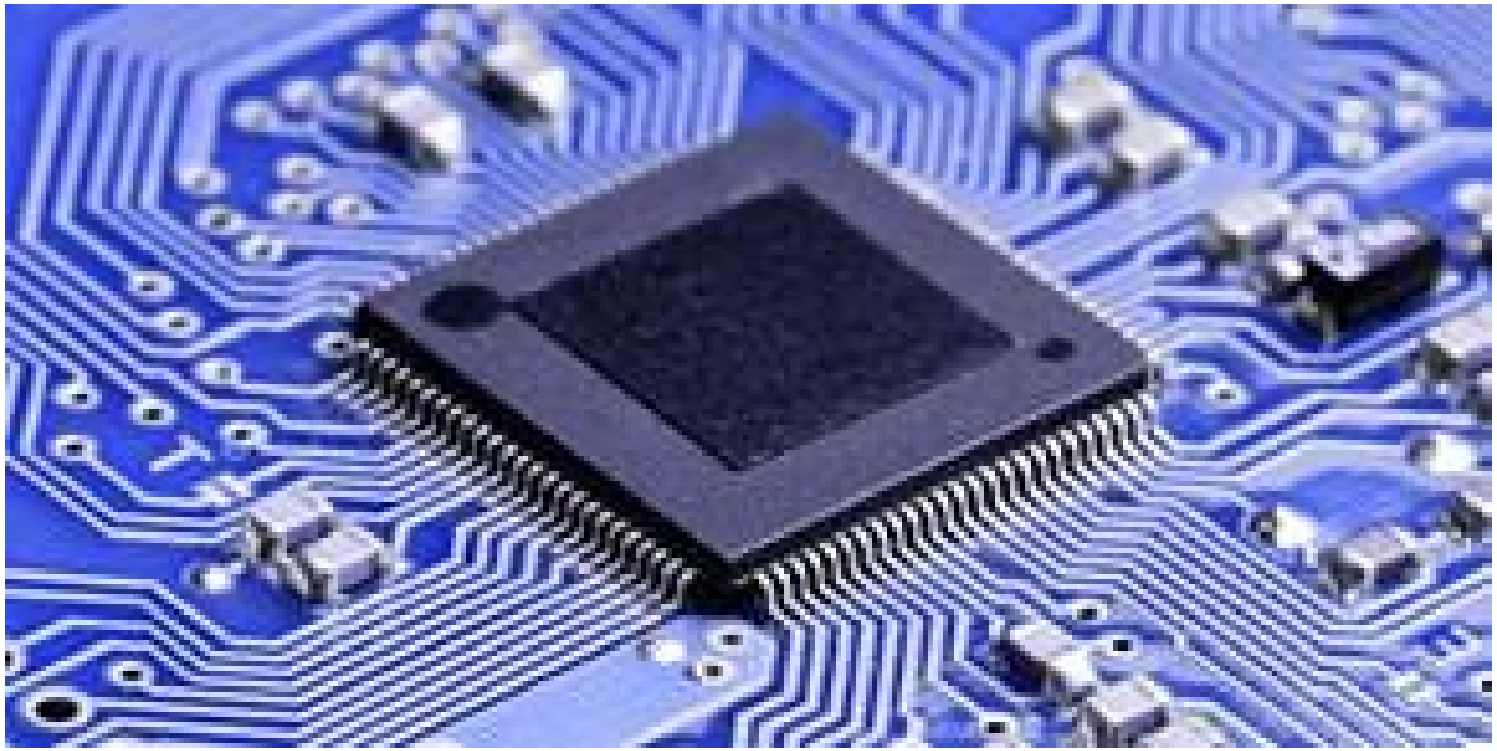
IoT Product Design and Rapid Prototyping

Brian Rashap, Ph.D.

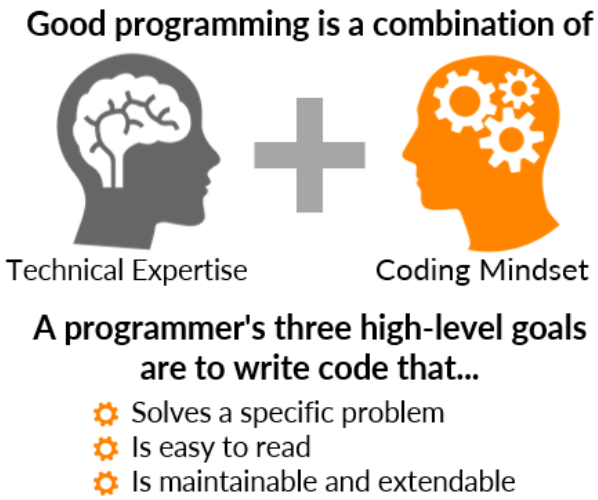
04-OCT-2021

Particle Argon

Our Second Microcontroller



Coding expectations for the rest of the course



1. Most important, be consistent.
2. Use proper indentation.
3. Brace placement: K&R or BSD
4. Do not check boolean for equality.
5. A variable's name (noun) should describe its contents.
6. A function's name (verb) should describe the set of actions it performs.
7. Reduce duplication; modularize.
8. So, what about capitalization?
   - variableNames
   - nameFunction
   - CONSTANTS
   - Classes and Enum

Previous Capstones

Previous Capstone Playlist

https://www.youtube.com/watch?v=s4TslpITeVw&list=PL0t2Pk5ETDgxfVptdyr6xbL6MW1-5CJey

Particle Argon

Main processor:

Nordic Semiconductor nRF52840 SoC

- ARM Cortex-M4F 32-bit processor @ 64MHz
- 1MB flash, 256KB RAM
- Bluetooth LE (BLE) central and peripheral support
- 20 mixed signal GPIO (6 x Analog, 8 x PWM), UART, I2C, SPI
- Supports DSP instructions, HW accelerated Floating Point Unit (FPU) calculations
- ARM TrustZone CryptoCell-310 Cryptographic and security module
- Up to +8 dBm TX power (down to -20 dBm in 4 dB steps)
- NFC-A radio

Argon Wi-Fi network coprocessor:

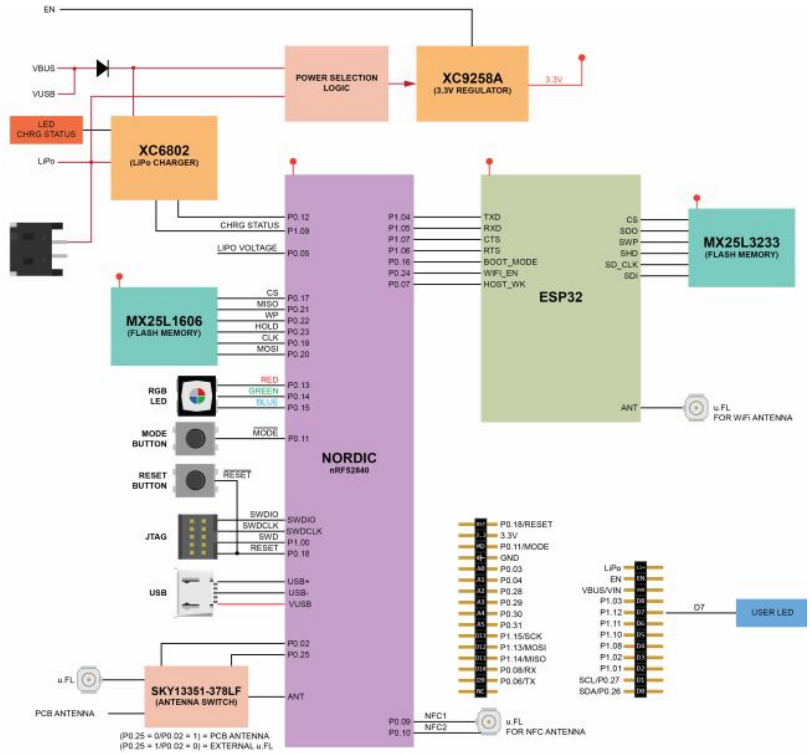
Espressif ESP32-D0WD 2.4 GHz Wi-Fi coprocessor

- On-board 4MB flash for the ESP32
- 802.11 b/g/n support
- 802.11 n (2.4 GHz), up to 150 Mbps

Argon general specifications:

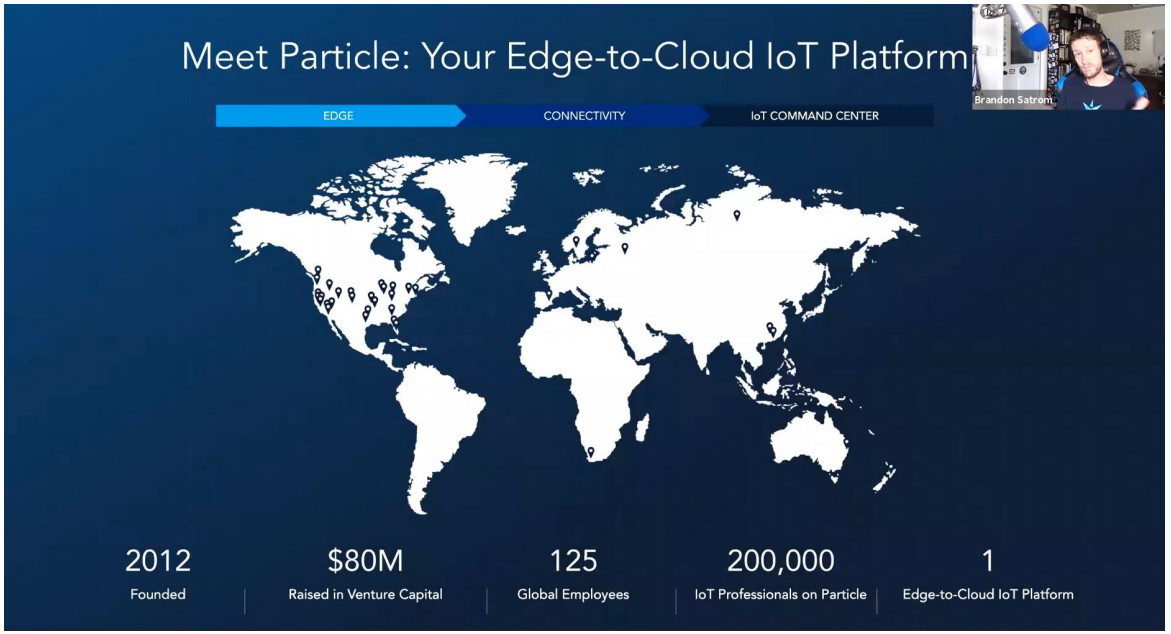
- On-board additional 4MB SPI flash
- Micro USB 2.0 full speed (12 Mbps)
- Integrated Li-Po charging and battery connector
- JTAG (SWD) Connector
- RGB status LED
- Reset and Mode buttons
- On-board 2.4GHz PCB antenna for Bluetooth (does not support Wi-Fi)
- Two U.FL connectors for external antennas (one for Bluetooth, another for Wi-Fi)
- Meets the Feather specification in dimensions and pinout
- FCC, CE and IC certified
- RoHS compliant (lead-free)

Particle Argon Block Diagram



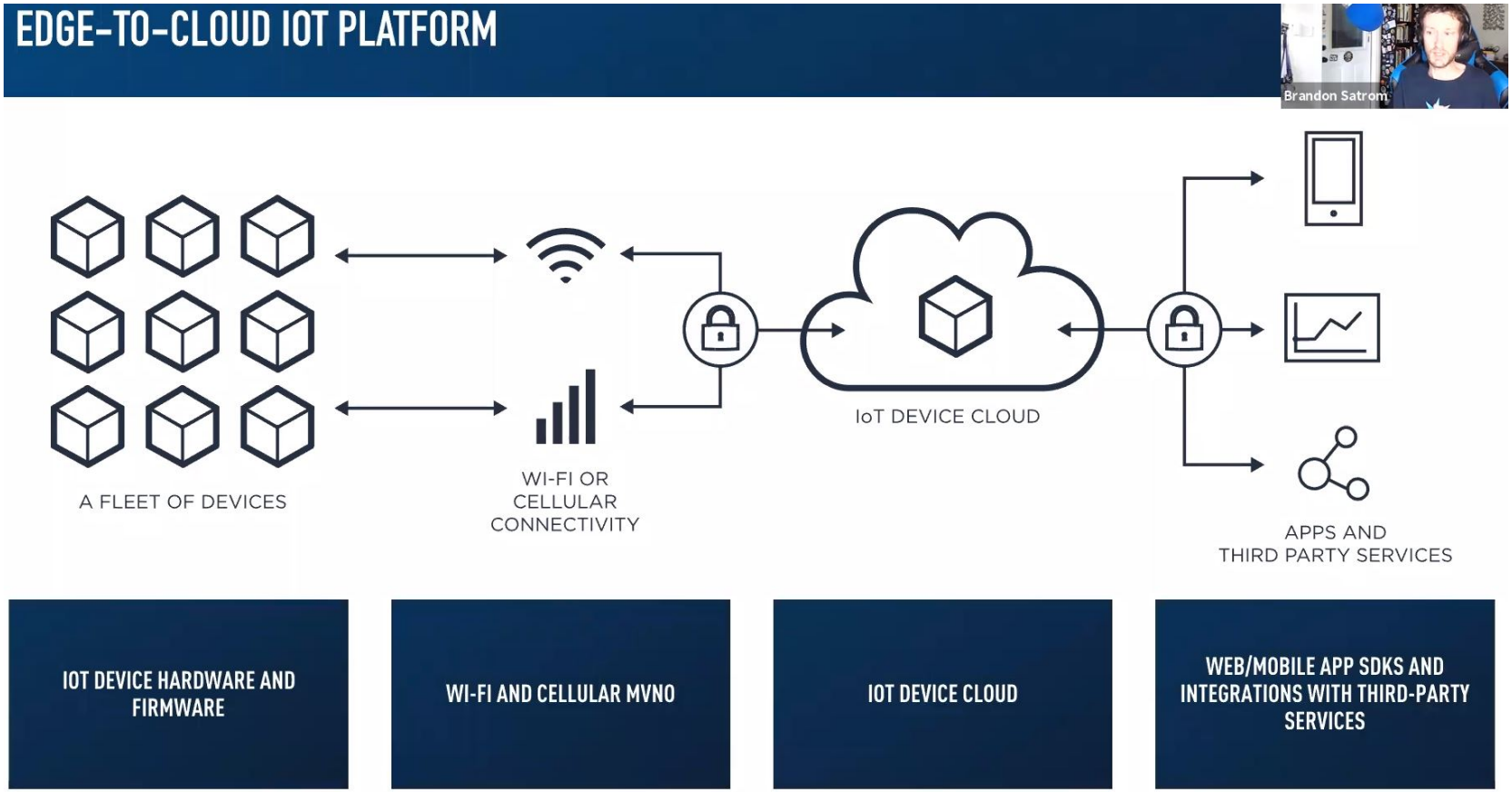
- nRF52840 (64 MHz ARM M4 Cortex with BLE and NFC)
- ESP32 (Wifi Coprocessor)
- 20 GPIO pins
- Additional 4MB SPI Flash
- Integrated LiPo battery charging
- Adafruit Feather pinout

Why Particle



- Global reach with over 200,000 IoT professionals.
- Edge-to-Cloud infrastructure.
- Prototyping to Production with same code.
- WI-FI, Bluetooth, and Cellular.
- Secure Device OS.
- Built-in cloud communication.
- Real-Time OS that works across all products.

Particle: Edge to Cloud

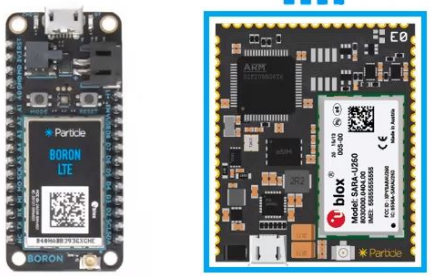


Particle: Prototyping to Production

HARDWARE AND CONNECTIVITY

1

HARDWARE FOR PROTOTYPING & PRODUCTION



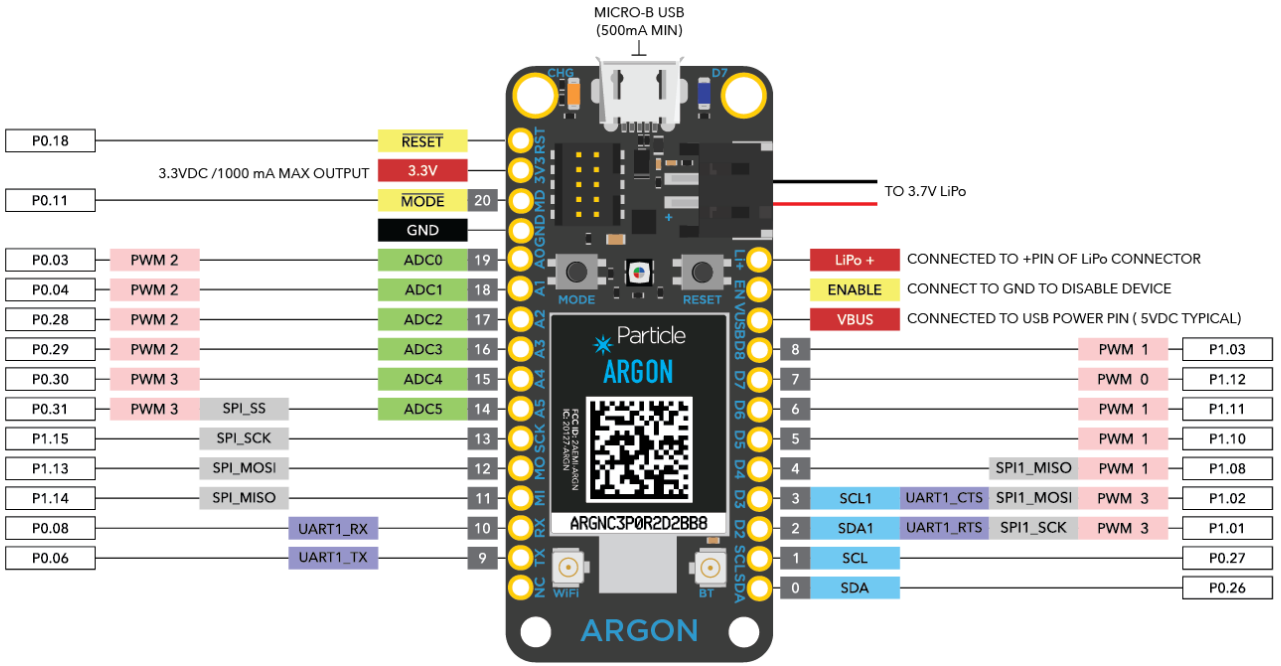
2

USE-CASE-SPECIFIC MODULES AND PRODUCTS

3

CELLULAR, BLE & WI-FI CONNECTIVITY

Particle Argon Pin Layout

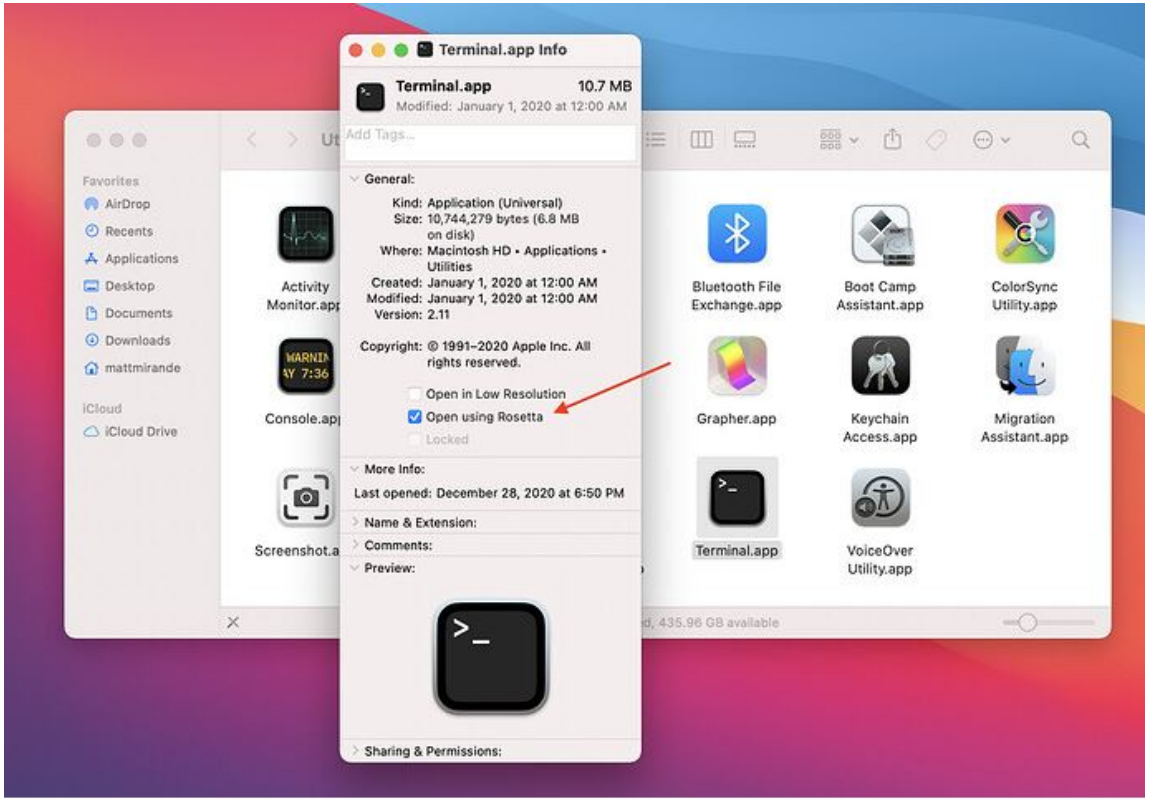


v1.0

Please Do Not Skip Ahead During Setup

If you have a Mac M1

Terminal needs to be “Open using Rosetta” option:



https://www.courier.com/blog/tips-and-tricks-to-setup-your-apple-m1-for-development/

Particle Software - ParticleCLI and Visual Studio Code

1. Create Particle login: `https://login.particle.io/signup`
2. Install Particle Command Line Interface. Download from `https://docs.particle.io/tutorials/developer-tools/cli/`
   - On Windows, run this by right-clicking on it and selecting "Run As Administrator."
   - Test that the Particle CLI installed correctly by going to PowerShell or Terminal and type particle.
3. Download Particle Workbench / Visual Studio Code `https://docs.particle.io/quickstart/workbench/`.
   - Select all default values during install.
   - **Do NOT install Azure IoT.**
   - After it is installed, when you launch it, it may ask you to Install Dependencies. If so, select yes.
4. On the Mac, install dfu-util per the instructions on the website.

Particle Setup

1. Attach the Wi-Fi antenna to your Argon. Use the correct connector. There are 3 U.FL connectors: WiFi, BT, and NFC.
2. Plug the Argon into a USB port. It should begin blinking blue.
3. Open PowerShell or Terminal.
4. Login into your Particle Account.

```
particle login
```

5. Ensure you have the latest Particle CLI.

```
particle update-cli
```

6. Put the Argon in DFU mode (blinking yellow) by holding down MODE. Tap RESET and continue to hold down MODE. The status LED will blink magenta (red and blue at the same time), then yellow. Release when it is blinking yellow.

Updating your Argon to latest Device OS

1. Update the device by running the following two commands. If the device goes out of blinking yellow after the first command, put it back into DFU mode. See Note [1].

```
particle update
particle flash --usb tinker
```

2. When the command reports Flash success!, reset the Argon. It should go back into listening mode (blinking dark blue).
3. Verify that the update worked by running the following command:

```
particle serial identify

Your device id is e00fce681fffffffffc08949b
Your system firmware version is 1.5.2
```

[1] paticle flash –usb tinker can be used for device troubleshooting

Setting Up WiFi

- Set your Argon into Listening Mode by holding the MODE button for three seconds, until the RGB LED begins blinking blue.
- Execute the command: particle serial wifi

```
brian:~$ particle serial wifi
? Should I scan for nearby Wi-Fi networks? No
? SSID DDCIOT
? Security Type WPA2
? Cipher Type AES+TKIP
? Wi-Fi Password ddcIOT2020
Done! Your device should now restart.
```

After setting, your Argon should go through the normal sequence of blinking green, blinking cyan (light blue), fast blinking cyan, and breathing cyan.

