circuit 7 - Simon Says

Program Overview

1. Check if a new game is starting. If it is, play the start sequence. Reset the counter that keeps track of rounds, and randomly generate a sequence of numbers from 0 to 3 that control which LEDs the user will have to remember.
2. The game works in rounds that progress from 0 to 10. Each round the game will flash LEDs in a pattern, then the player has to recreate the pattern by pressing the button(s) that match the LED(s). In the first round, one LED will flash, and the player will have to press one button. In the eighth round, eight LEDs will flash, and the player will have to press eight buttons.
3. Use a loop to flash LEDs from the sequence until you have flashed the number of LEDs that matches the round number (1 for round 1, 2 for round 2, etc).
4. Start a timer, and wait for the player to press a button. The player has 1.5 seconds to press the correct button.
 - a. If the time limit runs out before a button is pressed, the player loses.
 - b. If the player presses the wrong button, the player loses.
 - c. If the player presses the right button, move on to the next number in the sequence.
 - d. Repeat this process until the player has lost or correctly repeated the sequence for this round.
5. If the player repeats the entire sequence for that round. Increase the round number by one (this will add one extra item to the end of the pattern). Then go back to step 3.
6. Keep incrementing the round until the player loses or the player finishes 10 rounds. If the player finishes 10 rounds, play the winning sequence.

Lets Build It!



Q&A

Github Page : <https://github.com/faumahesh/arduino>

Contact information : mahesh@fau.edu