Claim Your Device

1. Claim the device to your account. This can only be done if it's breathing cyan. Replace e00fce681ffffffffc08949b with the device ID you got earlier from particle serial identify. Then, rename it to the name of your choice.

```
particle device add e00fce681ffffffffc08949b
particle device rename e00fce681ffffffffc08949b
   myArgon
```

2. Ensure that your setup flag is marked as done.

```
particle usb setup-done
```

3. You have successfully set up your Argon!

Useful Particle CLI Commands

1. Enter DFU mode from the CLI.

```
particle usb dfu
```

2. If the Argon won't enter DFU mode or is otherwise acting strangely, restore the base firmware.

```
particle flash --usb tinker
```

3. Get a list of your Particle devices and their connection status.

```
particle list
```

4. Search for available libraries.

```
particle library search <search term>
```

5. Link for the Particle Setup procedures: Particle Setup via CLI

Particle Troubleshooting

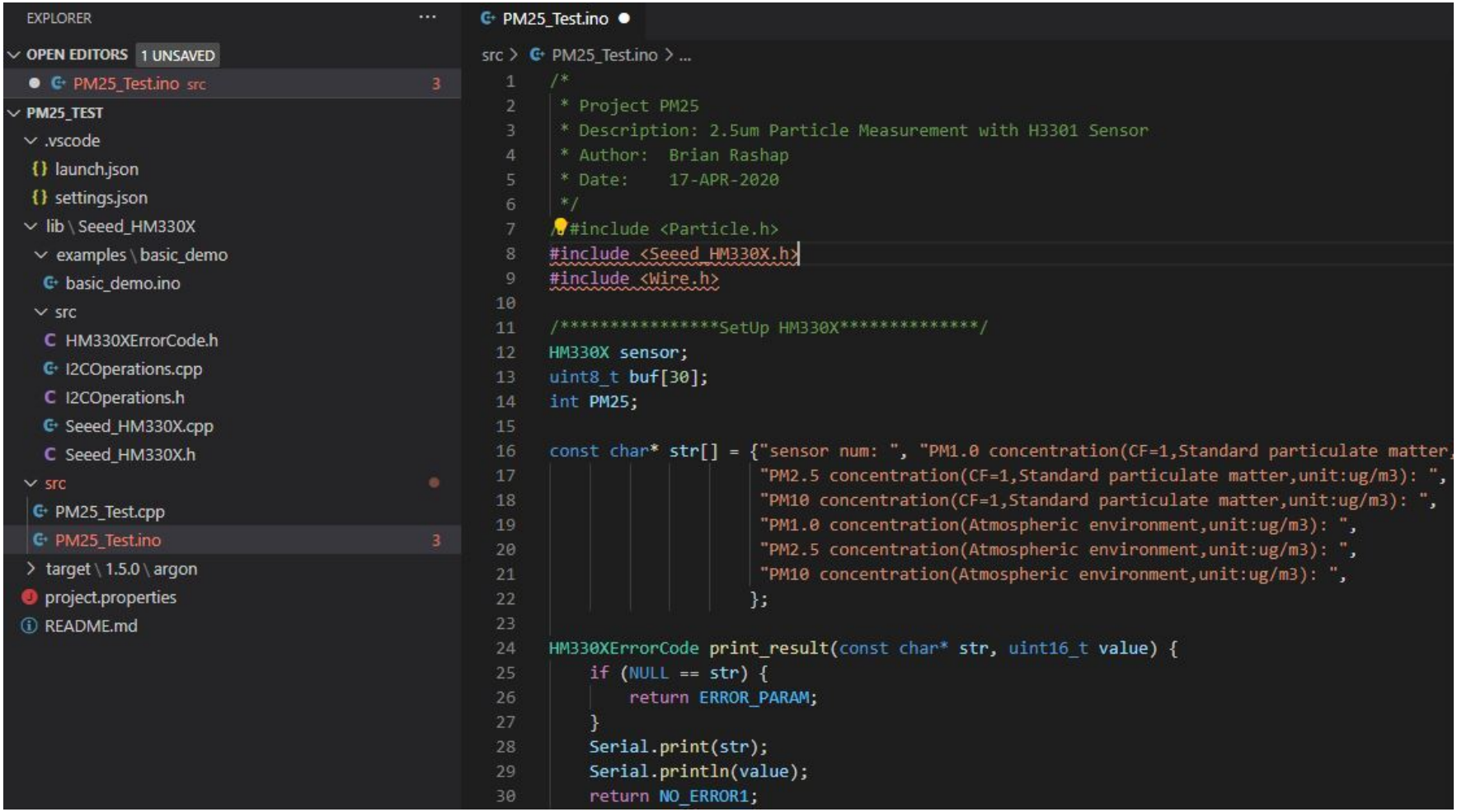
1. Enter DFU mode from the CLI.

```
particle usb configure
```

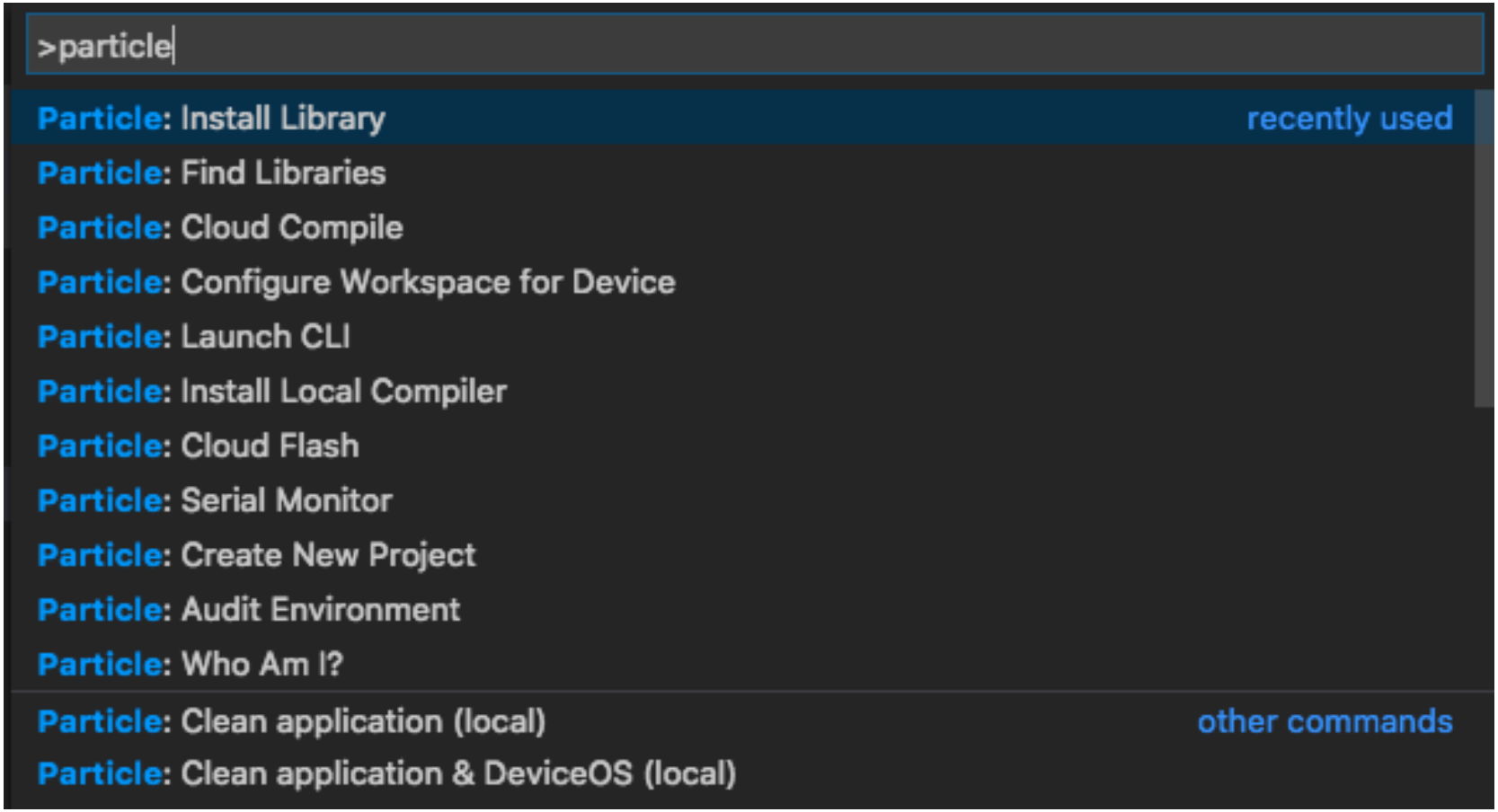
Argon LED Modes

Mode	LED Status
Connected	Breathing Cyan
OTA Firmware Update	Blinking Magenta (red and blue together)
Looking for Internet	Blinking Green
Connecting to Cloud	Rapid Blinking Cyan
Listening Mode	Blinking Blue
Network Reset	Rapid Blinking Blue
WiFi Off	Breathing White
Safe Mode	Breathing Magenta (red and blue together)
DFU (Device Firmware Upgrade	Blinking Yellow
Restore Factory Firmware	Rapid Blinking Yellow
Factory Reset	Rapid Blinking White
Decryption Error	Blinking Cyan followed by 1 Orange Blink
No Internet	Blinking Cyan followed by 2 Orange Blink
No Particle Cloud	Blinking Cyan followed by 3 Orange Blink
Authentication Error	Blinking Cyan followed by 1 Magenta Blink
Handshake Error	Blinking Cyan followed by 1 Red Blink
Decryption Error	Blinking Cyan followed by 1 Orange Blink
SOS - Firmware Crash	Blinking Red - 3 short, 3 long, 3 short, error code

Directory Structure - VERY IMPORTANT

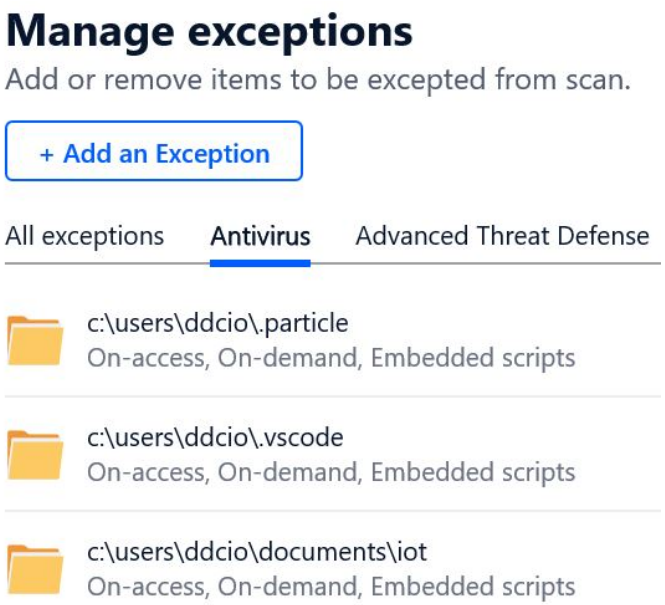


Command Palette - Ctrl-Shift-P



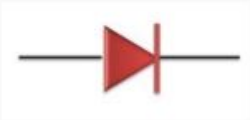
Improve Compile Time

Anti-virus programs scan everything that VSCode is doing. This leads to long compile times. To reduce the compile times, you can exclude your particle code from real-time virus scans.

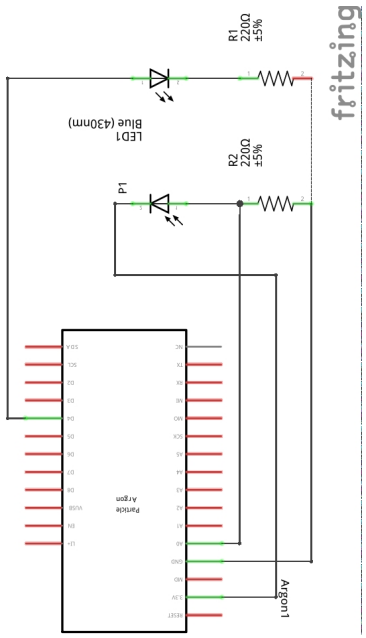


L12_HelloParticle

Photodiode

PARAMETERS	DIODE	PHOTODIODE
Definition	A diode is two terminal device which conducts when it is forwards biased.	A photodiode is a two terminal device which conducts when it is reversed biased.
Circuit symbol		
Main Function	Diode is mainly used as a switch.	Photodiode is used for conversion of light energy into electrical energy.
Material Used	Germanium or silicon, any of these two can be used.	Silicon is used for manufacturing photodiode. An anti-reflective layer of Silver Nitride is used for coating.
Applications	Used in clippers, clampers, rectifiers etc.	Used in optoelectronic device, camera, optocouplers etc.

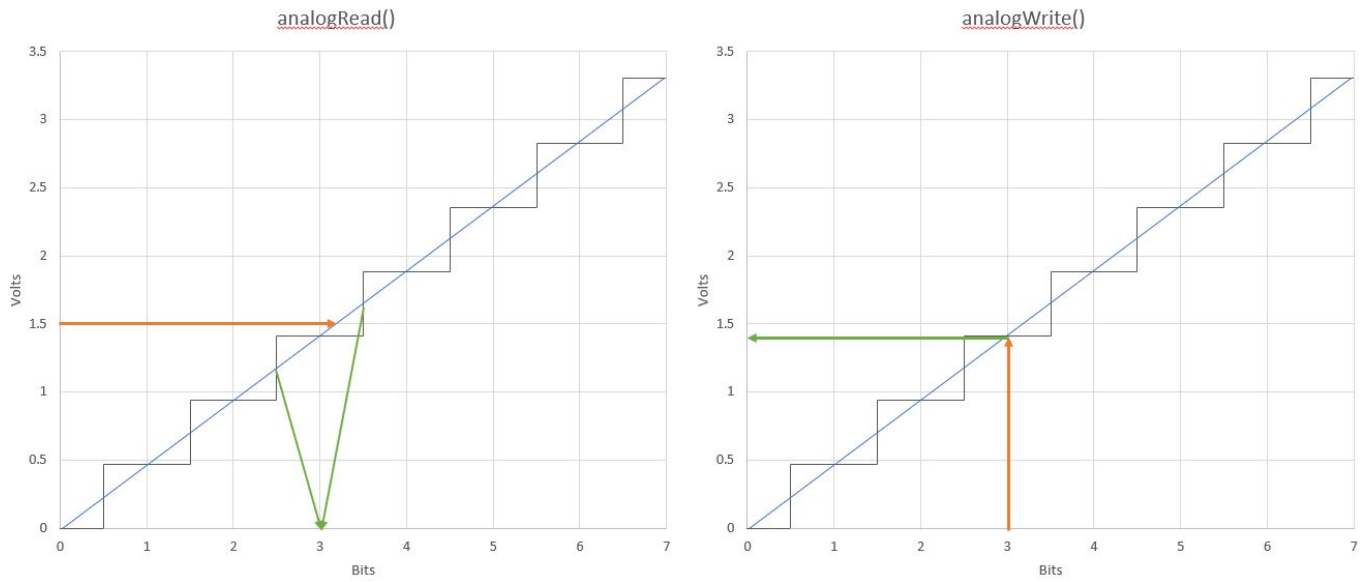
Assignment: L12_HelloParticle



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L12_01_HelloParticle
   - Blink the onboard LED (Pin D7).
2. L12_02_HelloNightLight
   - The anode of the photodiode is connected to Pin A0. *Note, unlike an LED, the cathode (short pin) of the photodiode is connected to 3.3V.*
   - The LED anode to Pin D4.
   - Using analogRead/digitalWrite, turn on the LED when the photodiode is dark.
   - Using analogWrite, turn on the LED slowly as the room darkens.

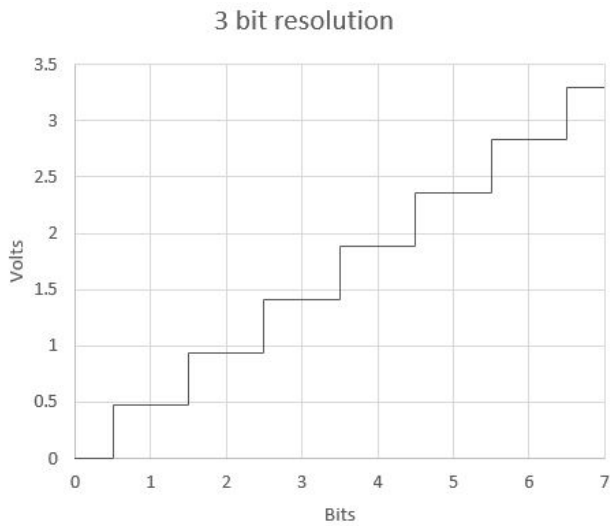
Analog Resolution

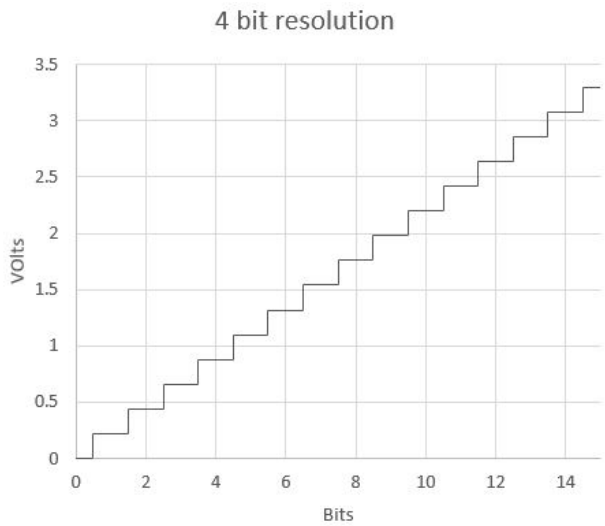


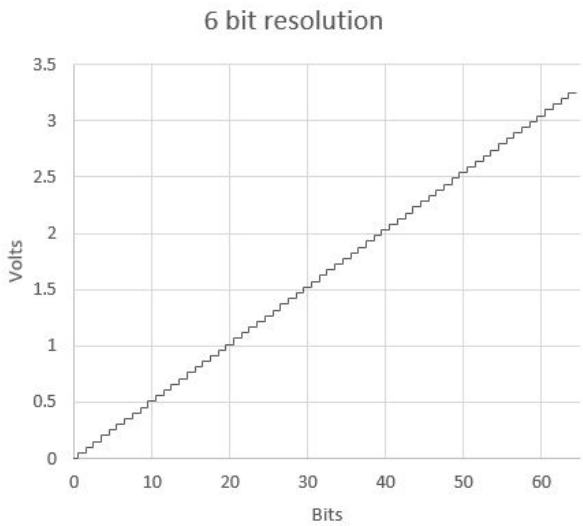
Assignment L12_03_Resolutions

- Using a known voltage value (i.e., 3.3V), determine the resolution (in bits) of analogRead() on the Argon.
- analogWrite the value 63 and measure the resulting voltage with your voltmeter. Use this to determine the resolution (in bits) of analogWrite(). Hint: the max value will produce 3.3V.

Analog Resolution - DAC



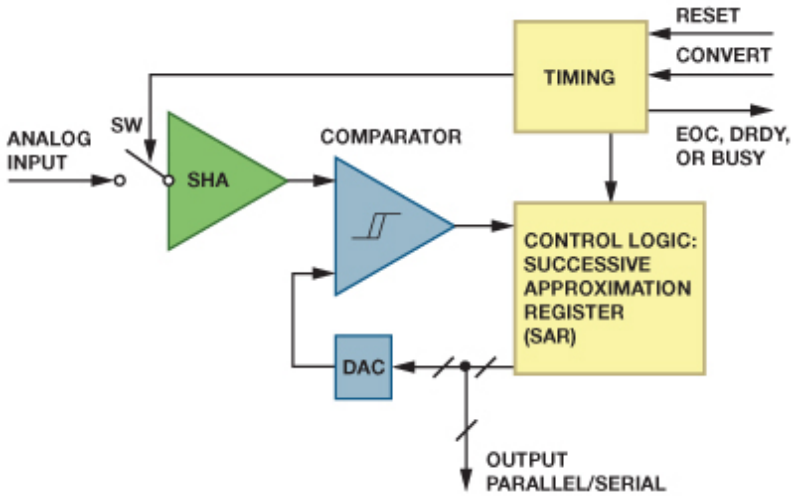


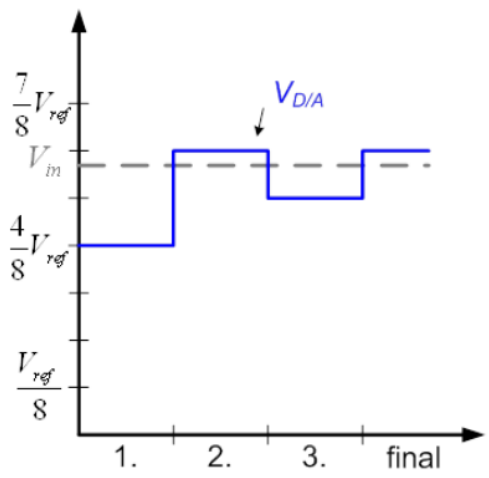


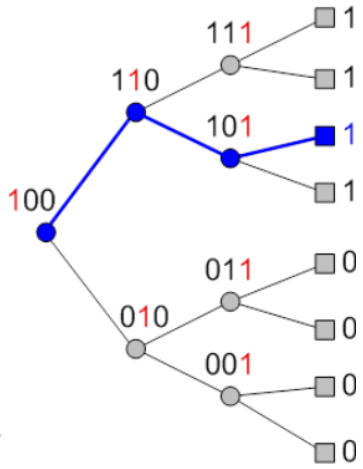
analogWrite() resolution can be modified between 2 and 31 bits.

```
1 analogWriteResolution(pin,bits);
```

Analog Resolution - ADC

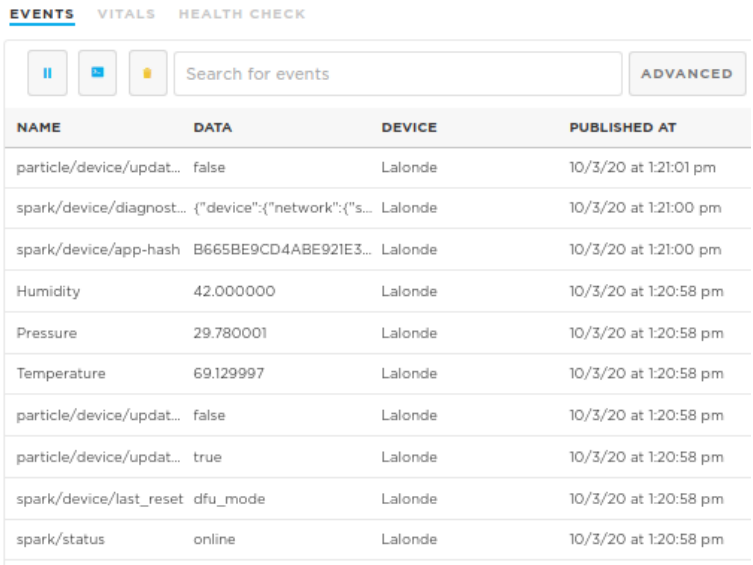






Particle Publish

One of the advantages of the Argon is seamless publishing to the Cloud.



NAME	DATA	DEVICE	PUBLISHED AT
particle/device/updat...	false	Lalonde	10/3/20 at 1:21:01 pm
spark/device/diagnost...	{"device":{"network":{"s...	Lalonde	10/3/20 at 1:21:00 pm
spark/device/app-hash	B665BE9CD4ABE921E3...	Lalonde	10/3/20 at 1:21:00 pm
Humidity	42.000000	Lalonde	10/3/20 at 1:20:58 pm
Pressure	29.780001	Lalonde	10/3/20 at 1:20:58 pm
Temperature	69.129997	Lalonde	10/3/20 at 1:20:58 pm
particle/device/updat...	false	Lalonde	10/3/20 at 1:20:58 pm
particle/device/updat...	true	Lalonde	10/3/20 at 1:20:58 pm
spark/device/last_reset	dfu_mode	Lalonde	10/3/20 at 1:20:58 pm
spark/status	online	Lalonde	10/3/20 at 1:20:58 pm

```
1  float temp, prs, hum;    // BME280 variables
2  String Temp, Prs, Hum;   // Strings to hold BME280 values
3
4  void loop() {
5    Temp = String(temp);
6    Prs = String(prs);
7    Hum = String(hum);
8
9    Particle.publish("Temperature", Temp, PRIVATE);
10   Particle.publish("Pressure", Prs, PRIVATE);
11   Particle.publish("Humidity", Hum, PRIVATE);
12 }
```

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

JSON Parser Generator

Creating objects in JSON are straightforward but can be tedious. There is a JSON Parser available to simplify the process.

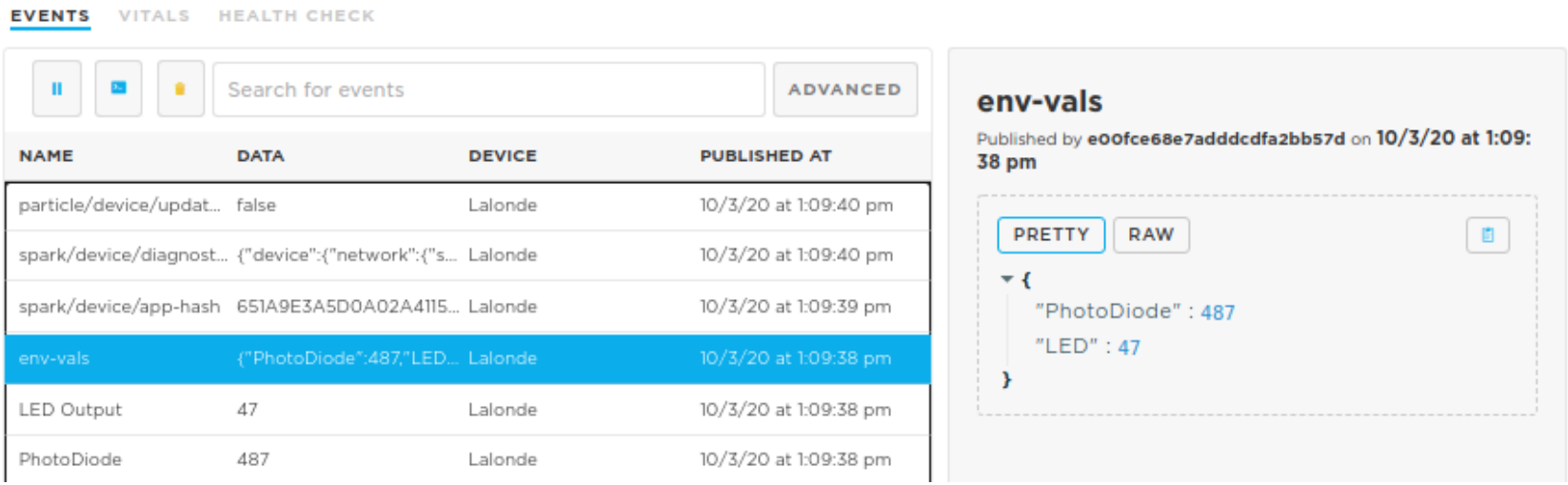
```
#include <JsonParserGeneratorRK.h>

void createEventPayLoad(float tempValue, float presValue, float humValue) {
  JsonWriterStatic<256> jw;
  {
    JsonWriterAutoObject obj(&jw);

    jw.insertKeyValue("Temperature", tempValue);
    jw.insertKeyValue("Pressure", presValue);
    jw.insertKeyValue("Humidity", humValue);
  }
  Particle.publish("env-vals",jw.getBuffer(), PRIVATE);
}
```

Installing Libraries

Using the Command Palette within VSCode:

- ctrl-shift-p → Particle: Find Libraries
- ctrl-shift-p → Particle: Install Library

Assignment: L12_HelloParticle



1. L12_04_HelloPublish
   - Using Particle.publish(), send the photodiode and LED values to the Particle Cloud.
   - Use the Command Palette to Install Library JsonParserGeneratorRK.
   - Use this library to send the same data using the JSON Generator.

Struct datatype and Member Operators

struct enables the programmer to create a variable that structures a selected set of data.

```
struct Employee {
  char name[10];
  int idNumber;
  float salary;
}

Employee instructor;
Employee IoTengineers[10];

void setup {
  instructor.name = “Brian”;
  instructor.idNumber = “42”;
  instructor.salary = 212.47;
  IoTengineers[1].name = “Sally”;
}
```

struct name

Members of the structure

Declare individual variable of data type Employee

Declare an array of data type Employee

Struct datatype and Member Operators

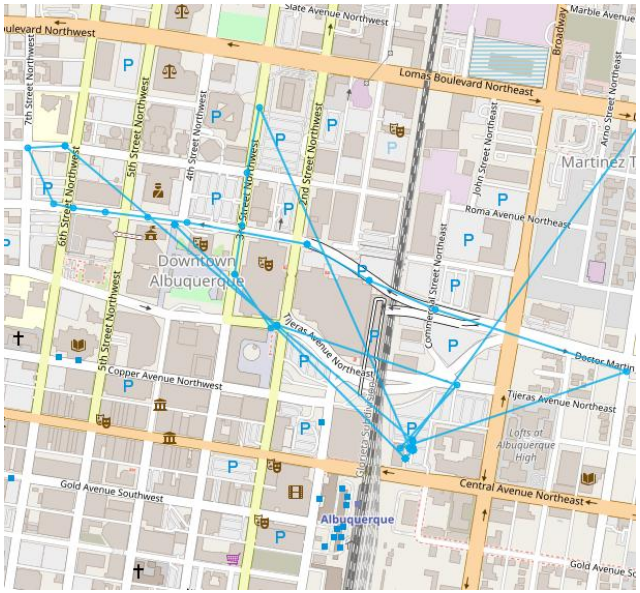
struct creates a variable that structures a set of data.

```
1  struct geo {    // create a struct of name geo that hold GPS data
2    float lat;
3    float lon;
4    int alt;
5  }; // ends with a ; as struct can also declare variables while being declared
6
7  geo myLoc;          //declare a variable called myLoc of type geo
8  geo locations[13];  //declare an array of type geo
9
10 void setup() {
11   //initialize myLoc with latitude, loggitude, and altitude
12   myLoc.lat = 35.120606;
13   myLoc.lon = -106.65818;
14   myLoc.alt = 1517;
15
16   Serial.printf("Location: lat %f, lon %f, alt %i \n",myLoc.lat,myLoc.lon,myLoc.alt);
17 }
```

The . (dot) operator and the $->$ (arrow) operator are used to reference individual members of structures.

- The dot operator is applied to the actual object.
- The arrow operator is used with a pointer to the object (we will learn about Pointers in Lesson 15).

Assignment: L12_HelloParticle



1. L12_05_HelloGPS
   - Create a struct to hold GPS location
   - Manually store the GPS location of your favorite pizza parlor into a variable of this data type.
   - Create a function that has as a single parameter a GPS location struct variable. The function should:
     - Create a JSON playload from the GPS coordinates.
     - Publish to Particle Cloud

Note: functions that use a struct as an parameter need to be declared in the header (right before void setup()).

Declaring, Calling, and Defining

```
1  struct Employee {
2    String name;
3    int salary;
4    };
5
6  Employee instructor;
7
8  /*
9   * Declare the function: while declaring functions is optional,
10  * functions with   struct must be declared in the header section.
11  */
12 void printSalary(Employee person);
13
14 void setup() {
15   Serial.begin(9600);
16   waitFor(Serial.isConnected,5000);
17   delay(1000);
18
19   instructor.name = "Brian";
20   instructor.salary = 212.47;
21
22   //call function
23   printSalary(instructor);
24 }
25
26 void loop() {}
27
28 //define the function
29 void printSalary(Employee person) {
30   Serial.printf("%s salary is %i\n",person.name.c_str(),person.salary);
31 }
```

SYSTEM_MODE

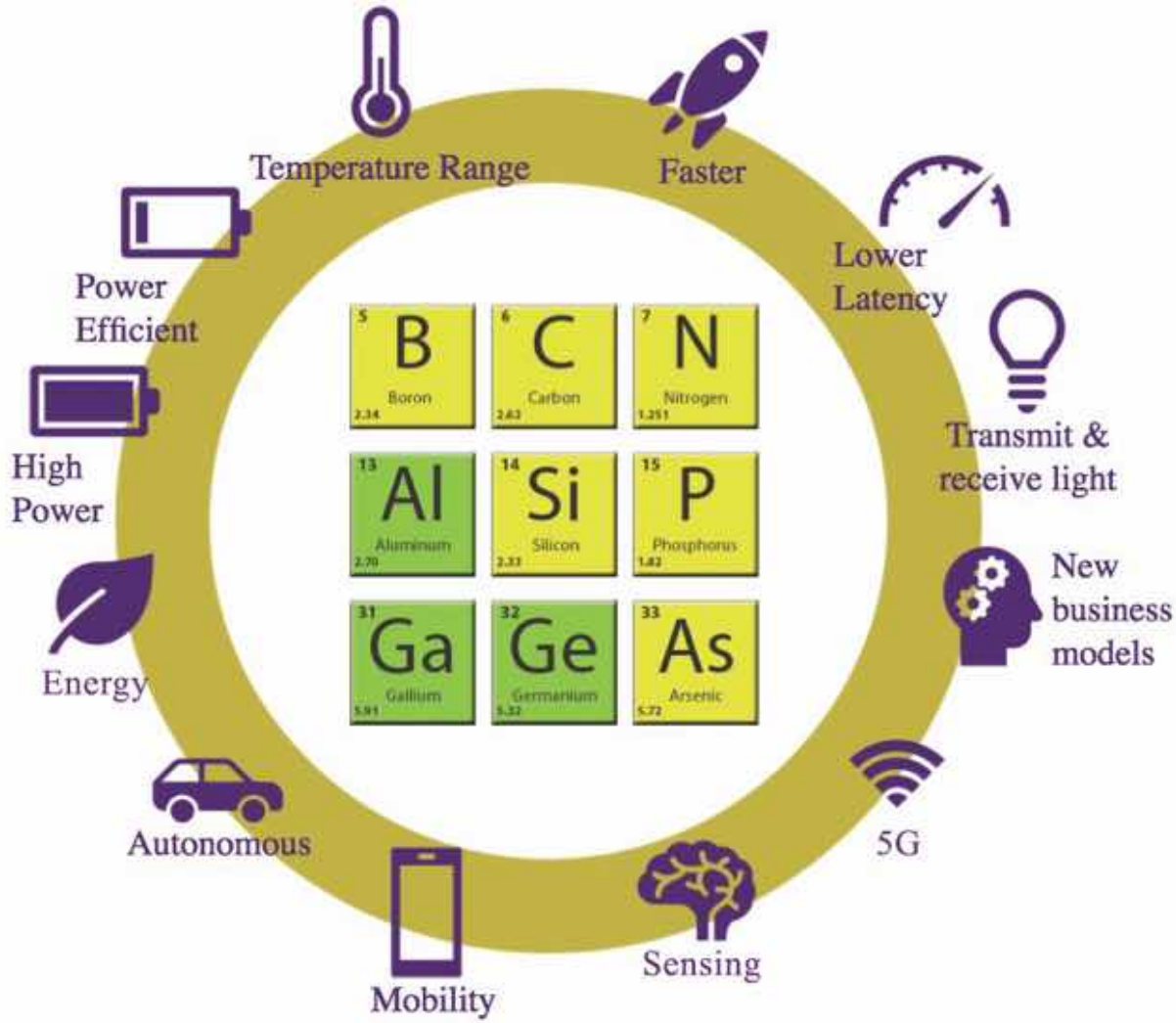
System modes help control how the device manages the connection with the cloud. By default, the device connects to the Cloud and processes messages automatically. However there are many cases where a user will want to take control over that connection. There are three available system modes:

- AUTOMATIC,
- SEMI_AUTOMATIC,
- MANUAL.

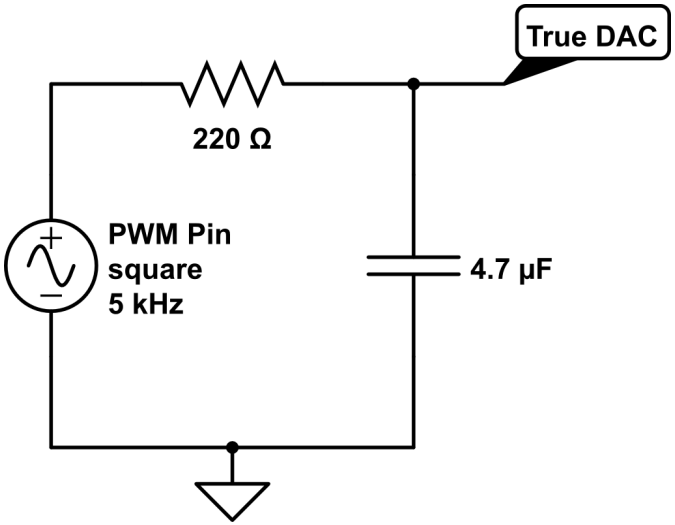
```
1
2 //The below is placed in the header before Void Setup()
3
4 // SYSTEM_MODE(AUTOMATIC);      // Default if no SYSTEM_MODE included
5    SYSTEM_MODE(SEMI_AUTOMATIC); // Uncomment if using without Wifi
6 // SYSTEM_MODE(MANUAL);         // Fully Manual
```

L13_Semiconductors

Semiconductors



But First... True DAC on Argon

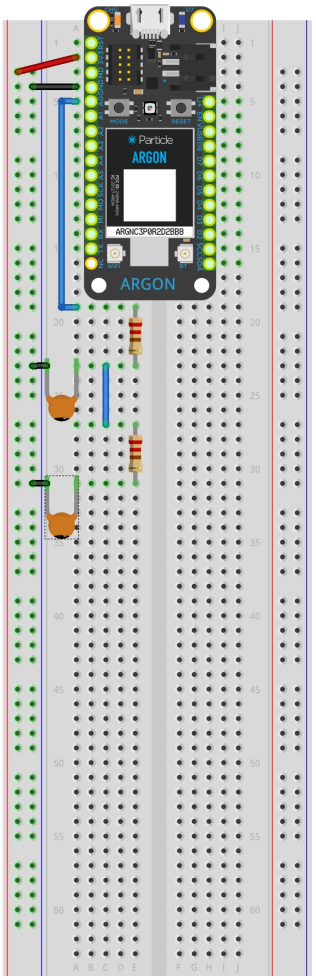


analogWrite(*pin*, *value*, *frequency*)

The Particle microcontrollers do not have a true DAC (Digital to Analog Converter) like pin A14 on the Teensy. However, we can convert a PWM pin to a DAC signal using a low pass filter.

- In lesson L06_Encoders we learned about Low Pass Filters.
- The PWM signal oscillates at 500 Hz, but we can increase up to 5,000 Hz using a third parameter in analogWrite().
- Using $R = 220\Omega$ and $C = 4.7\mu F$ will give a $f_c = 153Hz$.

Assignment: L13_00_DAC

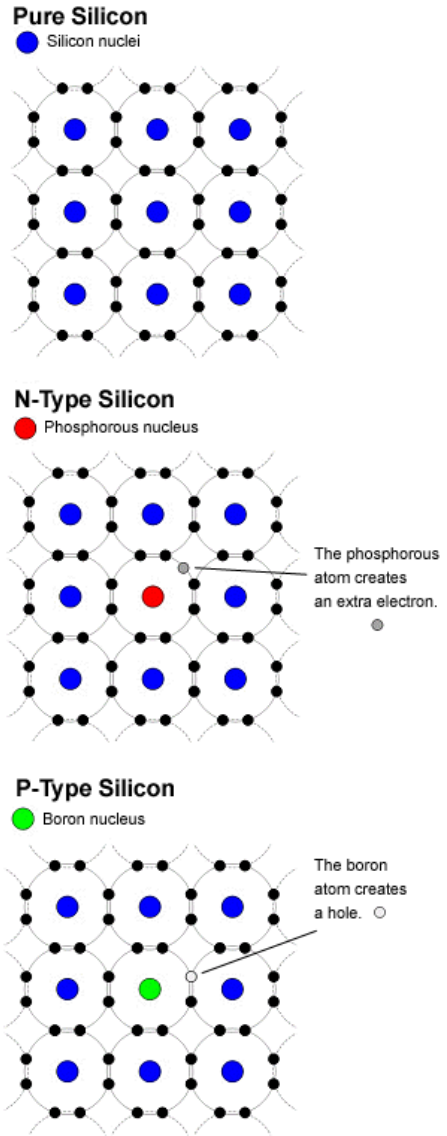


Using the full size breadboard and $4.7\mu F$ capacitors:

- Output 0.825V to Pin D4.
- On Pin A0, create a PWM output of a sine wave at $75Hz$ using a PWM frequency of $5kHz$.
- Add in a low pass filter of $150Hz < f_c < 750Hz$ to create a true DAC output.
- Add in a second low pass filter with an $f_c < 35Hz$.
- On the oscilloscope, measure the D4, A0, post-DAC, and low-pass filter signals. Record what you observe in your lab notebook.

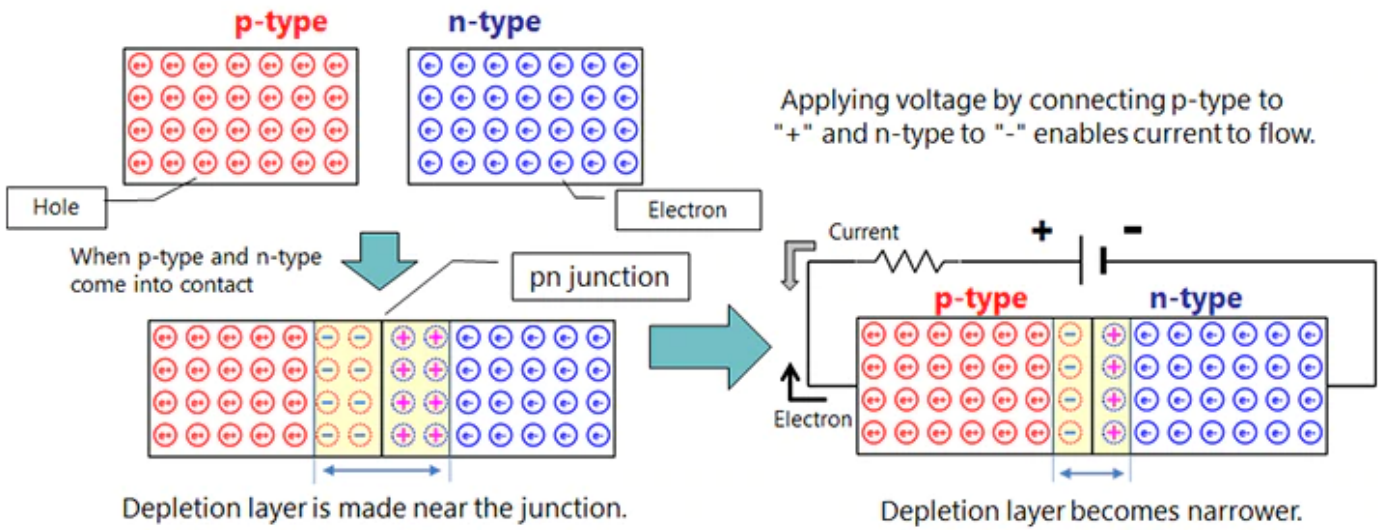
Semiconductor

- A silicon atom has four electrons in its outer shell and bonds tightly with four surrounding silicon atoms creating a crystal matrix with eight electrons in the outer shells. The tight bonds make pure silicon non-conducting.
- Phosphorus has five electrons, and when combined, the fifth electron becomes a "free" electron that moves easily within the crystal when a voltage is applied.
- Boron has only three electrons in its outer shell and can bond with only three of surrounding silicon atoms. Thus one silicon atom has a vacant location in its outer shell, called a "hole," that readily accepts an electron.



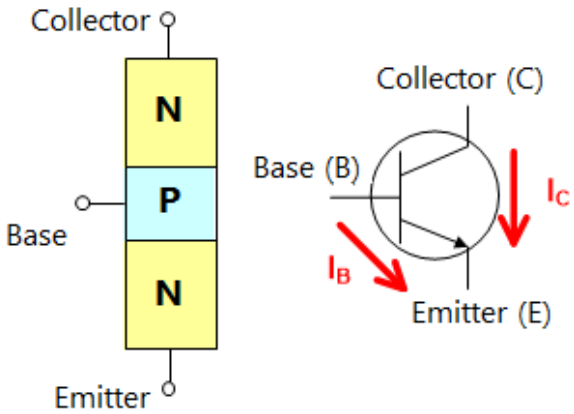
pn junction diode

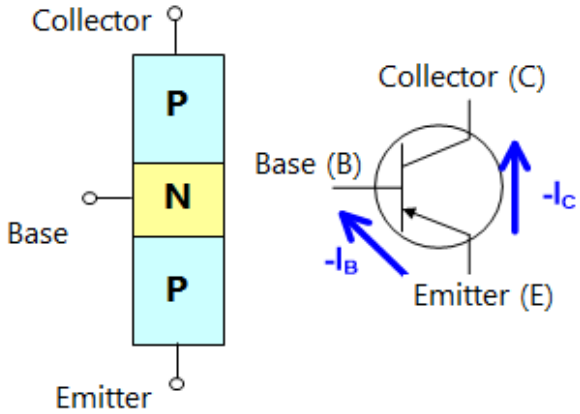
- When p-type and n-type semiconductors are bonded, holes and free electrons are attracted, combine, and disappear near the boundary. Since there are no carriers in this area, it is called a depletion layer and it is an insulator.
- A positive voltage applied to the p-type region causes electrons to flow sequentially from the n-type. The electrons will first disappear by combining with holes, but excess electrons will move to the positive pole and current will flow.



Bipolar Junction Transistor

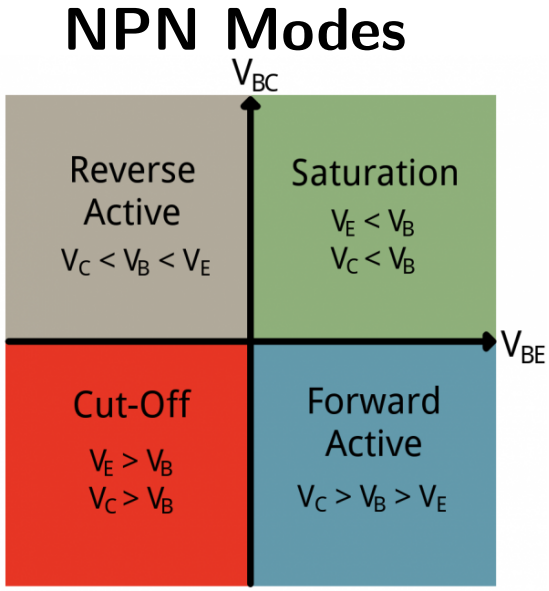
The transistor has three regions, namely base, emitter and collector. The emitter is a heavily doped terminal and emits electrons into the base. The base terminal is lightly doped and passes the emitter-injected electrons on to the collector. The collector terminal is intermediately doped and collects electrons from base. This collector is large as compared with other two regions so it dissipates more heat.





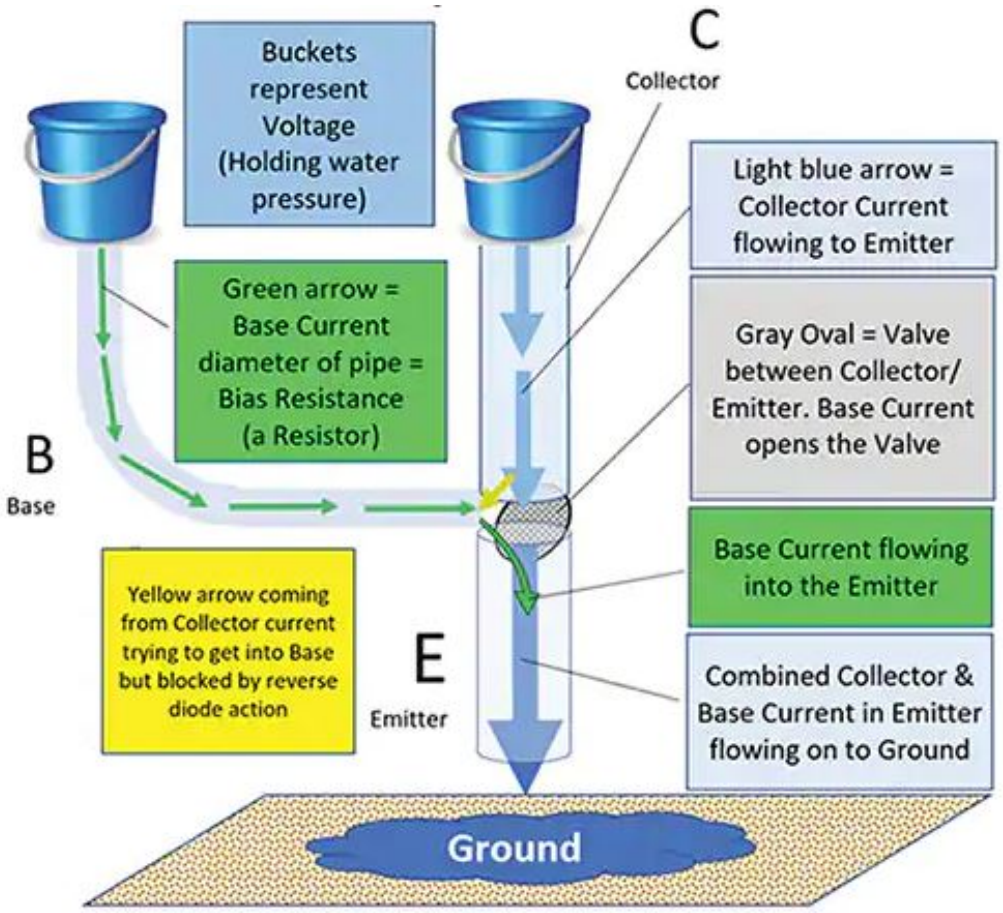
Bipolar Junction Transistor - Modes of Operation

- **Saturation:** Current freely flows from collector to emitter. (ON Switch)
- **Cut-off:** No current flows from collector to emitter. (OFF Switch)
- **Active:** The current from collector to emitter is proportional to the current flowing into the base. (Amplifier)
- **Reverse-Active:** Like active mode, the current is proportional to the base current, but it flows in reverse from emitter to collector (not the purpose transistors were designed for).



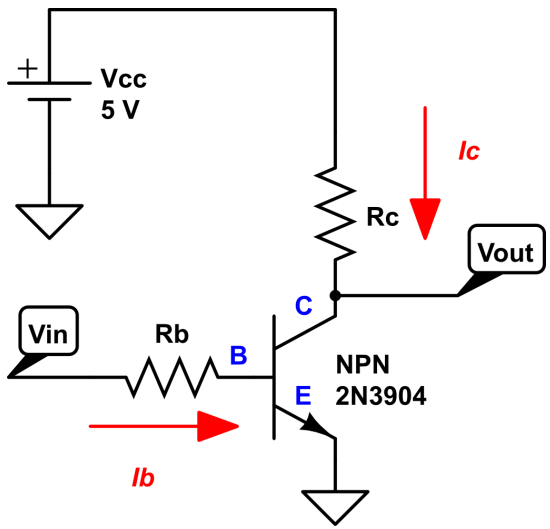
Voltage relations	NPN Mode	PNP Mode
$V_E < V_B < V_C$	Active	Reverse
$V_E < V_B > V_C$	Saturation	Cutoff
$V_E > V_B < V_C$	Cutoff	Saturation
$V_E > V_B > V_C$	Reverse	Active

Water Analogy



Active Mode NPN Transistor Circuit

If you apply a voltage V_{IN} that is high enough to forward-bias the base-to-emitter junction, current (I_B) will flow from the input terminal, through R_B, through the BE junction, to ground. Current (I_C) will also flow through R_C and the collector-to-emitter portion of the transistor.

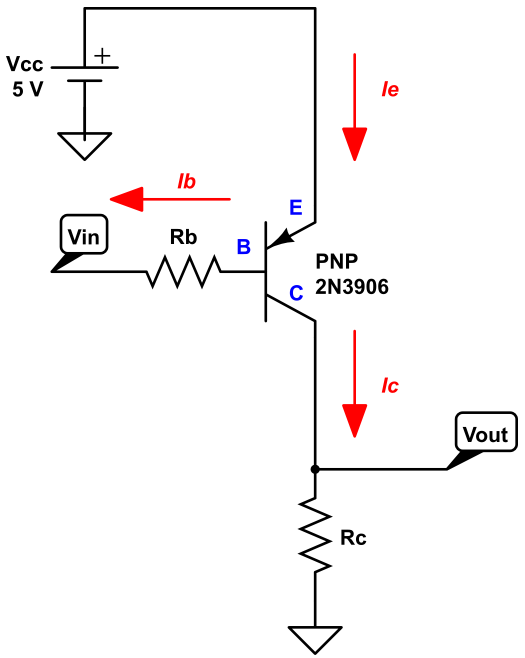


NOTE: V_{OUT} is an amplified but inverted signal of V_{IN}.
This simple circuit will step-up a 0 - 3.3V output from the microcontroller to 0 - 5.0V (inverted). The low impedance of the output will also provide sufficient current to drive a higher current device (e.g., a relay).

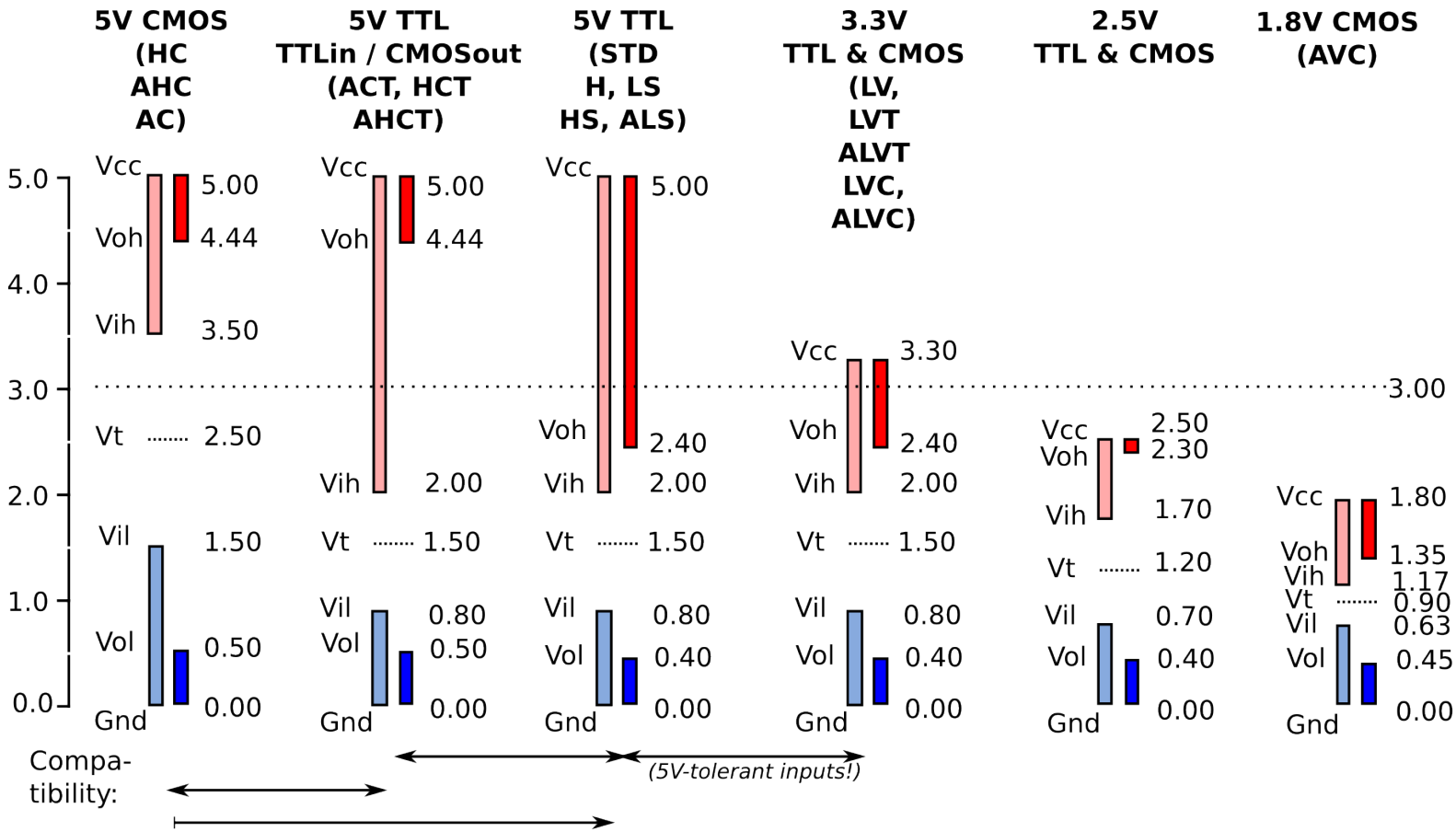
PNP Transistor

NPN Transistors are more common than PNP for a number of reasons:

- The voltage and current behavior of an NPN transistor is significantly more intuitive.
- When a switch or driver circuit is required, NPNs provide a more straightforward interface to digital output signals (such as a control signal generated by a microcontroller).
- NPNs are higher performance (faster switching speeds) due to higher mobility of electrons vs holes.



Logic Voltage Level Standards

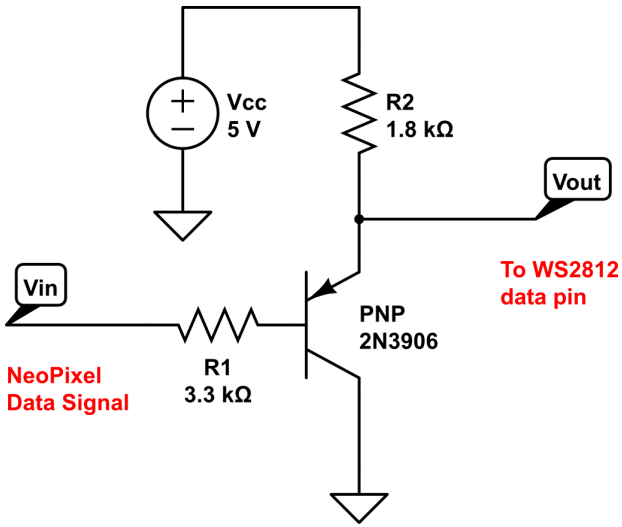


Data source: EETimes, A brief recap of popular logic standards (Mark Pearson, Maxim).

Or, what is wrong with the NeoPixels.

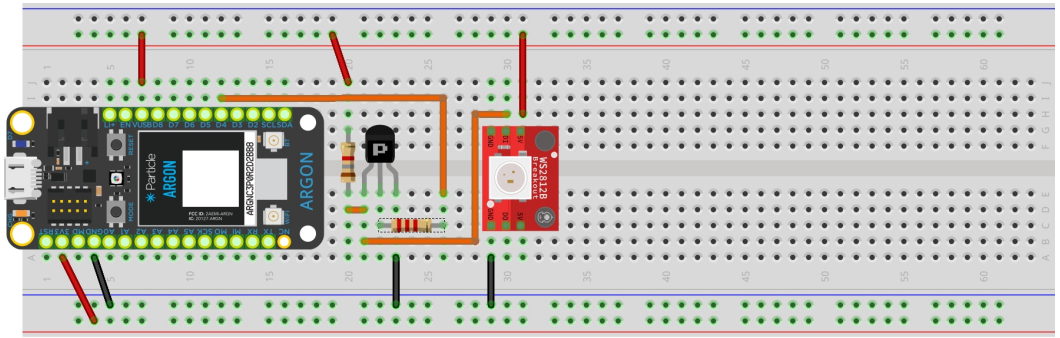
Emitter Follower - Saturation and Cutoff Mode

- NeoPixels are designed around 5V CMOS transistors.
  - $V_{IH} > 3.5V$
  - 3.3V Microcontroller $V_{OH} = 3.3V$
- An Emitter Follower (i.e., a PNP transistor wired backwards) is a current amplifier, but will also produce a $V_{OUT} = 3.9V$.
- Alternatively, the first NeoPixel could be sacrificed by reducing its V_{cc} to 4.3V with a diode.

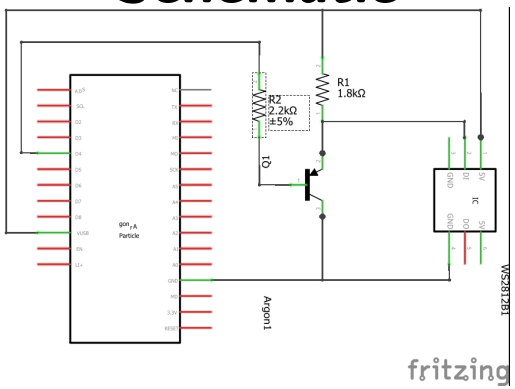


Emitter Follower Layout

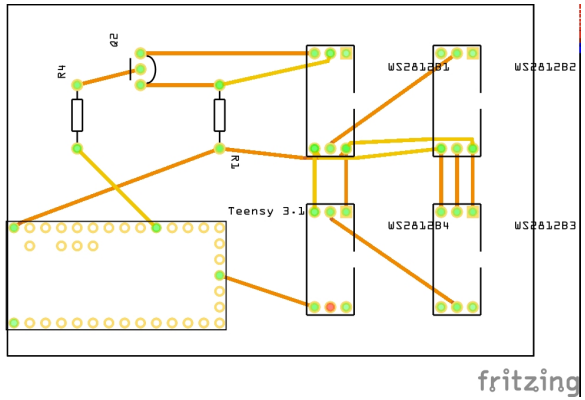
Fritzing



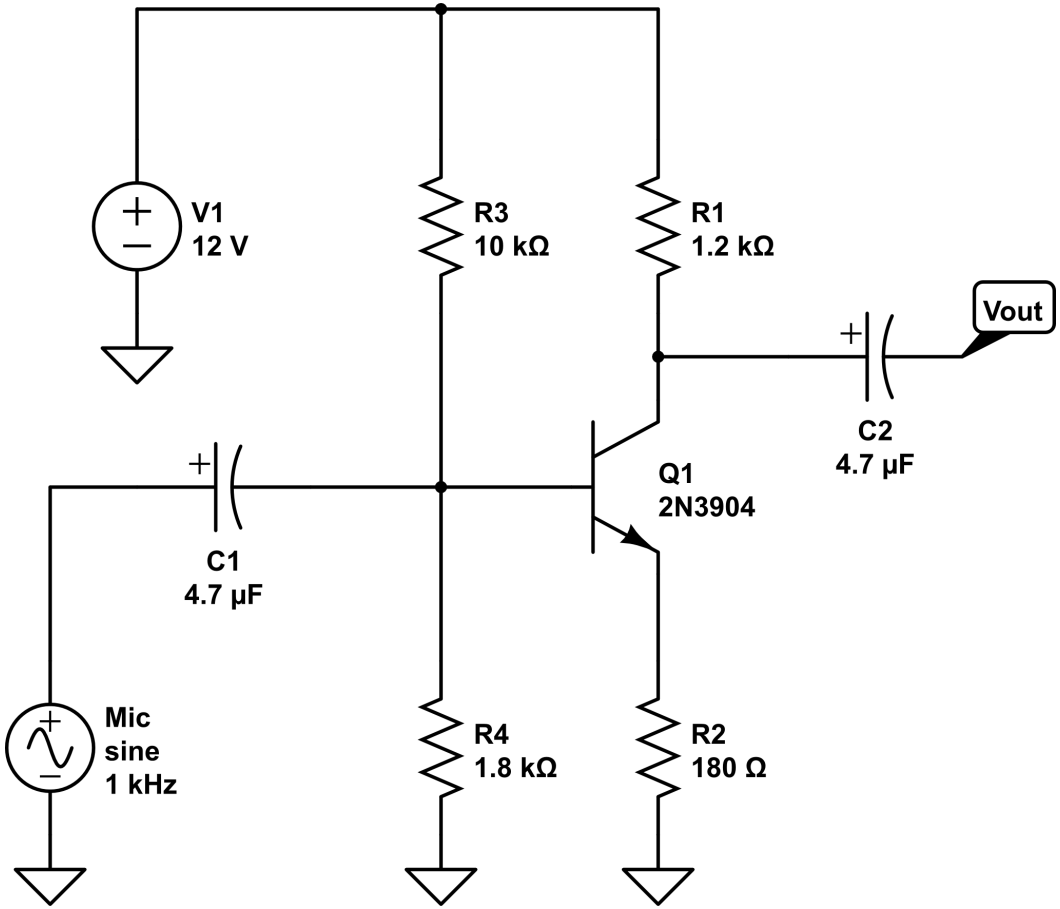
Schematic



PCB Layout

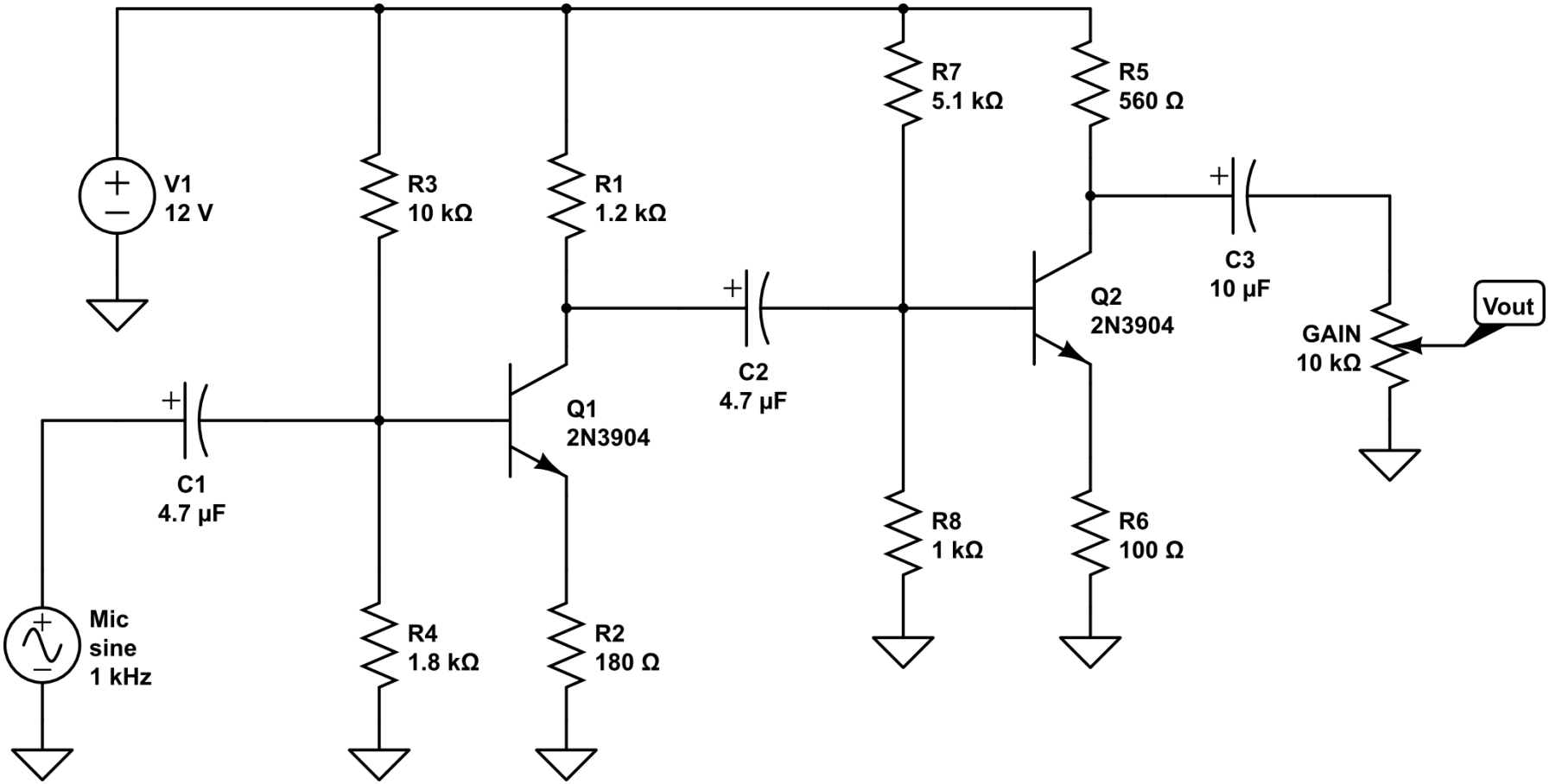


Typical NPN Pre-Amplifier Circuit

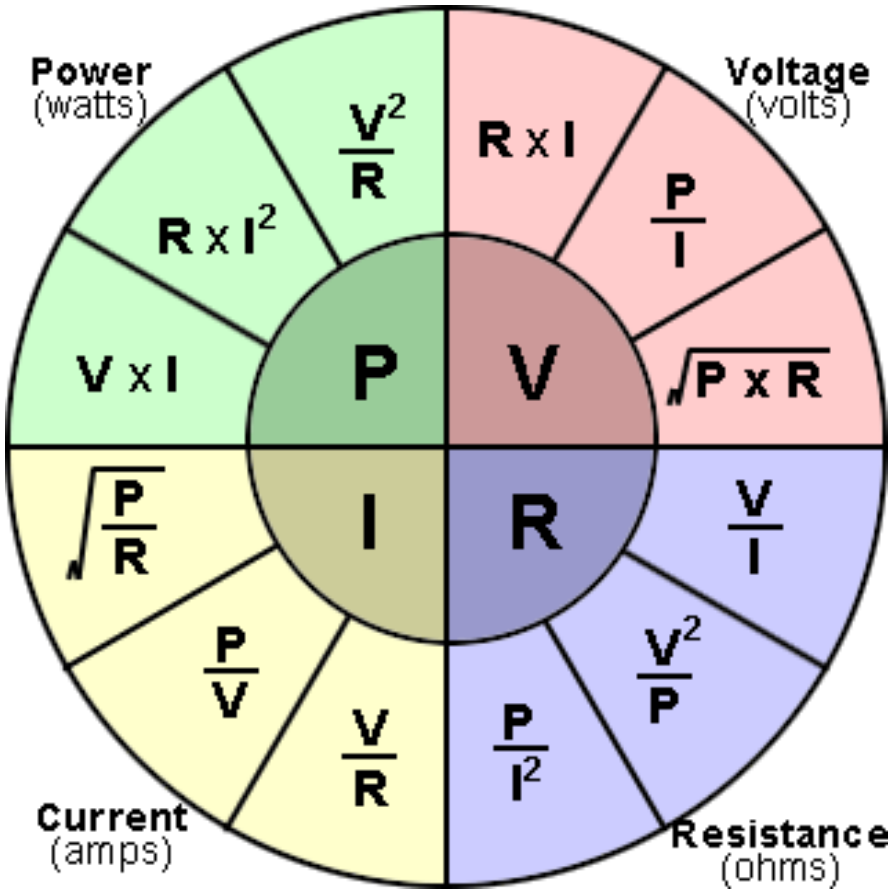


This is referred to as a Common Emitter amplifier as the Emitter ground is common to both the input and output voltage. The Common Emitter amplifies both voltage and current.

Two Stage Pre-Amplifier



Ohm's Law - Revisited



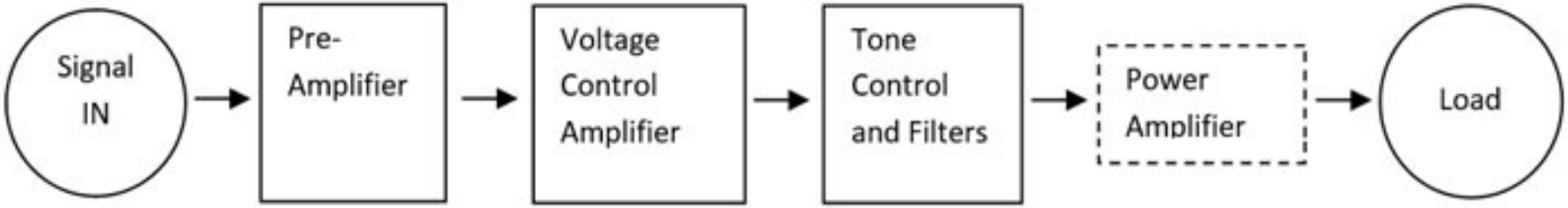
PreAmp vs PowerAmp

PreAmp:

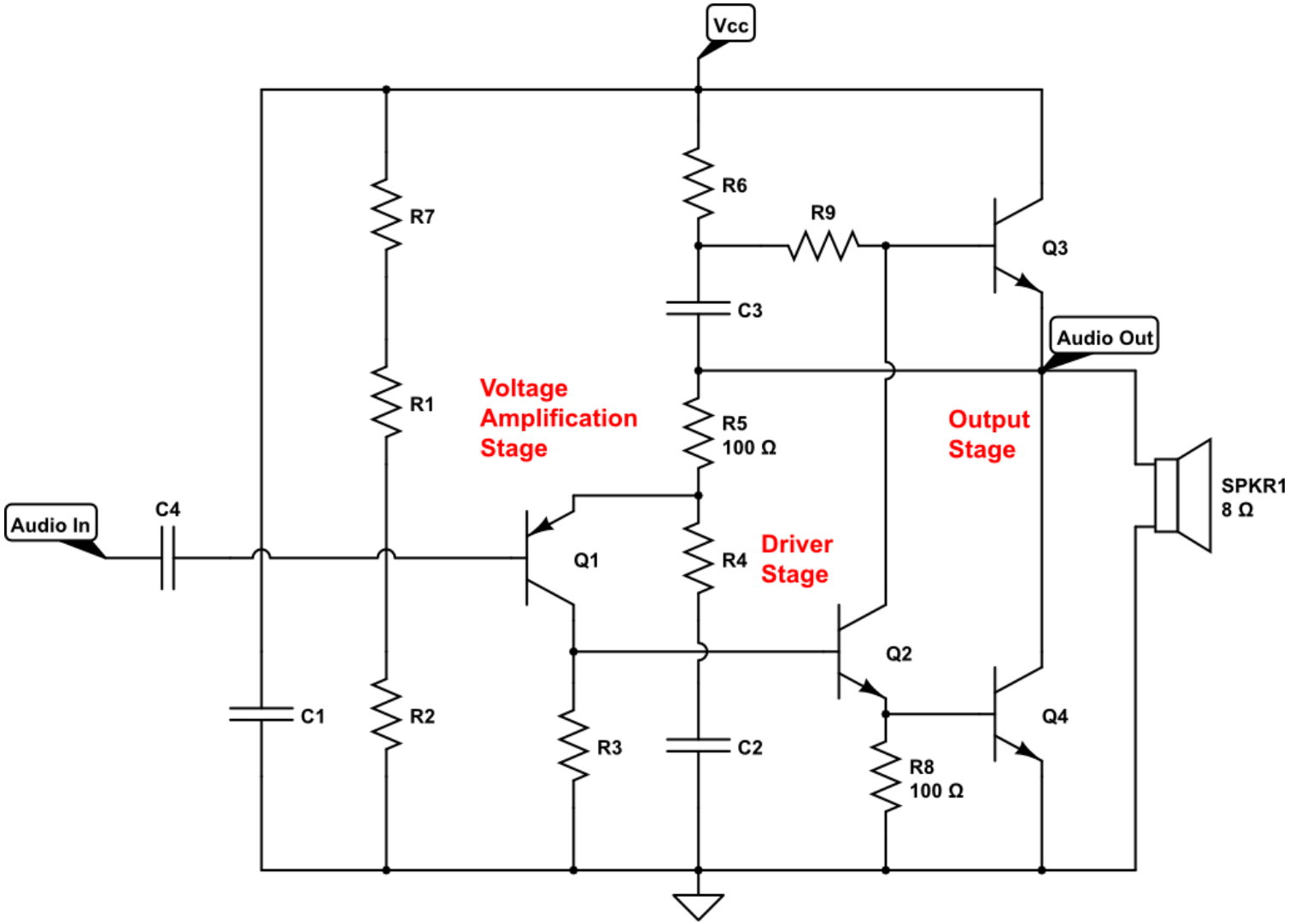
- A preamp boosts the signal up to 'line level'.
- Guitar PreAmp
  - A pure guitar signal typically sounds weak and anaemic, as is seen if a guitar is directly plugged PA system.
  - A preamp is able to raise a guitar's signal up to an audible volume.
  - It can also be used to affect the audio characteristics.

PowerAmp:

- A power amp boosts that line level signal even more – so that it can be projected through speakers.

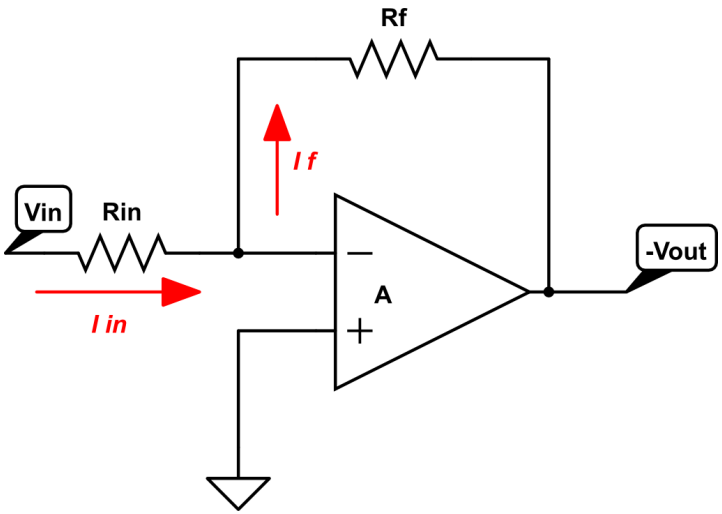


Power Amplifier



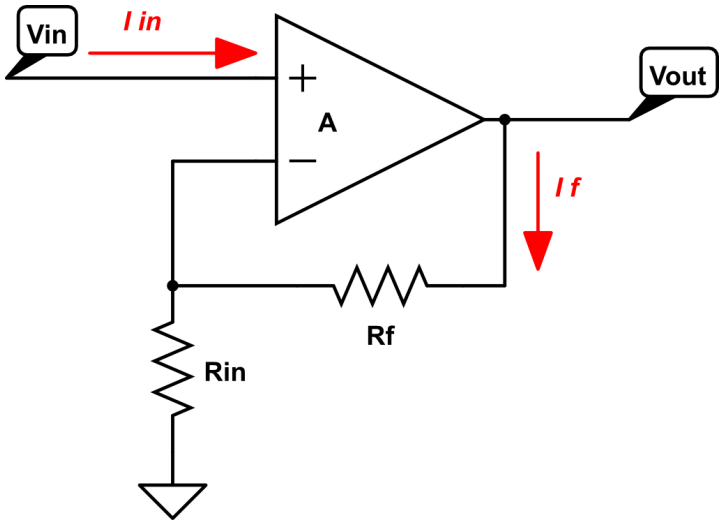
Op Amp Lesson

Inverting Op Amp



$$A = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_{in}}$$

Non-inverting Op Amp



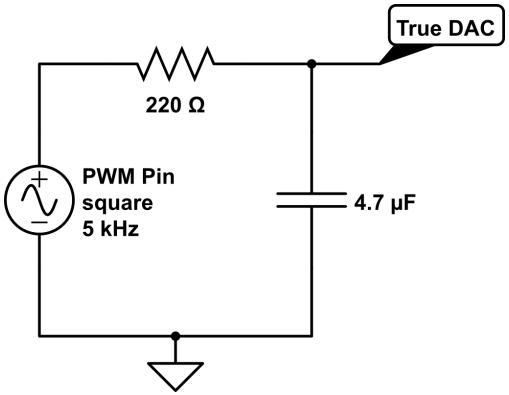
$$A = \frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_{in}}$$

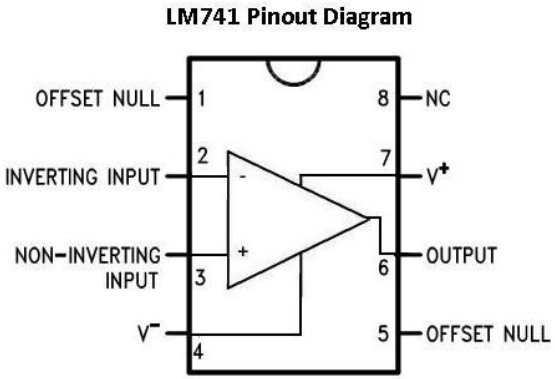
Power the OpAmp with $V^+ = 12V$ and $V^- = -12V$ using TPS5430

Assignment: Amplifiers

Using your breadboard from L13_00_DAC:

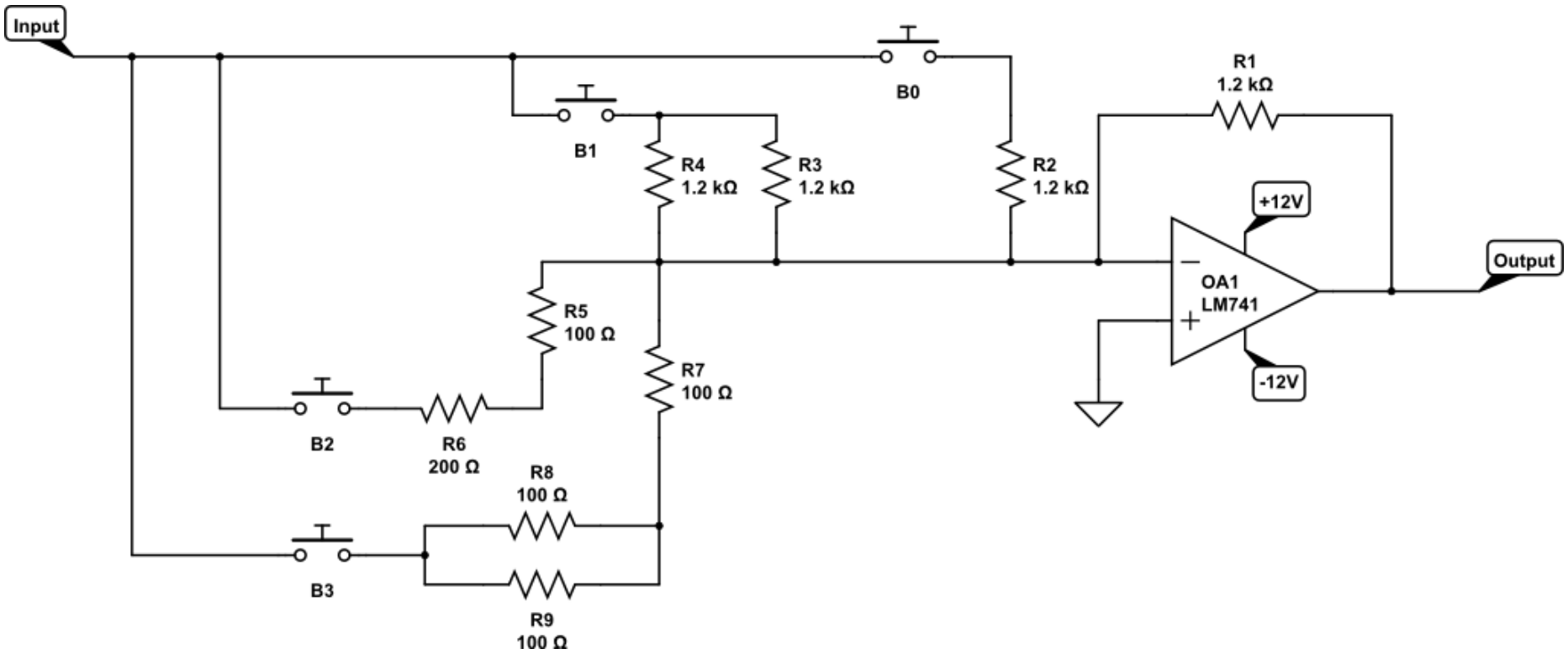
1. L13_01_NeoPixel
   - Add a Emitter-Follower into your NeoPixel circuit to boost the pixel commands to 5V.
2. L13_02_NPNAmp
   - Amplify your True DAC signal using an NPN preamp.
   - Measure your circuit at each node using the oscilliscope [a].
3. L13_03_OpAmp
   - Replace the NPN preamp with an amplification circuit using the LM741 Op Amp. Use a potentiomter for R_{in}.





[a] If you do not have an oscilloscope, you can use your Teensy with the code from L06_01_lowPass.

L13_04_MysteryCircuit



- Layout circuit in Fritzing
- Build and test circuit with input at Oscilliscope station
- Record output voltage for all combinations of buttons presses
- Figure out what it is doing and why it works.

L14_PlantWatering

More DataTypes - Strings, strings, and char[]

```
1  // A string (lowercase 's') is an array of characters
2    char lastName[7] = "Rashap";
3    char firstName[6] = {'B','R','I','A','N'};
4    char name[12];
5
6    //The "*" indicates a pointer, which we will learn about later
7    char *myName = "Brian";
8
9  // A String is a Class that holds a character array
10   String instructor = "BRIAN RASHAP";
11
12 void setup() {
13   Serial.begin();
14
15   Serial.printf("lastName = %s, %i\n",lastName,sizeof(lastName));
16   Serial.printf("firstName = %s, %i\n",firstName,sizeof(firstName));
17   Serial.printf("name = %s, %i\n",name,sizeof(name));
18   Serial.printf("myName = %s, %i\n",myName,sizeof(myName));
19
20   // We can not use %s for the variable instructor, why?
21 }
```

```
Serial monitor opened successfully:
lastName = Rashap, 7
firstName = BRIAN, 6
name = , 12
myName = Brian, 4
```

Too Much Time On My Hands

When the Particle Argon connects to the Particle Cloud, it synchronizes its clock to the current time.

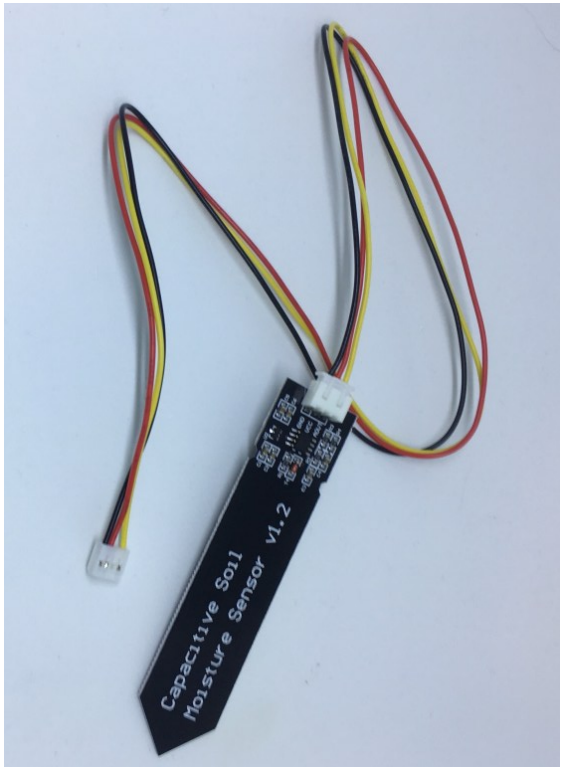
```
// Declare Global Variables in Header
String DateTime, TimeOnly;

void setup() {
  Time.zone(-7);            // MST = -7, MDT = -6
  Particle.syncTime();      // Sync time with Particle Cloud
}

void loop() {
  DateTime = Time.timeStr();              //Current Date and Time from Particle Time class
  TimeOnly = DateTime.substring(11,19); //Extract the Time from the DateTime String

  //%s prints an array of char
  //the .c_str() method converts a String to an array of char
  Serial.printf("Date and time is %s\n",DateTime.c_str());
  Serial.printf("Time is %s\n",TimeOnly.c_str());

  delay(10000); //only loop every 10 seconds
}
```

To learn more about the String Class: `https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/`

Soil Moisture Sensors

Resistive Sensor

Capacitive Sensor

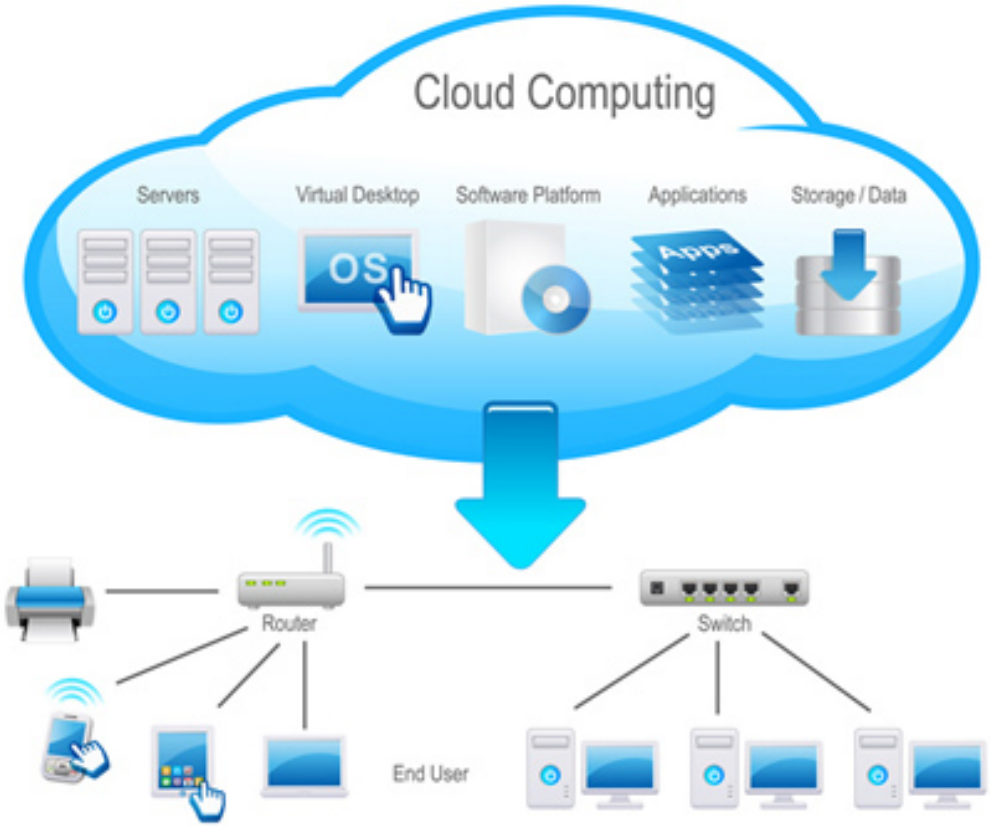


Assignment: L14_SoilMoisture

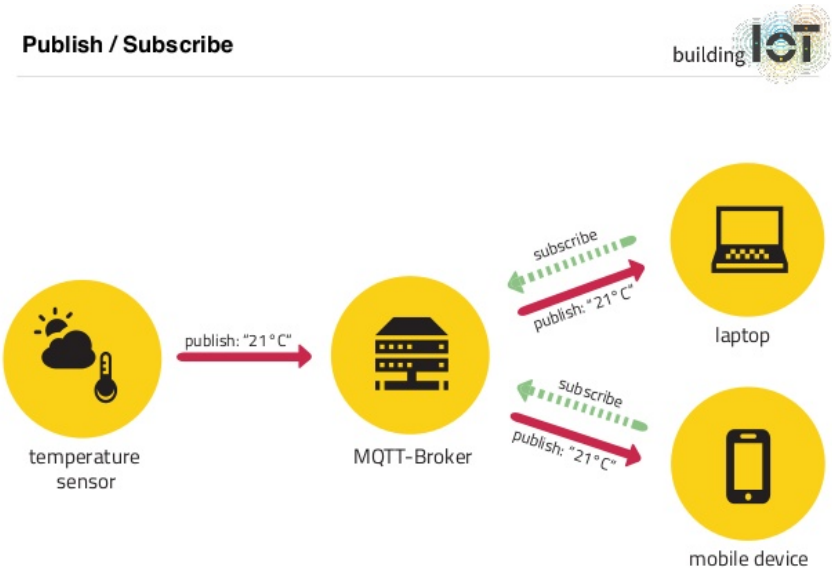
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

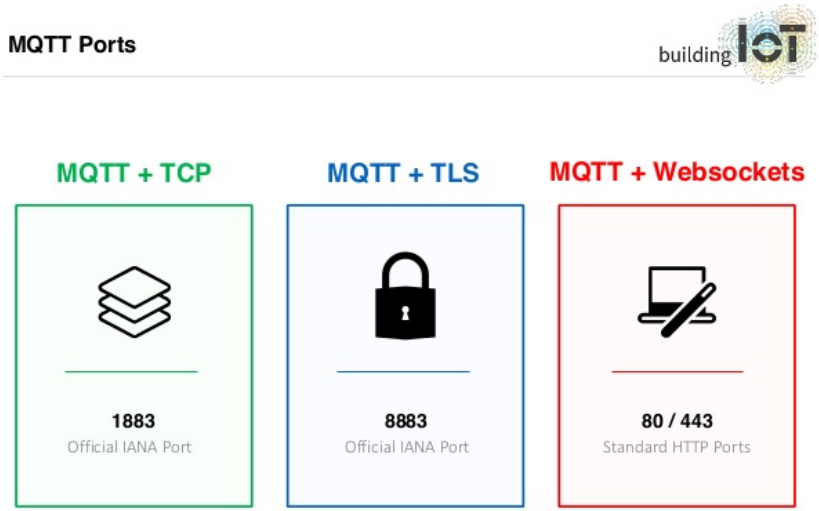
1. L14_01_OLED
   - Install the Adafruit_SSD1306 library.
   - Use I2C_Scan to get the OLED I2C address.
   - Create sample code displaying your name and time to the OLED.
2. L14_02_Moisture
   - Using the Capacitive Soil Moisture probe, in your notebook note the moisture readings when:
     - Empty Cup
     - Submerged in water to the notch
     - Dry Soil
     - Soil after watered
   - Display the moisture to the OLED with a Time-stamp.

The Cloud


Cloud Computing
Servers
Virtual Desktop
Software Platform
Applications
Storage / Data
OS
Apps
Router
Switch
End User

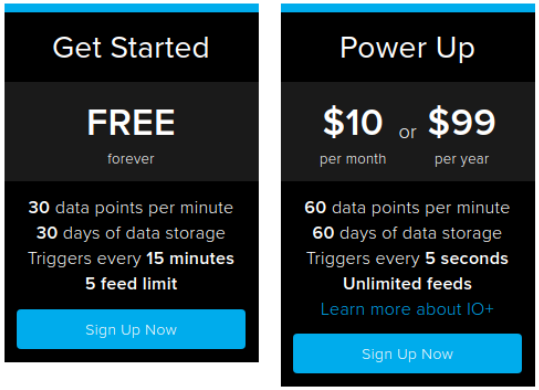
MQTT: MQ Telemetry Transport





Adafruit.io

Let's create an Adafruit.io account.



MQTT Elements explained

- TheClient - object that defines the TCP (Transmission Control Protocol) connection over WiFi.
- mqtt - object that defines the MQTT connection using the WiFi object, the MQTT server/port, and user name/password.
- FeedName - a "variable" located on Adafruit.io that can be subscribed or published to. There can be many of these.
- mqttObj - object that will be used in the C++ code that will be used to publish or subscribe to an Adafruit.io feed. There needs to be one object for each feed.
- value - Variable in the C++ code that stores information to be published or to receive information from a feed that is subscribed to.

NOTE: FeedName, mqttObj, and value should be given descriptive "names" similar to the naming convention for all variables and objects in the C++ code.

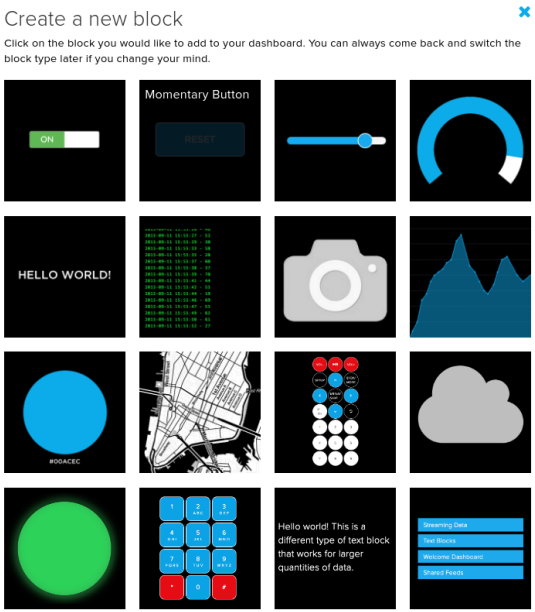
MQTT Elements in VSCode

```
#include <Adafruit_MQTT.h>

#include "Adafruit_MQTT/Adafruit_MQTT.h"
#include "Adafruit_MQTT/Adafruit_MQTT_SPARK.h"

/************ Global State (you don't need to change this!) ******************/
TCPClient TheClient;

/************************* Adafruit.io Setup *********************************/
#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "<username>"
#define AIO_KEY         "<key>"

// Setup the MQTT client class with the WiFi client, MQTT server and login details.
Adafruit_MQTT_SPARK mqtt(&TheClient,AIO_SERVER,AIO_SERVERPORT,AIO_USERNAME,AIO_KEY);

/****************************** Feeds ***************************************/
// Setup Feeds to publish or subscribe
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
Adafruit_MQTT_Publish mqttObj1 = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/
    FeedNameA");
Adafruit_MQTT_Subscribe mqttObj2 = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/
    FeedNameB");


/************Declare Variables*************/
float value1;   //variable that will hold data to be published to adafruit.io feed
float value2;   //variable that will hold data received from adafruit.io feed
```

MQTT Publish and Subscribe

```
void setup() {
  Serial.begin(9600);
  waitFor(Serial.isConnected, 15000); //wait for Serial Monitor to startup

  WiFi.connect();                     //Connect to internet, but not Particle Cloud
  while(WiFi.connecting()) {
    Serial.printf(".");
  }

  mqtt.subscribe(&mqttObj2);          // Setup MQTT subscription for FeedNameB feed.
}

void loop() {
  // Publishing to a MQTT feed
  if(mqtt.Update()) {   //if mqtt object (Adafruit.io) is available to receive data
    Serial.printf("Publishing %0.2f to Adafruit.io feed FeedNameB \n",value1);
    mqttObj1.publish(value1);
  }
    // Two new functions that will be useful:
    // atof() - ASCII to Float: converts an ASCII string to a floating point number
    // atoi() - ASCII to Integer: converts an ASCII string to an integer

  // Receive data from a subscription to an MQTT feed
  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(100))) { //wait a moment for new feed data
    if (subscription == &mqttObj2) {  // assign new data to appropriate variable
      value2 = atof((char *)mqttObj2.lastread); //value2 = data from MQTT subscription
      Serial.printf("Received %0.2f from Adafruit.io feed FeedNameB \n",value2);
    }
  }
}
```

Assignment: L14_03_SubscribePublish

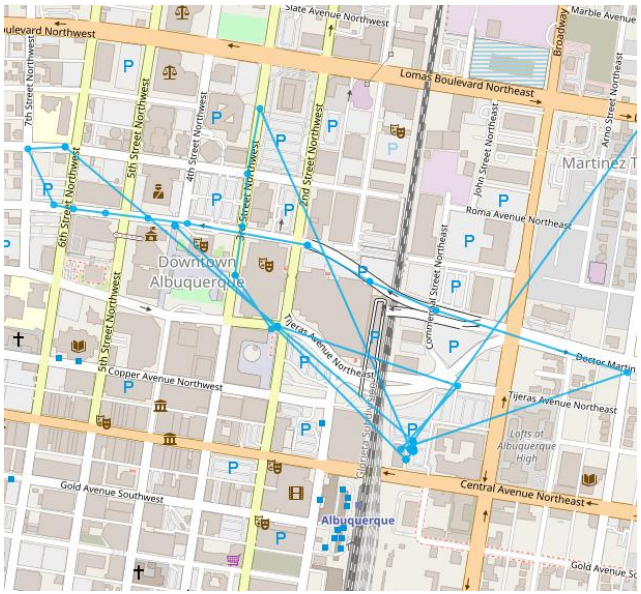


1. Modify the starter code for your Adafruit.io
2. Publish
   - Publish a random number to a feed once every 6 seconds (do not use a delay).
   - Create a line chart on your dashboard to display the random number.
3. Subscribe
   - Add a button to your Adafruit.io dashboard and connect it to a feed called buttonOnOff.
   - Subscribe to the buttonOnOff and turn on the on board LED (D7) when pressed.
4. Experiment with other blocks
   - Replace the button with a slider.
   - Control the brightness of an LED.
   - Display data with other dashboard blocks.

Local and Static Variables

```
1  const int TEMPFREQ = 10000, MOISTFREQ = 30000, MOISTPIN = A3;
2  float tempC;
3  int moist;
4  void loop() {
5    tempC = getTemp(TEMPFREQ);
6    moist = getMoisture(MOISTPIN,MOISTFREQ);
7  }
8
9  float getTemp(int timeInterval) {
10   int currentTime;
11   static int lastTime = -999999;
12   static float data;
13
14   currentTime = millis();
15   if(currentTime - lastTime > timeInterval) {
16     lastTime = millis();
17     data = BME.readTemperature();
18   }
19   return data;
20 }
21
22 int getMoisture(int probePIN, int timeInterval) {
23   static int lastTime = -999999;
24   static int data;
25
26   if(millis() - lastTime > timeInterval) {
27     lastTime = millis();
28     data = analogRead(probePIN);
29   }
30   return data;
31 }
```

Assignment: L14_03andahalf_GPSPublish

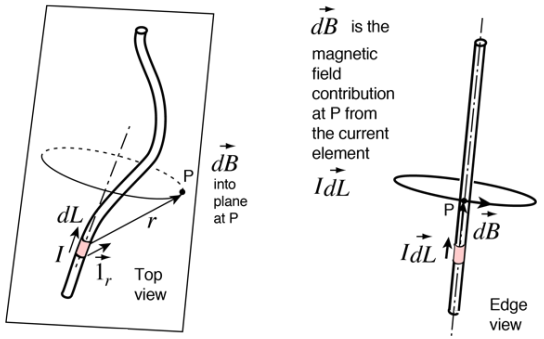


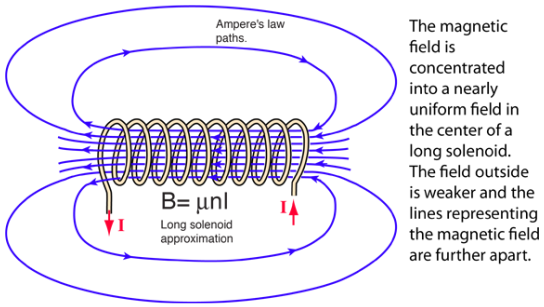
Using the data structures from lesson L12_04_HelloGPS:

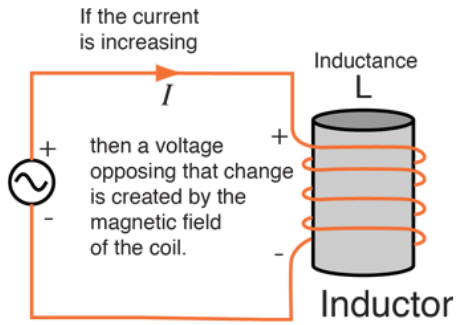
1. Every 10 seconds (without using delays), generate random GPS coordinates within Albuquerque.
2. Publish to Adafruit.io using MQTT and JSON format. Use "lat" and "lon" as the names for the JSON data elements.
3. Using the Map block to create a dashboard that shows the GPS coordinates

Midterm 2 - Plant Watering System

Inductors

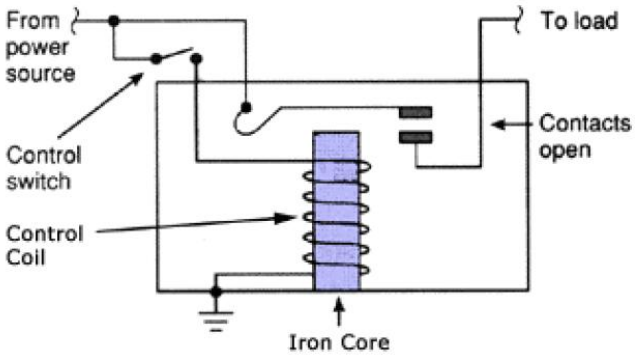


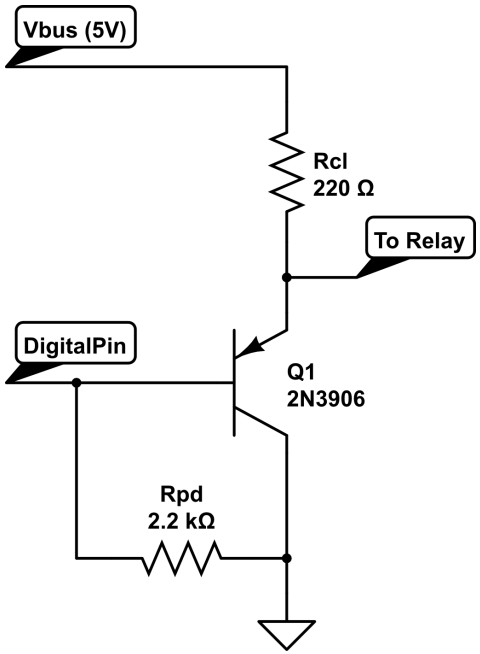




- Current flowing in a wire produces a magnetic field (B) around the wire (from Ampere's Law).
- Wire wrapped into a coil produces a magnetic field that resembles a bar magnet through the center of the coil.
- Also, in a coil, this magnetic field produces an effect known as Inductance (L) that opposes changes in electric current.

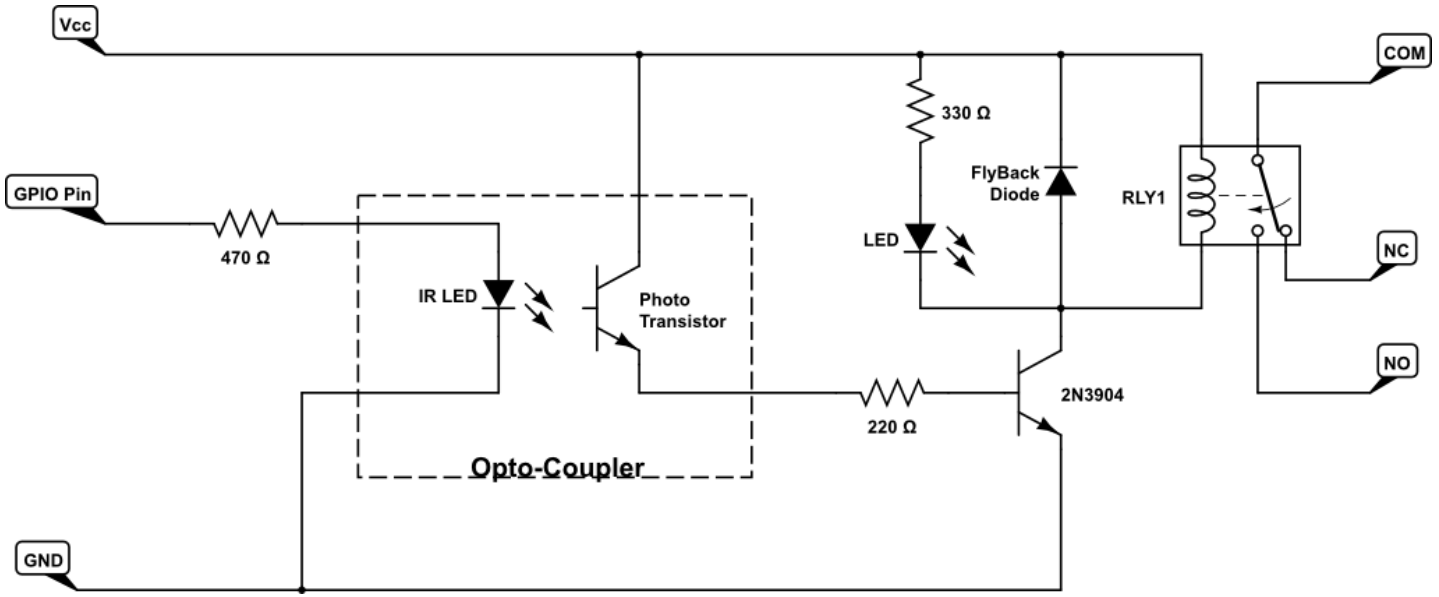
Relays

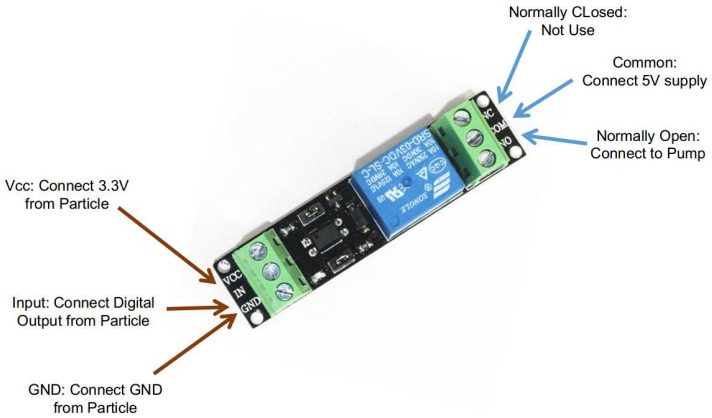




- When a device (e.g. a pump) requires higher voltage ($> 5V$) or higher current, then a relay can be used as a switch for the device
- The relay is activated by a digital pin from the microcontroller.
  - However, as the relay requires 100mA from the digital pin. To provide sufficient current, use a current amplifying emitter follower to draw current directly from the USB connection (V_{BUS}).

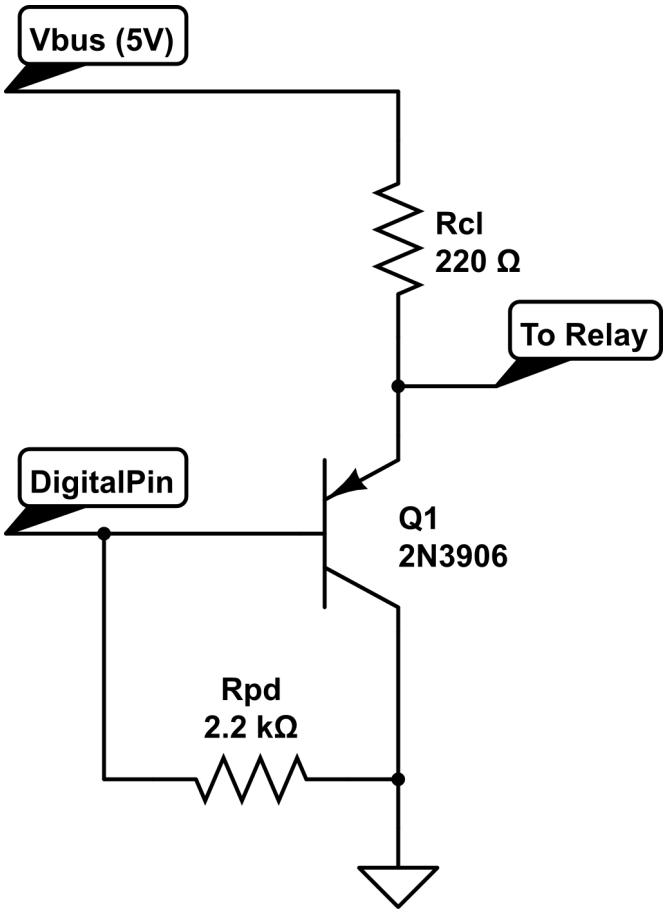
Optocoupled Relay





- Optocoupler isolates the relay load (which could be up to 240V) from the microcontroller electronics.

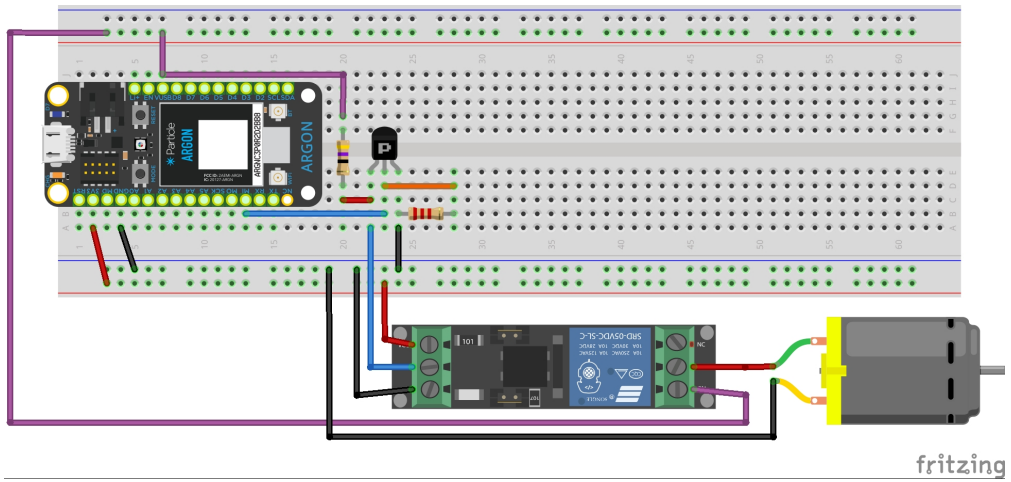
Assignment: Midterm 2



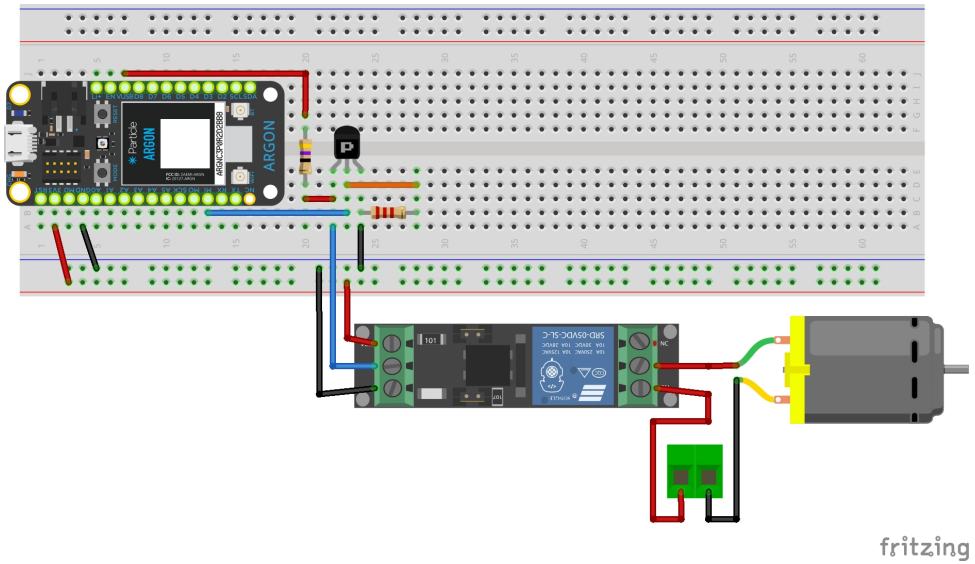
2N3906 Emitter Follower

1. Integrate a 2N3906 Emitter Follower and Relay, as well as a BME280 and Display.
2. Publish soil moisture and room environmental data to a new dashboard.
3. Automatically water your plant when the soil is too dry.
   - Only turn on the pump for a very short period of time ($\frac{1}{2} sec$).
4. Integrate a button into your dashboard that manually waters the plant.

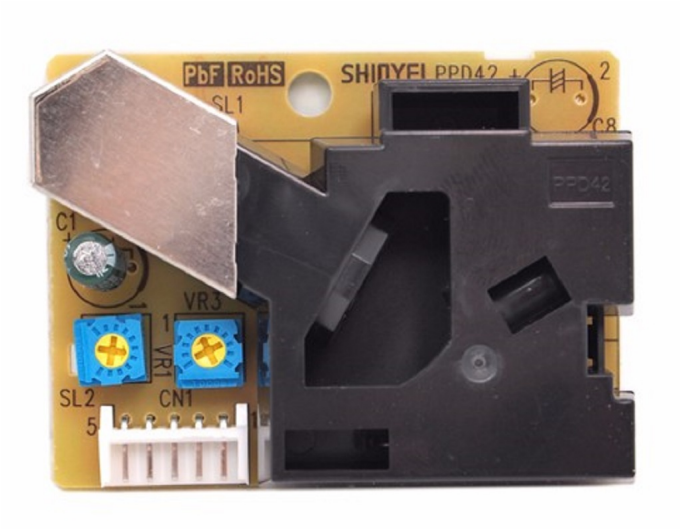
Relay and Pump Fritzing Diagram



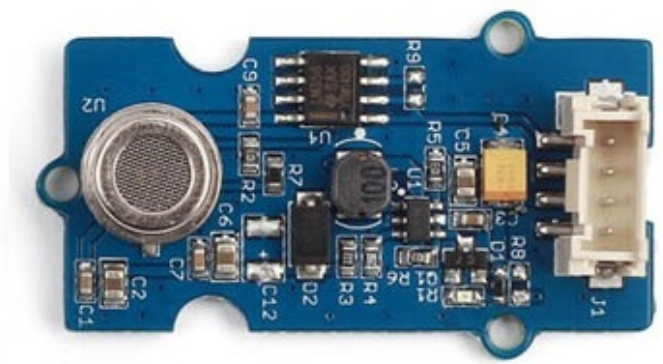
If V_{BUS} doesn't provide enough power, add external supply.



Seeed Sensors

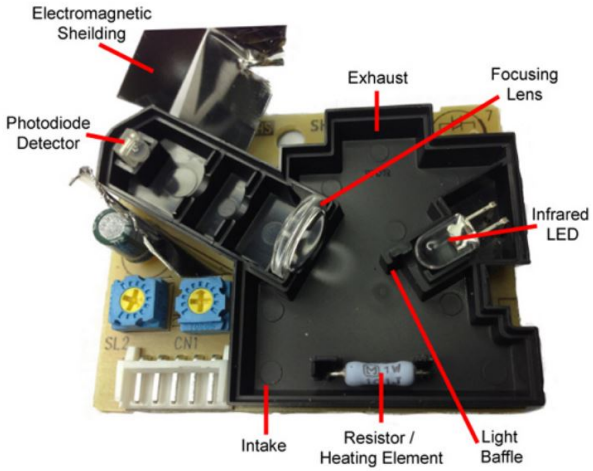


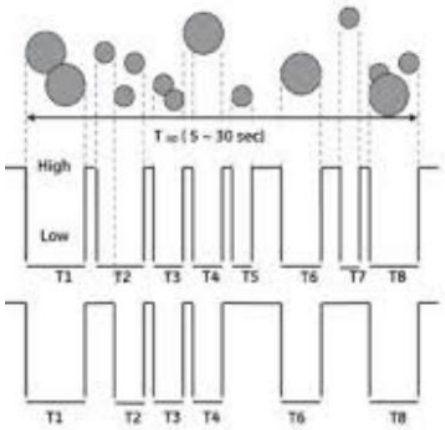
Seeed Grove - Dust Sensor

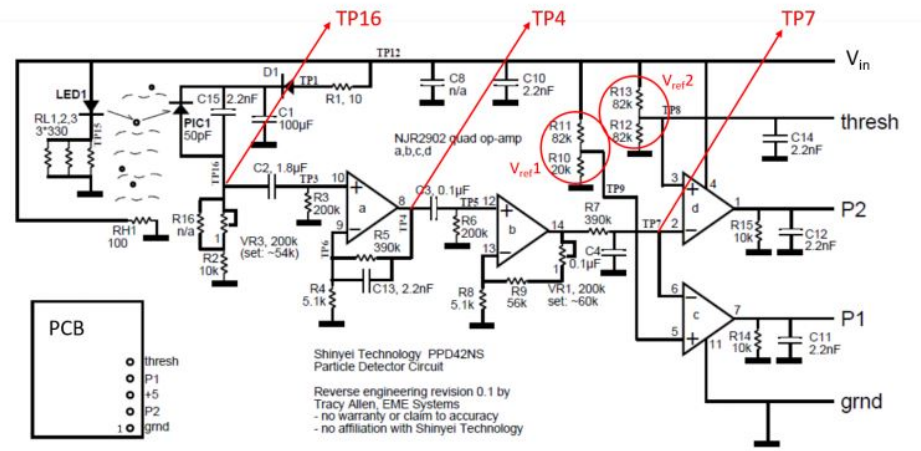


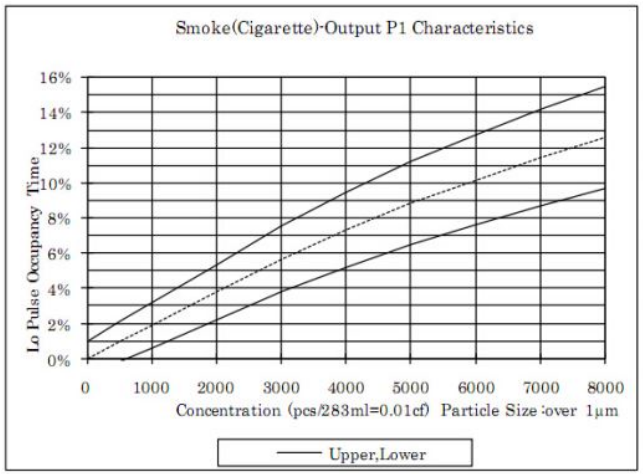
Seeed Grove - Air Quality Sensor v1.3

Shinyei PPD42NS low-cost dust sensor

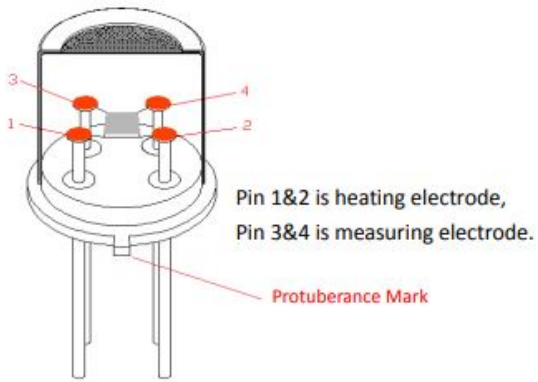


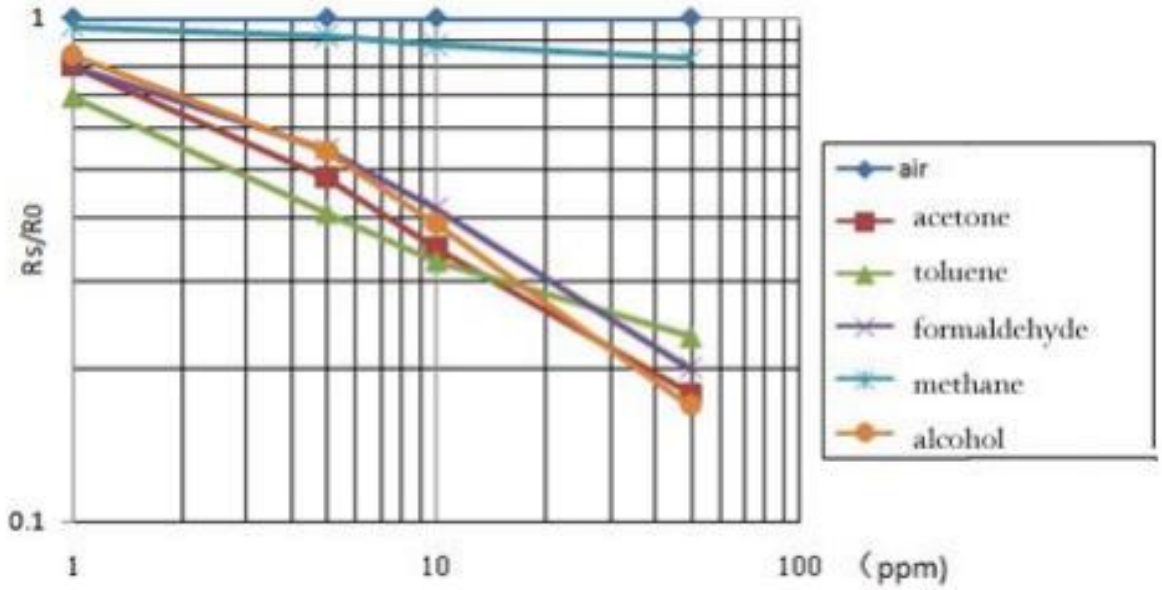






MP-503 Air Quality Sensor





Midterm 2 Continued

1. Integrate both Seeed sensors into your Plant Water project.
   - Look up the Seeed sensors online to see how they work.
   - Do not blindly copy the examples. Only use the code you need.
   - By looking at the .cpp code, determine how to get a quantitative value for air quality, in addition to the qualitative level.
2. Integrate the entire system
   - Create a user friendly Adafruit.io dashboard
   - Buy and/or create a structure to hold the plant, pump, and sensors.
   - Integration of SMS/email messaging using Zapier (following slides).
   - Video demo your Plant Watering System
3. Add to your portfolio
   - Add the project to your Hackster.io feed.
   - Create your own Github repository with a copy of your final project folder.
4. Class presentation - hackster, video demo, dashboard

Using Zapier

Zapier is an online automation tool that connects your favorite apps, such as Outlook, Slack, Mailchimp, and more. You can connect two or more apps to automate repetitive tasks.

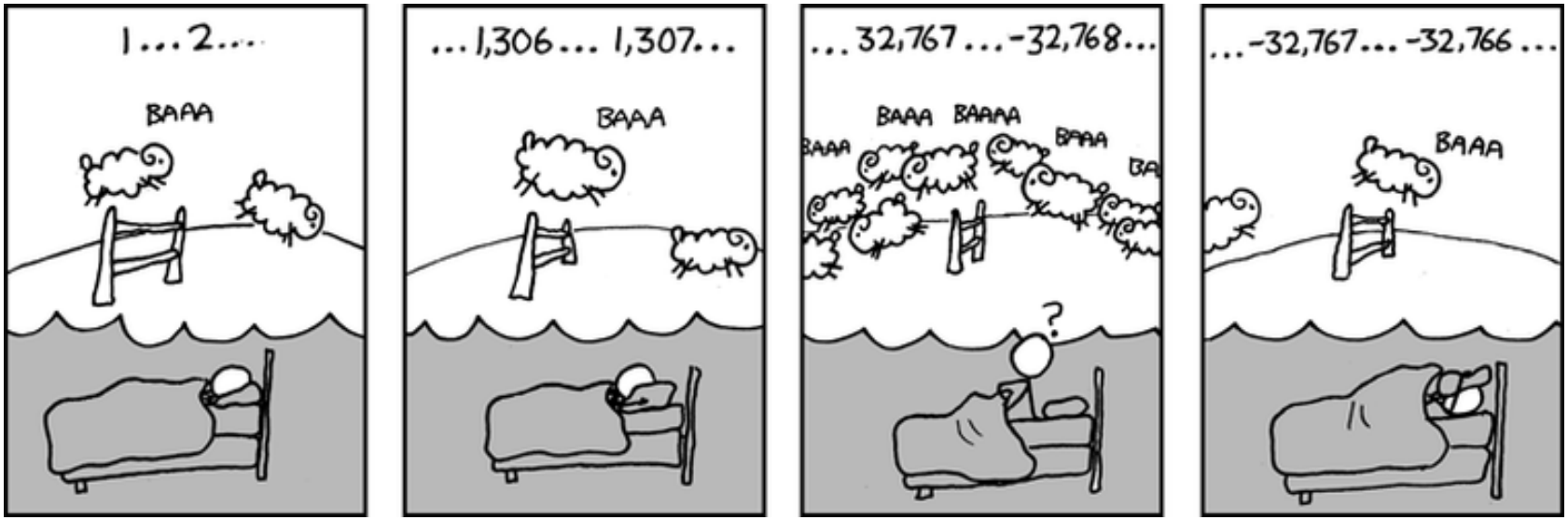
Zapier Example


```
Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(10000))) {
  if (subscription == &gotmail) {
    Serial.printf("You Got Mail \n");
    digitalWrite(D7, HIGH);
    flagServo.write(90);
    delay(10000);
    digitalWrite(D7, LOW);
    flagServo.write(5);
  }
}
```

1. Create an Zapier account using your cnm.edu email and sign up for the 14-day "Professional" trial.
2. Use the Zapier instructions in class_slides to light up your D7 LED when new mail arrives in your cnm.edu Outlook account.
3. Add to your Midterm project to send yourself an SMS text whenever your soil is too dry. In Zapier:
   - Create Feed Trigger from Adafruit.io.
   - Add a "Only Continue If...".
   - Add a Send to SMS action.

L15_Memory - Bit, Bites, and Memory

Counting Sheep



Negative Numbers

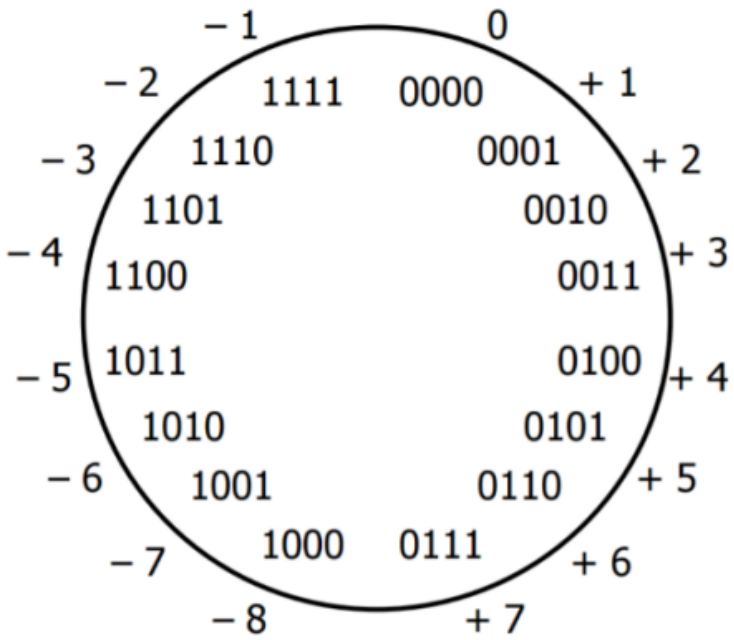
Question: How are negative numbers represented in binary?

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Answer: Left-most bit is 1 to signify negative. But wait...

2's Compliment

2's compliment is used as it makes the math consistent.



Integer		2's Complement
Signed	Unsigned	
5	5	0000 0101
4	4	0000 0100
3	3	0000 0011
2	2	0000 0010
1	1	0000 0001
0	0	0000 0000
-1	255	1111 1111
-2	254	1111 1110
-3	253	1111 1101
-4	252	1111 1100
-5	251	1111 1011

The negative plus the positive equals zero.

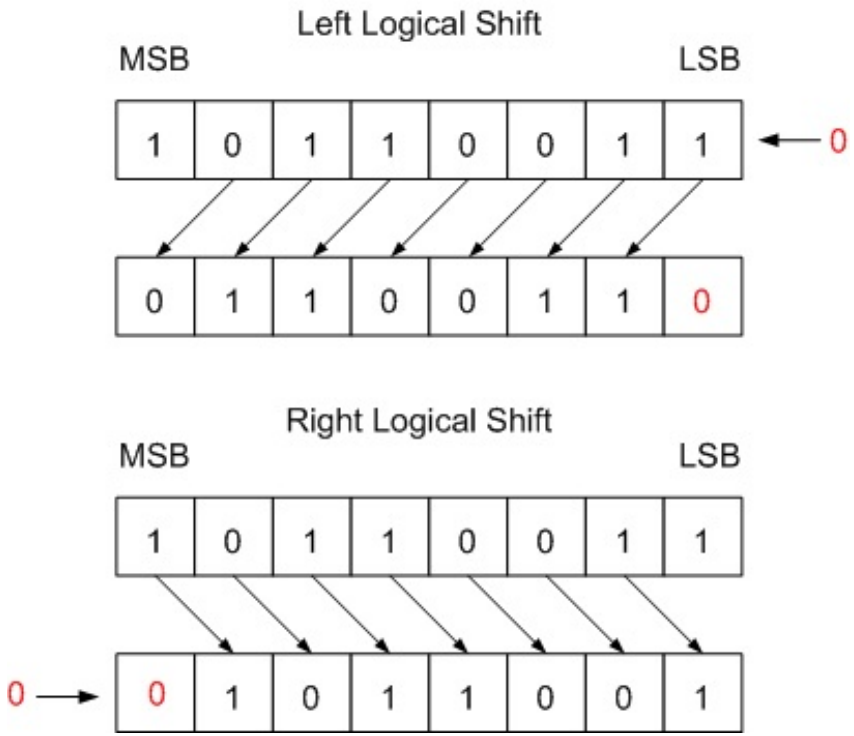
Bitwise Operations

The following table lists the Bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

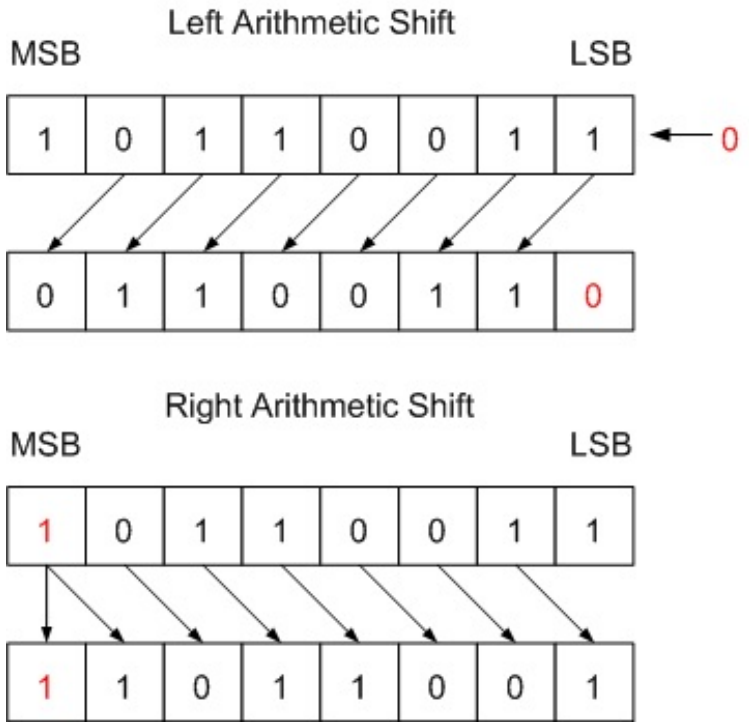
Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Bit Shifting

Logical Bit Shift



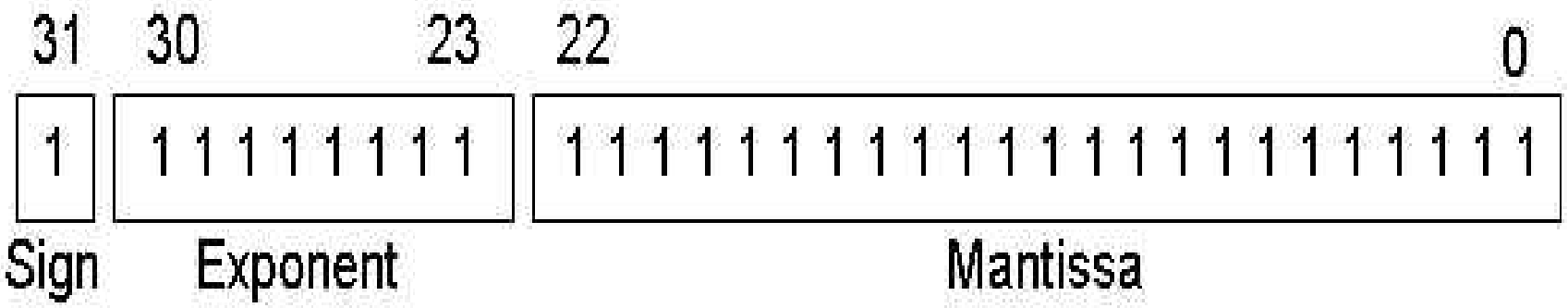
Arithmetic Bit Shift



Whether the logical or arithmetic right shift is used depends on the datatype of the variable (unsigned or signed).

BONUS: But what about Floating Point

IEEE-754 Floating Point



Example: In scientific notation: $-36382.36 = -3.638236 * 10^4$
With binary exponential equals $-1 * 1.1103014945983887 * 2^{15}$

- Sign: Negative = 1
- Exponent: 15 = 10001110 (with an exponent bias of 10000000)
- Mantissa: 1110301494598387 = 00001110000111100101110 0

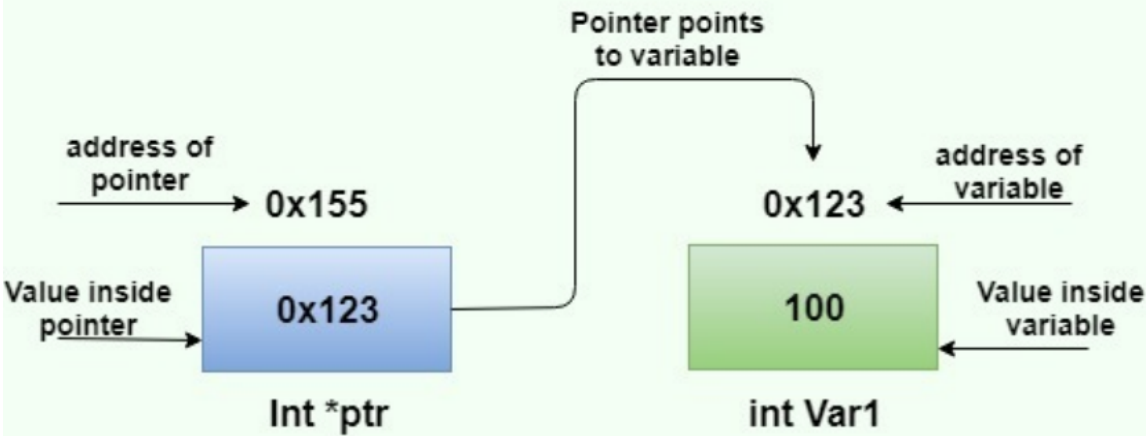
Floating point representation is: 11000111000011100001111001011100

Electrically Erasable Programmable Read Only Memory

EEPROM emulation allows small amounts of data to be stored and persisted even across reset, power down, and user and system firmware flash operations. Since the data is spread across a large number of flash sectors, flash erase-write cycle limits should not be an issue in general.

```
1  len = EEPROM.length(); //available EEPROM bytes
2  // Argons have 4096 bytes of emulated EEPROM.
3  // Addresses 0x0000 through 0x0FFF
4
5  addr = 0x00AE;    //addr between 0 and len-1
6
7  val = 0x45;
8  EEPROM.write(addr, val);
9
10 value = EEPROM.read(addr);
```

Pointers



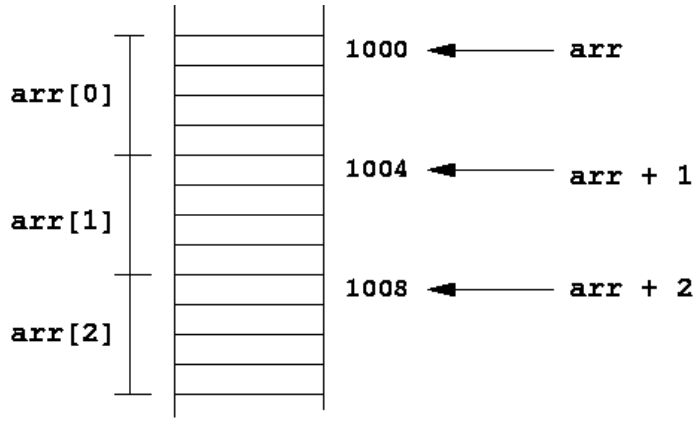
1. A pointer is a variable whose value is the address of another variable.
2. When you declare a pointer, the $*$ symbol denotes that this variable is a pointer variable. For example:
   - Pointer to an Integer: int *ptr;
3. Reference operator (&) gives the address of a variable.
4. To get the value stored in the memory address, we use the dereference operator ($*$).

Pointers

```
1  int data = 13;
2  int data2;
3  int *ptr;
4
5  void setup() {
6    Serial.begin(9600);
7    delay(1000);
8    ptr = &data;          //point ptr to the memory location of data
9    data2 = *ptr;         //set data2 to value of data (13)
10
11   // Print the Value and Address of the Variables
12   Serial.printf("Variable         Value      Address \n");
13   Serial.printf("  data            %i       0x%X   \n",data, &data);
14   Serial.printf("  ptr           0x%X       0x%X   \n",ptr, &ptr);
15   Serial.printf("  data2           %i       0x%X   \n",data2,&data2);
16 }
```

```
Serial monitor opened successfully:
Variable          Value       Address
  data              13        0x2003E380
  ptr          0x2003E380     0x2003E3F4
  data2             13        0x2003E3F0
```

Pointers and Arrays




```
1 int arr[] = {100, 200, 300};
2
3 void loop() {
4   // Compiler converts below to *(arr + 2).
5   Serial.printf("%i \n", arr[2]);
6
7   // So below also works.
8   Serial.printf("%i \n", *(arr + 2));
9 }
```

When an array (arr[]) is declared, the variable is a pointer to the first element of a continuous block of memory. In this case there are 3 elements, each 4-bytes in size, for a total of 12-bytes.

Finding Average of an Array

```
1  // This function finds the average of an array.
2  // The array is passed to it as a pointer.
3
4  float getAverage(int *array,int size) {
5    int j;
6    float total=0;
7    for(j=0;j<size;j++) {
8      total += array[j];
9    }
10   return total/size;
11 }
```

Finding Average of Arrays in Action

```
1  int xArray[4], yArray[256];
2  int *pointerX, *pointerY;
3  float average;
4  int i, sizeX, sizeY;
5
6  void setup() {
7    pointerX=&xArray[0];
8    sizeX=sizeof(xArray)/4;
9    pointerY=&yArray[0];
10   sizeY=sizeof(yArray)/4;
11   for(i=0;i<sizeX;i++) {
12     xArray[i] = random(0,255);
13   }
14   for(i=0;i<sizeY;i++) {
15     yArray[i] = random(256,512);
16   }
17   average = getAverage(pointerX, sizeX);
18   average = getAverage(pointerY, sizeY);
19 }
```

```
Array X Average = 162.50
Array Y Average = 388.35
xArray[0] value: 173, *pointerX:     173, pointerX:   0x2003E3DC
xArray[1] value: 179, *(pointerX+1): 179, pointerX+1: 0x2003E3E0
xArray[2] value: 110, *(pointerX+2): 110, pointerX+2: 0x2003E3E4
xArray[3] value: 188, *(pointerX+3): 188, pointerX+3: 0x2003E3E8
```

Returning Multiple Values from a Function

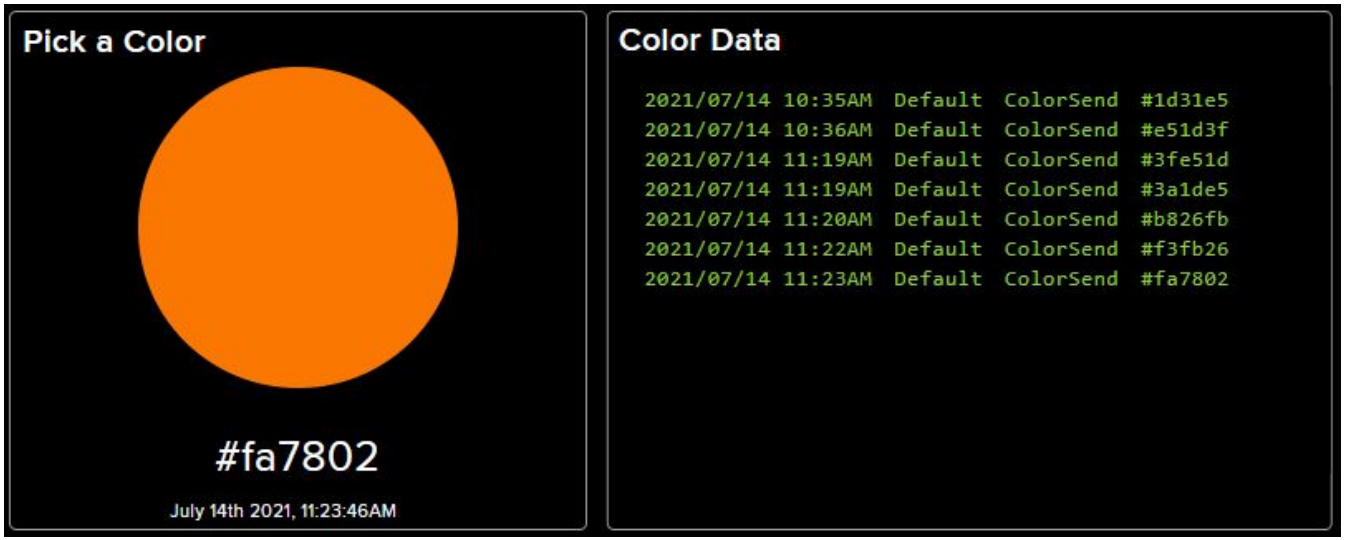
Arguments can be passed to a function by reference; thus, allowing multiple parameters to be returned by a function.

```
1  void setup() {
2    x = 4;
3    y = 2;
4  }
5
6  void loop() {
7    swap(&x, &y);
8    Serial.printf("%i%i\n",x,y);
9    delay(1000);
10 }
11
12 void swap(int *x, int *y) {
13   int temp;
14
15   temp = *x;
16   *x = *y;
17   *y = temp;
18 }
```

memcpy(), (char *), and strtol()

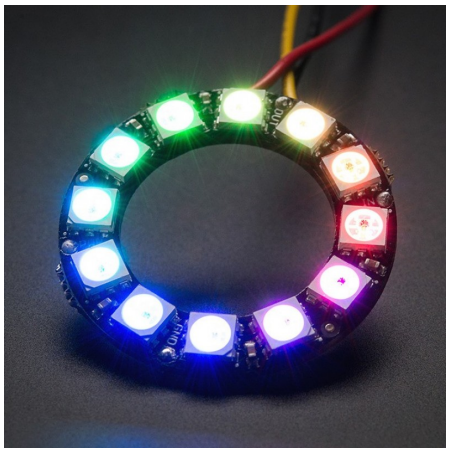
```
int color;
byte data[] = {0x23,0x42,0x41,0x34,0x32,0x35,0x44,0x39,0x35};
byte buf[6];


/* memcpy() - copy from specific memory locations to new locations
 * memcpy(to, from, size);
 *     to -> pointer to starting address of where to copy to
 *     from -> pointer to starting address of where to copy from
 *     size -> number of btyes to copy
 */

memcpy(buf, &data[1],6);      copy bytes 1 through 6 and place in buf

/* (char *) - typecasting a data type to a char-type pointer */

Serial.printf("Converting the data array to ascii symbols returns %s,\n",(char *)data);


/* strtol() - string to long - similar to atoi()
 * strtol(charString, end, base)
 *     charString -> string that contains number to be converted
 *     end -> character to end conversion on (set to NULL)
 *     base -> base of integer (16 for hex)
 */

color = strtol((char *)buf,NULL,16); // convert string to int (hex)
```

EXAMPLE: Adafruit MQTT Subscribe - Color Picker




```
int color;
byte buf[6];

  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(1000))) {
    if (subscription == &mqttColor) {
      Serial.printf("Received from Adafruit: %s \n",(char *)mqttColor.lastread);
      memcpy(buf, &mqttColor.lastread[1],6);      //strip off the '#'
      Serial.printf("Buffer: %s \n",(char *)buf);
      color = strtol((char *)buf,NULL,16);        // convert string to int (hex)
      Serial.printf("Buffer: 0x%02X \n",color);
    }
  }
```

L15_Memory Assignments



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L15_01_ColorPicker
   - Create a feed and dashboard on Adafruit.io using the Color Picker block.
   - Subscribe to your Color Picker feed and convert the lastRead() to a integer (hex)
   - Light up your NeoPixel ring the received color.
   - Using pointers create a function to convert the hex color into individual R,G,B components using Bit Shifting and AND.
   - From void loop, store the components of the color as bytes in the Argon's EEPROM.
2. L15_02_RetrieveShow
   - Retrieve the color from EEPROM memory.
   - Convert to a hex color code (for example 0xABCDEF)
   - Display the color on the NeoPixel ring using setPixelColor(n,hexColor)

Useful Properties: Identity Element

An identity element is a special type of element of a set with respect to a binary operation on that set, which leaves any element of the set unchanged when combined with it.

Addition:

$$x + 0 = x \tag{1}$$

Multiplication:

$$x * 1 = x \tag{2}$$

Bitwise AND:

$$x \& 1 = x \tag{3}$$

Bitwise OR:

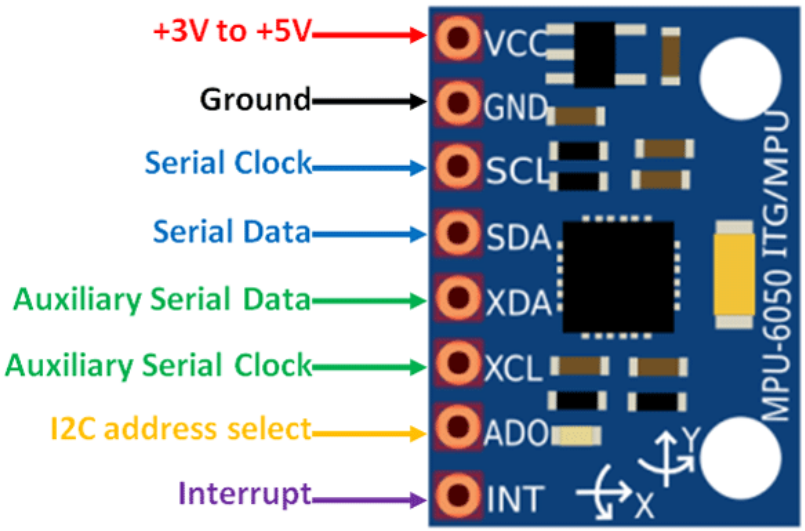
$$x | 0 = x \tag{4}$$

Added Bonus: Array of Functions via Pointers

```
1  //Array of pointers to each of the functions
2  int (* funky[4])(int x, int y) = {add,sub,mult,divi};
3
4  int a,b,answer,i;
5
6  void setup() {
7    Serial.begin(9600);
8  }
9
10 void loop() {
11   a = random(0,100);
12   b = random(0,100);
13   for(i=0;i<4;i++) {
14     answer = funky[i](a,b);
15     Serial.printf("For function %i: a = %i and b = %i equals %i \n",i,a,b,answer);
16     delay(250);
17   }
18   Serial.printf("\n\n\n");
19   delay(3000);
20 }
21
22 // The Functions
23 int add(int x,int y) {return x+y;}
24
25 int sub(int x,int y) {return x-y;}
26
27 int mult(int x,int y) (return x*y;}
28
29 int divi(int x,int y) {return x/y;}
```

L16_Motion

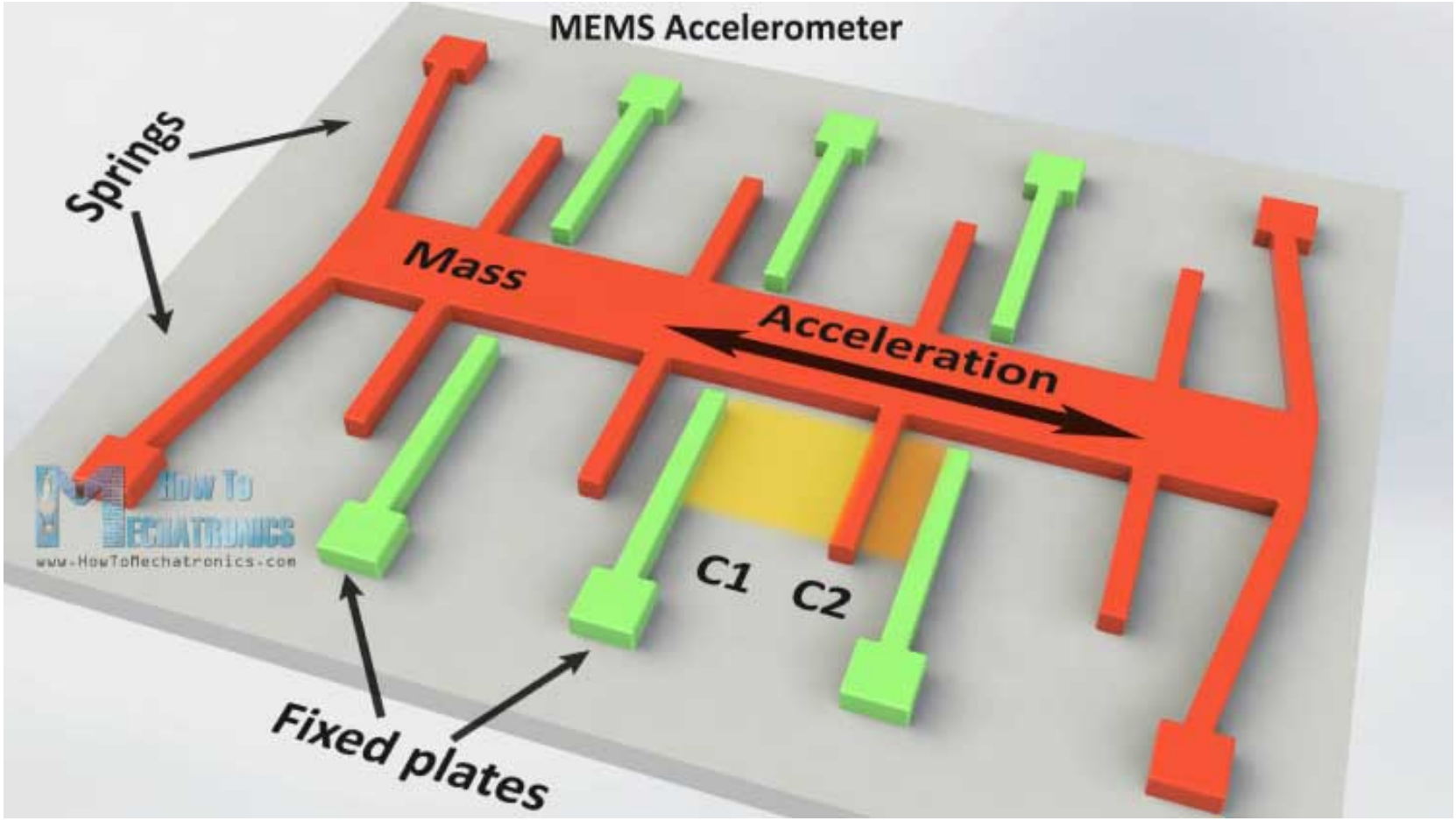
MPU6050 Accelerometer



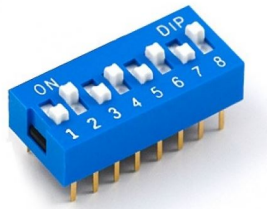
- Data Output - 16 Bit
- Gyros range: $\pm$ 250 500 1000 2000 °/s
- Accel range: $\pm$ 2 $\pm$ 4 $\pm$ 8 $\pm$ 16g

- The interrupt pin notifies the MPU about available data. To reduce power consumption, the processor can go into sleep mode and the interrupt can be used to wake up the processor.
- XDA and XCL refer to the I2C bus that the MPU-6050 controls, so it can read from slave devices such as magnetometers etc.

Accelerometers



DIP Switches and Register Maps



Remember: serial usb setup-done

Table 18: Memory map

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
hum_lsb	0xFE	hum_lsb<7:0>								0x00
hum_msb	0xFD	hum_msb<7:0>								0x80
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26..calib41	0xE1...0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00..calib25	0x88...0xA1	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Chip ID	Reset
Type:	do not change	read only	read / write	read only	read only	read only	write only

REMINDER - Data Types: Numbers

Data Type	8-bit AVR systems (Arduino Uno)			32-bit ARM systems (Teensy 3.2)		
	bytes	range (signed)	range (unsigned)	bytes	range (signed)	range (unsigned)
char	1	-128 to 127	0 to 255	1	-128 to 127	0 to 255
short	2	+/- 32,767	0 to 65,353	2	+/- 32,767	0 to 65,353
int	2	+/- 32,767	0 to 65,353	4	+/- 2,147,483,648	0 - 4,294,967,295
long	4	+/- 2,147,483,648	0 - 4,294,967,295	4	+/- 2,147,483,648	0 - 4,294,967,295
long long	8	+/- 9,223,372,036,854,770,000	0 to 18,446,744,073,709,551,615	8	+/- 9,223,372,036,854,770,000	0 to 18,446,744,073,709,551,615
float	4	3.4E +/- 38 (7 digits)	n/a	4	3.4E +/- 38 (7 digits)	n/a
double	4	3.4E +/- 38 (7 digits)	n/a	8	1.7E +/- 308 (15 digits)	n/a
long double	8	1.7E +/- 308 (15 digits)	n/a	8	1.7E +/- 308 (15 digits)	n/a
Unambiguous						
uint8_t	1	n/a	0 to 255	1	n/a	0 to 255
int8_t	1	-128 to 127	n/a	1	-128 to 127	n/a
uint16_t	2	n/a	0 to 65,353	2	n/a	0 to 65,353
int16_t	2	+/- 32,767	n/a	2	+/- 32,767	n/a
uint32_t	4	n/a	0 - 4,294,967,295	4	n/a	0 - 4,294,967,295
int32_t	4	+/- 2,147,483,648	n/a	4	+/- 2,147,483,648	n/a

There are $7.5x10^{18}$ grains of sand on Earth. A long long integer and floating point numbers are larger than this.

Initializing MPU6050

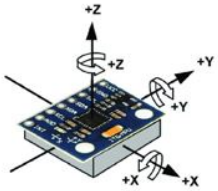
```
// Initialize the MPU in the void setup()
void setup() {
  // Begin I2C communications
  Wire.begin();

  // Begin transmission to MPU-6050
  Wire.beginTransmission(MPU_ADDR);

  // Select and write to PWR_MGMT1 register
  Wire.write(0x6B);
  Wire.write(0x00); // wakes up MPU-6050

  // End transmission and close connection
  Wire.endTransmission(true);
}
```

Reading Acceleration Data from the MPU-6050

```
// Declare variables
byte accel_x_h, accel_x_l;    //variables to store the individual btyes
int16_t accel_x;              //variable to store the x-acceleration

// Set the "pointer" to the 0x3B memory location of the MPU and wait for data
Wire.beginTransmission(MPU_ADDR);
Wire.write(0x3B); // starting with register 0x3B
Wire.endTransmission(false); // keep active.

// Request and then read 2 bytes
// Syntax:
//    Wire.requestFrom(I2C_addr, quantity, stop);
//    Wire.read(); //repeat this for each byte to be read

Wire.requestFrom(MPU_ADDR, 2, true);
accel_x_h = Wire.read(); // x accel MSB
accel_x_l = Wire.read(); // x accel LSB

accel_x = accel_x_h << 8 | accel_x_l;     // what happens if declared int instead?
Serial.printf("X-axis acceleration is %i \n",accel_x);
```

Note: the data is stored in Big Endian Byte Order. The most significant byte (the "big end") of the data is placed at the byte with the lowest address. The rest of the data is placed in the next byte.

Assignment: L16_Motion



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L16_01_MPU6050
   - Read values from the memory addresses associated with X, Y, and Z acceleration.
   - Convert the returned acceleration values to standard gravity units (e.g. when flat on the table, a_z = -1G).
2. L16_02_AutoRotate
   - Display date and time on an OLED display.
   - Use accel values to auto-rotate the OLED.
3. Extra
   - Modify L16_01_MPU6050 to be able to modify range/sensitivity with a button or encoder.

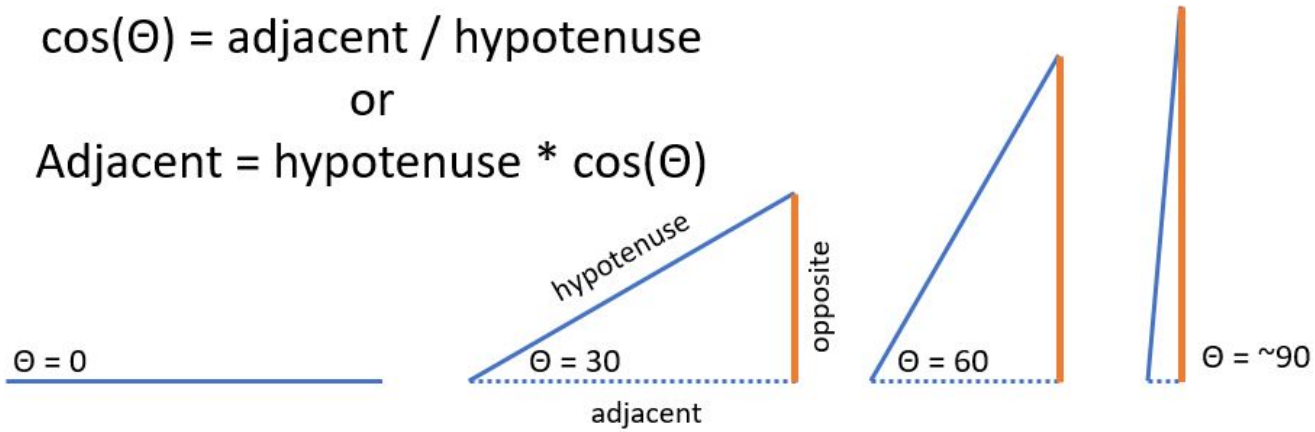
SOH CAH TOA

- sin = opposite over hypotenuse
- cos = adjacent over hypotenuse
- tan = opposite over adjacent

cos(Θ) = adjacent / hypotenuse

or

Adjacent = hypotenuse * cos(Θ)

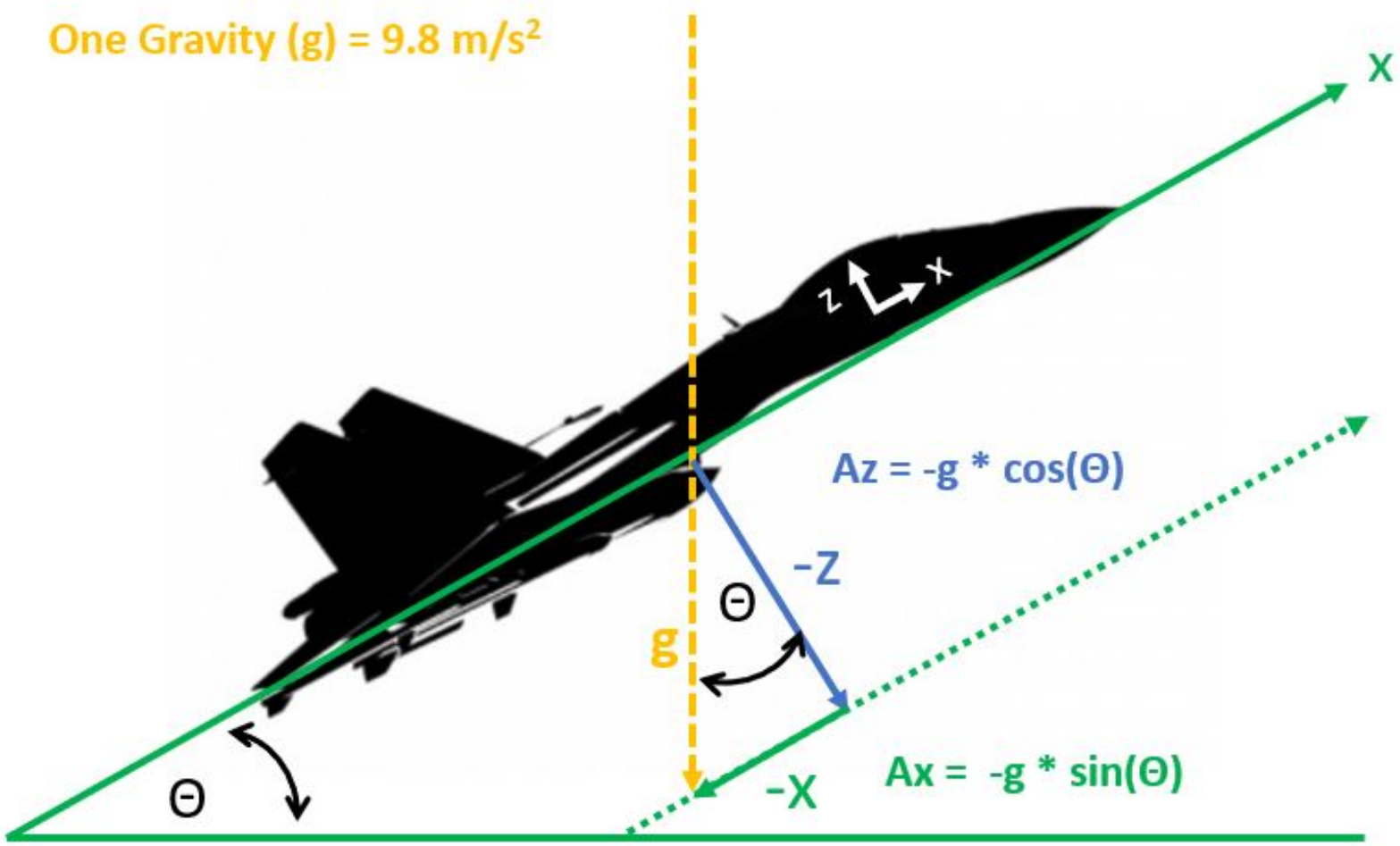


sin(Θ) = opposite / hypotenuse

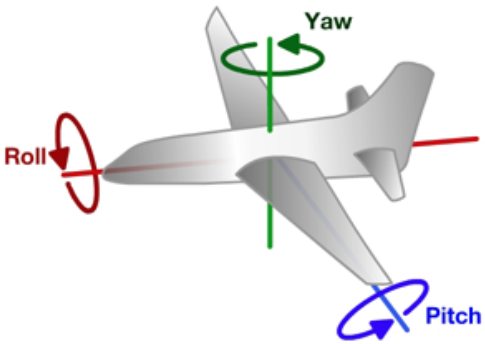
or

opposite = hypotenuse * sin(Θ)

Gravity and Orientation



Assignment: L16_Motion

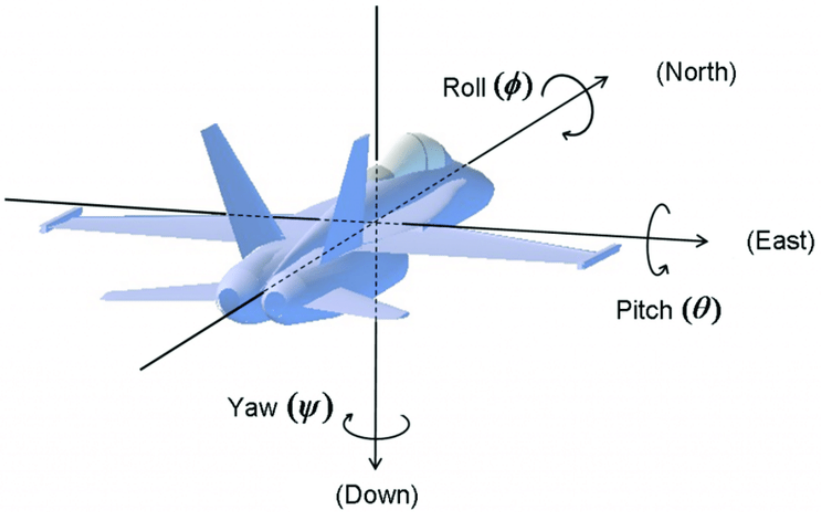


- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L16_03_Airplane
   - Calculate pitch $\theta = -asin(a_x)$.
   - Calculate roll $\phi = atan2(a_y, a_z)$.
2. L16_04_Shock
   - Store a_{tot} in an array every 10ms for 5s.
   - Find and print the max value from the array.
   - Repeat.
3. Extra
   - Improve L16_03_Airplane with equations from next slide

Recall, that trigonometric functions return radians which needs to be converted to degrees. See the Unit Circle slide in L02_HelloLED.

Pitch and Roll Improved

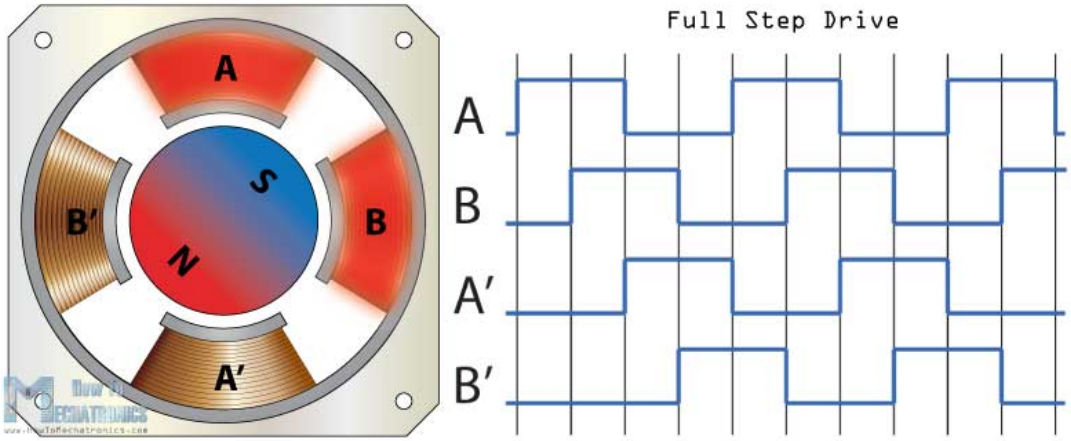


$$\text{Pitch}(\Theta) = \arctan\left(\frac{a_x}{\sqrt{{a_y}^2+{a_z}^2}}\right)$$

$$\text{Roll}\ (\Phi) = \arctan\left(\frac{a_y}{\sqrt{{a_x}^2+{a_z}^2}}\right)$$

$$\text{Yaw}\ (\Psi) = \arctan\left(\frac{\sqrt{{a_x}^2+{a_y}^2}}{{a_z}^2}\right)$$

Stepper Motors



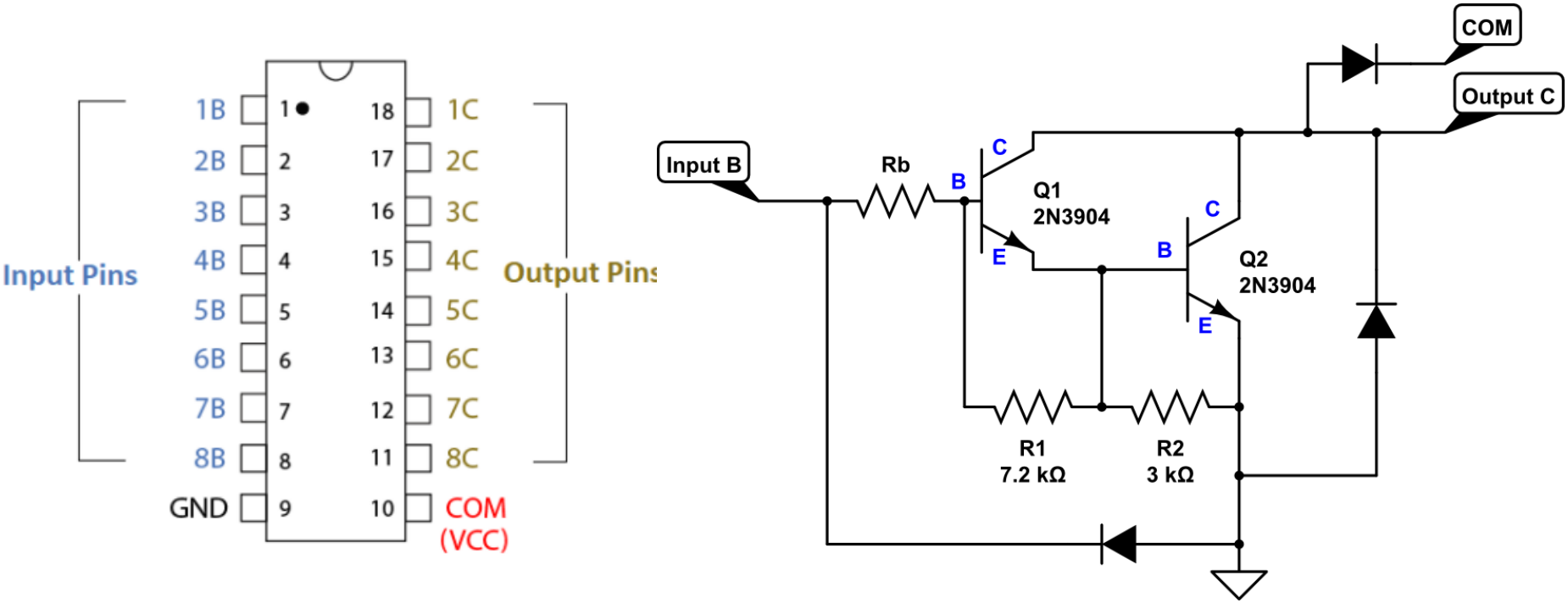
28BYJ Stepper Motor

- 2048 steps per revolution
  - 32 steps per rotor revolution
  - Gear ratio 1:64
- Capable of 10-15 RPM (at 5V)
- ULN2003 Darlington Array driver

Stepper.h

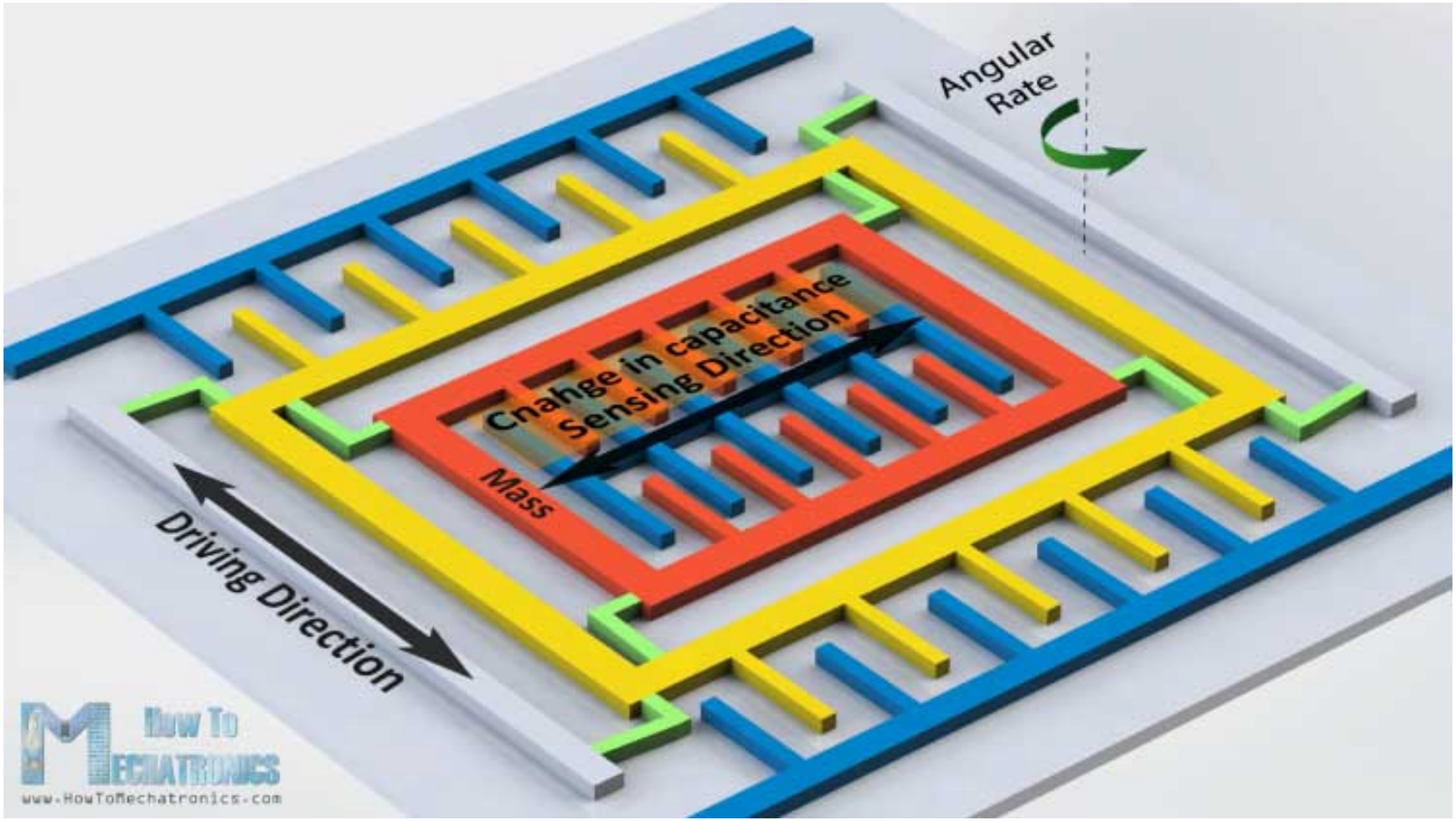
- Stepper myStepper (spr,IN1,IN3,IN2,IN4)
- myStepper.setSpeed(speed)
- myStepper.step(steps)

ULN2003 Darlington Array



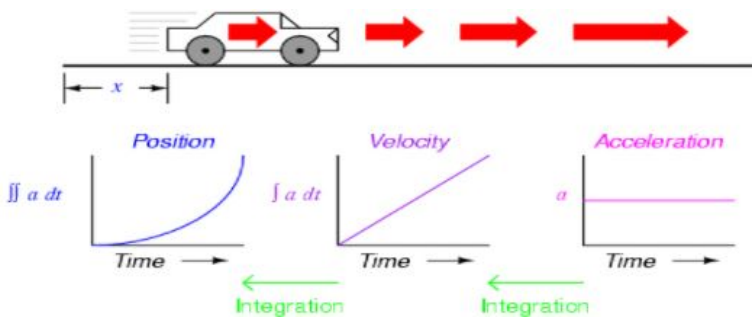
A Darlington Array is a set of current amplifying circuits that take outputs from the microcontroller and boost the current used to drive the motor.

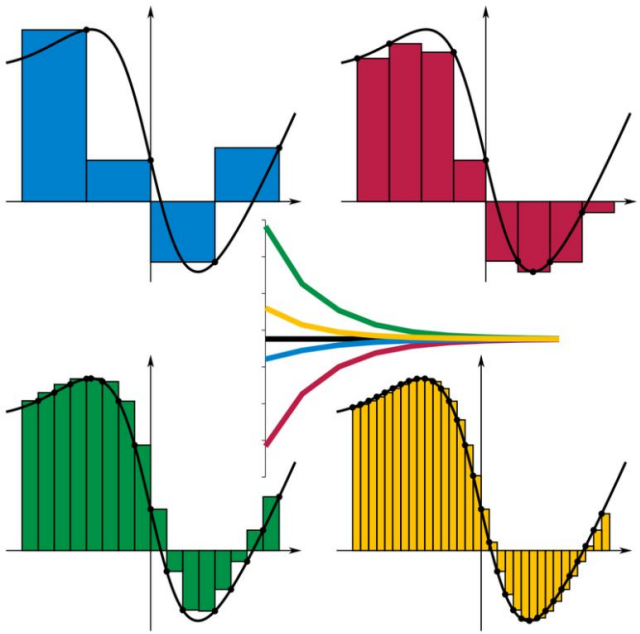
Gyroscopes



The gyroscope measures angular velocity (degrees per second).

Acceleration, Velocity, and Position



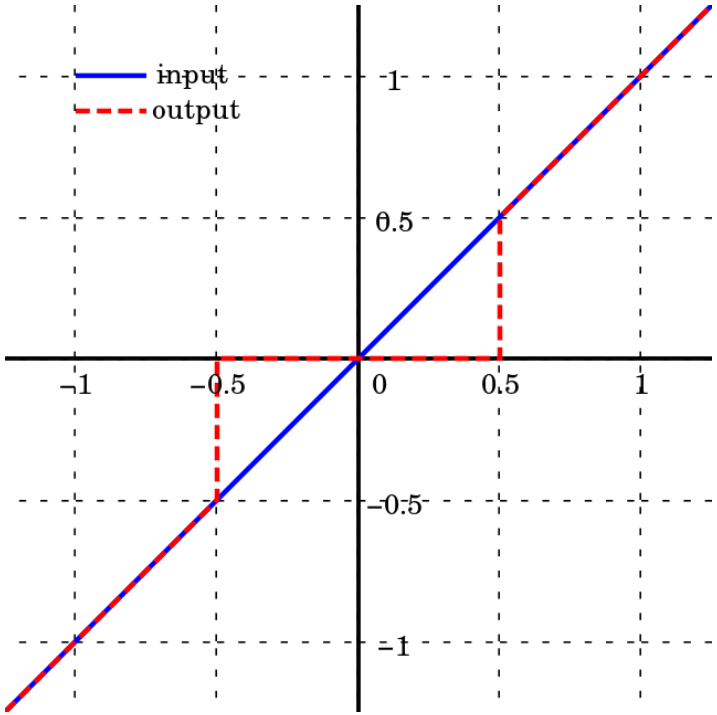


The Reimann Sum method can be used to "integrate" acceleration to velocity and velocity to position.

- Gyroscope output is angular velocity: ω ($\frac{degrees}{second}$)
- To get change in angular position ($\Delta\theta$), multiple each angular velocity by the time step: $\Delta\theta = \omega * \Delta t$
- Use the Reimann Sum to get the resulting angular position

$$\theta = \sum_{t=0}^{t=T} (\omega * \Delta t)$$

Deadband



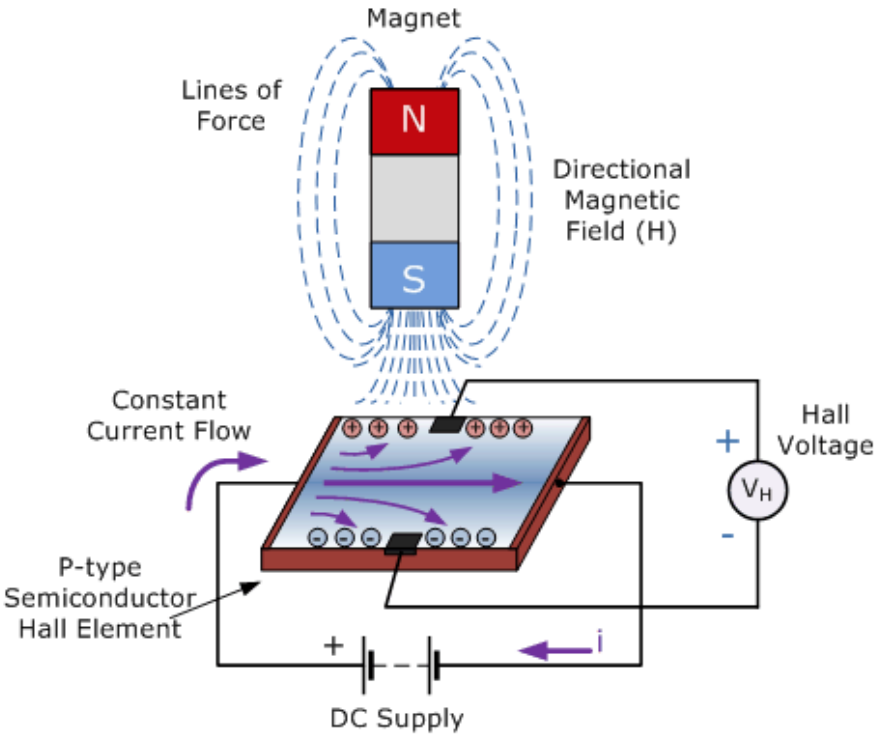
A deadband is a band of input values that in a control system create an output that is zero.

Assignment: L16_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L16_05_Stepper
   - Make a gear/pointer to show motor position (cardboard, laser wood, 3D print)
   - Wire the stepper motor noting the order: IN1, IN3, IN2, IN4.
   - Move the motor 2 rotations clockwise, pause, 1 rotation counter-clockwise, repeat.
2. L16_06_DriveByWire
   - Connect the MPU-6050 to your system.
   - Obtain the z-axis rotation from the appropriate register on the MPU-6050. Convert to angular rotation (°per sec).
   - Calculate the angular position (°) of the gyroscope.
   - Have the stepper motor track movement in the gyroscope.

Hall Effect Sensor



Discovered by Edwin Hall in 1879, the Hall Effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and to an applied magnetic field perpendicular to the current.

Interrupts

Interrupts are a way to write code that is run when an external event occurs. As a general rule, interrupt code should be very fast, and non-blocking. This means performing transfers, such as I2C, Serial, TCP should not be done as part of the interrupt handler. Rather, the interrupt handler can set a variable which instructs the main loop that the event has occurred.

```
1 pinMode(pin, INPUT); \\ can also use INPUT_PULLUP or INPUT_PULLDOWN
2 attachInterrupt(pin, function, mode);
```

Mode: defines when the interrupt should be triggered. Three constants are predefined as valid values:

- CHANGE to trigger the interrupt whenever the pin changes value,
- RISING to trigger when the pin value goes from low to high,
- FALLING for when the pin value goes from high to low.

Software Timers

There are also software timer interrupts. The Argon can manage up to 10 timers simultaneously.

```
1  Timer timer(1000,printEverySecond);
2
3  void setup() {
4    Serial.begin(9600);
5    timer.start();
6  }
7
8  void printEverySecond() {
9      static int count = 0;
10     Serial.printf("count=%i, time = %u ms \n", count,millis());
11     count++;
12 }
```

Note:

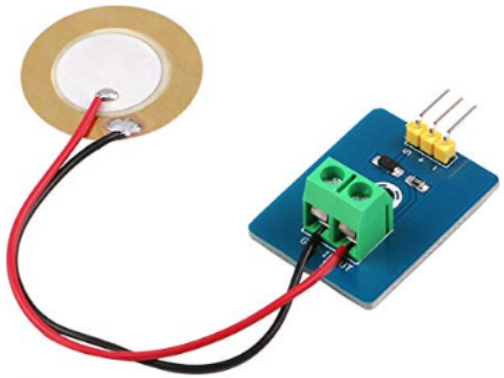
- The timer callback is similar to an interrupt - it shouldn't block.
- Multiple timers are serviced sequentially when several timers trigger simultaneously, thus requiring special consideration when writing callback functions.

Assignment: L16_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

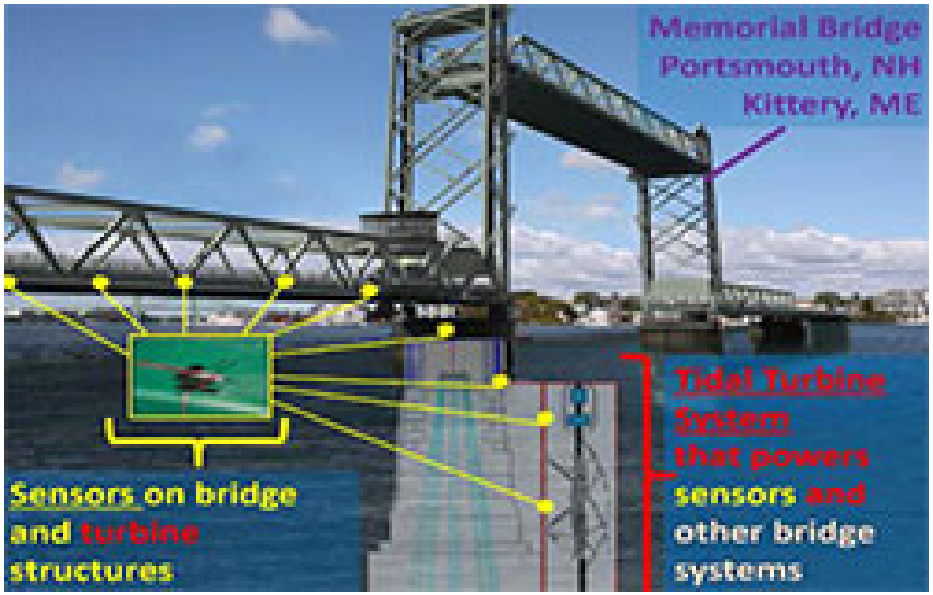
1. L16_07_Alarm
   - Connect Hall Effect Sensor, button, and Neopixel to simulate an alarm system.
   - Use the button to enable / disable alarm.
   - When armed:
     - Neopixel is GREEN when magnet detected.
     - Blinking RED when magnet not detected.
2. L16_08_RPM
   - Place magnet on shaft of lathe.
   - Create an interrupt function that returns the time per rotation using the Hall Effect Sensor.
   - Convert this time to rotations per minute.
   - Display on Adafruit.io databoard.
   - EXTRA:
     - Create a speedometer using a servo motor. (The Servo class natively available).
     - Measure RPM on other equipment at FUSE.

Piezoelectric Elements



- The piezoelectric effect is the appearance of electrical potential (voltage) across the side of a crystal when subject to mechanical stress.
- Conversely, a crystal becomes mechanically stressed (deformed in shape) when a voltage is applied across opposite faces.
- By utilizing an analogRead(), the vibration (change in mechanical stress) can be monitored over time.

Structural Engineering Sensors



FAT File System - SDCard Argon Project

Feature	FAT32	NTFS
Maximum Partition Size	2TB	2TB
Maximum File Size	4GB	16TB
Maximum File Name	8.3 Characters	255 Characters
File/Folder Encryption	No	Yes
Fault Tolerance	No	Auto Repair
Security	Network Only	Local and Network
Compression	No	Yes
Compatibility	Win 95/98/2000/XP and the derivations	Win NT/2000/XP/Vista/7 and the later versions

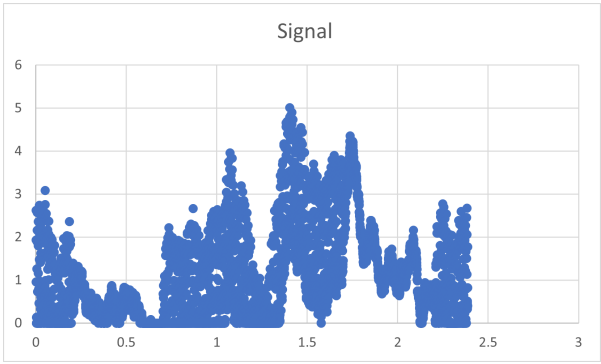
The FAT (File Allocation Table) file system, originally designed in 1977 for floppy disks, is simple and robust. It offers good performance in very light-weight implementations, but does not deliver performance, reliability and scalability afforded by modern file systems (such as NTFS or exFAT).

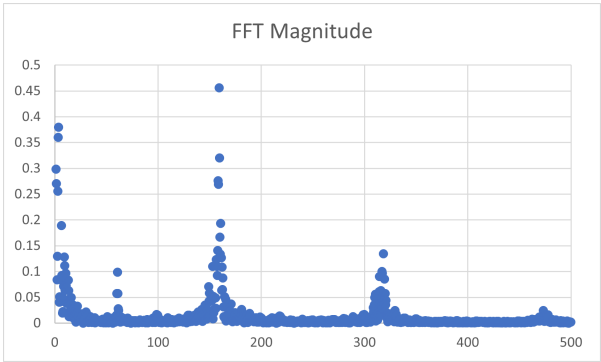
Note: FAT file name follows a 8.3 format (e.g., ABCDEFGH.txt). We will be appending two digits, so our base name can be up to 6 characters (e.g. FILE_BASE_NAME = "mydata" → mydata42.csv).

Galaxy S9+ Vibration Analysis

Using the FFT Tutorial in Class_Slides

	A	B	C	D	E
1	TimeStamp	Signal	Frequency	FFT Magnitude	Complex FFT
2	0.00061	2.62	0.4209321	2.000009766	4096.02
3	0.00119	1.93	0.8418642	0.836477508	-552.77950380473+1621.47055713326i
4	0.00177	0.96	1.2627963	0.298364661	-414.982558995766-448.522671528173i
5	0.00235	0.13	1.6837284	0.270681692	353.443826952842-427.069259698417i
6	0.00293	0	2.1046606	0.083873335	-72.9068609990069+155.532672030055i
7	0.003509	0	2.5255927	0.129856545	252.456289478309+83.62538763186061
8	0.004089	0	2.9465248	0.255636434	-516.88842919695+83.210943362584i
9	0.004669	0	3.3674569	0.360085774	423.28074046682-603.882664071708i
10	0.005249	0.14	3.788389	0.379410535	88.80273300538+771.941714466294i
11	0.005829	1.26	4.2093211	0.04066392	-41.3210095359081-72.3054897410451i
12	0.006409	2.16	4.6302532	0.051383144	94.6397062012862-46.0135075977235i
13	0.006988	2.57	5.0511853	0.084507321	25.9352596801745-171.116717569232i
14	0.007568	1.89	5.4721175	0.041174283	36.0724144460193-76.2199128809511i
15	0.008148	0.73	5.8930496	0.04976769	-73.9953870941929-70.0944447984849i





The Galaxy S9 vibrates at 159.11 Hz

Assignment: L16_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

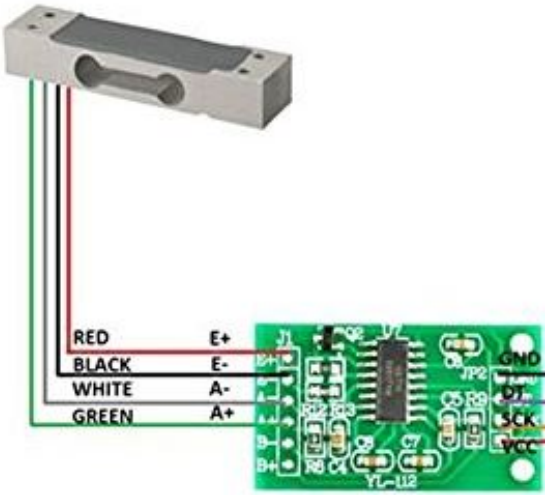
1. L16_09_Vibration
   - Connect the piezo sensor, a button, and a μSD module to your Argon.
   - Each time button is pressed, execute a loop 4096 times:
     - Every 500μsec, collect piezoelectric data (without using a delay).
     - Save the piezo data and a timestamp (converting micros() to seconds) to a 2-dimensional array.
   - When the loop is complete, write the timestamp and data to a file.
   - Collect vibration data from the lathe, cell phone vibration, other machines at FUSE.
   - Use Excel and the FFT Tutorial (class_slides) to resample graph data in frequency domain (This process will be reviewed as a class.).

Assignment: L16_Motion EXTRA

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

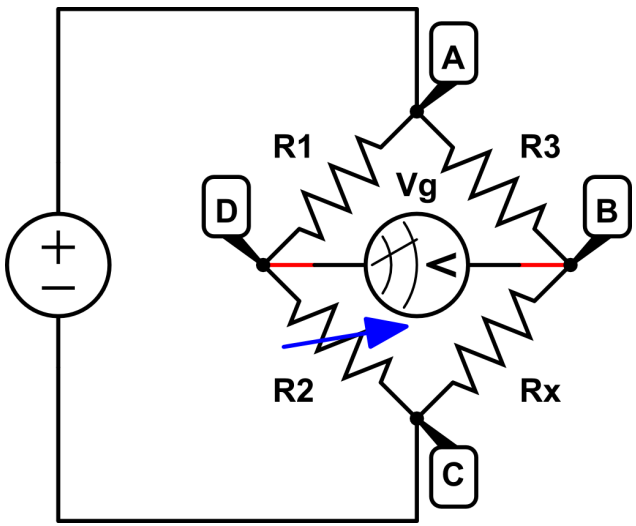
1. L16_09_Vibration
   - Borrow a microphone.
   - Install in place of the piezo sensor.
   - Use Physics Toolbox Sensor Suite - Tone Generator to create a tone of a specific frequency.
   - Record/save with the Argon, and create an FFT

Load Cells



1. A load cell is a force transducer. It converts a force such as tension, compression, pressure, or torque into an electrical signal that can be measured and standardized. As the force applied to the load cell increases, the electrical signal changes proportionally. The most common types of load cells used are hydraulic, pneumatic, and strain gauge.
2. The HX711 module is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor.

Wheatstone Bridge

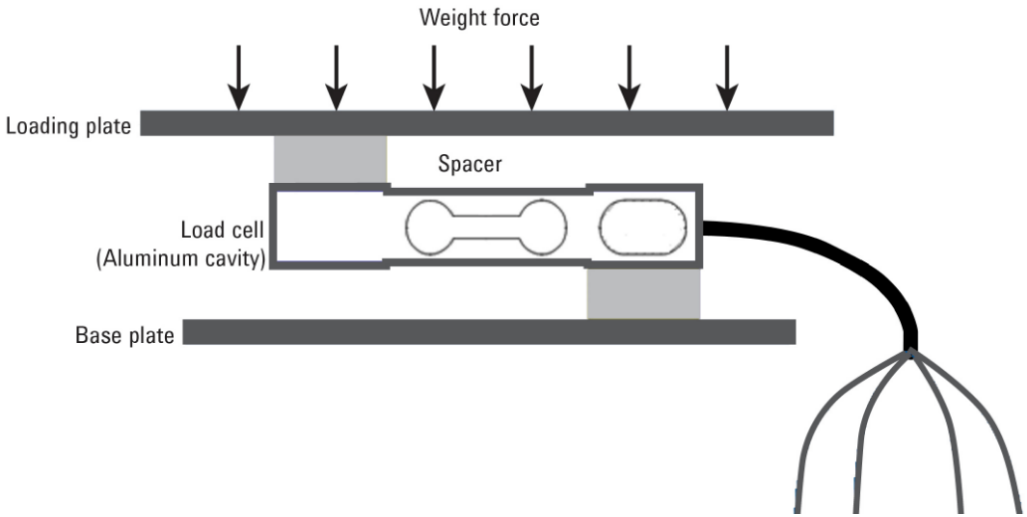


1. The Wheatstone bridge was invented by Samuel Hunter Christie in 1833 and improved and popularized by Sir Charles Wheatstone in 1843.
2. A Wheatstone bridge is an electrical circuit used to measure an unknown electrical resistance by balancing two legs of a bridge circuit, one leg of which includes the unknown component.
3. The primary benefit of the circuit is its ability to provide extremely accurate measurements (in contrast with something like a simple voltage divider).

HX711A Library

```
// From the Command Palette install the HX711A library, that will give you HX711.h
#include "HX711.h"
HX711 myScale(DT,CLK);     // any two digital pins

const int CAL_FACTOR=1000; //changing value changes get_units units (lb, g, ton, etc.)
const int SAMPLES=10; //number of data points averaged when using get_units or get_value

float weight, rawData, calibration;
int offset;

void setup() {
  myScale.set_scale();            // initialize loadcell
  delay(5000);                    // let the loadcell settle
  myScale.tare();                 // set the tare weight (or zero)
  myScale.set_scale(CAL_FACTOR); //adjust when calibrating scale to desired units
}

void loop() {
  // Using data from loadcell
  weight = myScale.get_units(SAMPLES); // return weight in units set by set_scale();
  delay(5000)                          // add a short wait between readings

  // Other useful HX711 methods
  rawData = myScale.get_value(SAMPLES);// returns raw loadcell reading minus offset
  offset = myScale.get_offset();       // returns the offset set by tare();
  calibration = myScale.get_scale();   // returns the cal_factor used by set_scale();
}
```

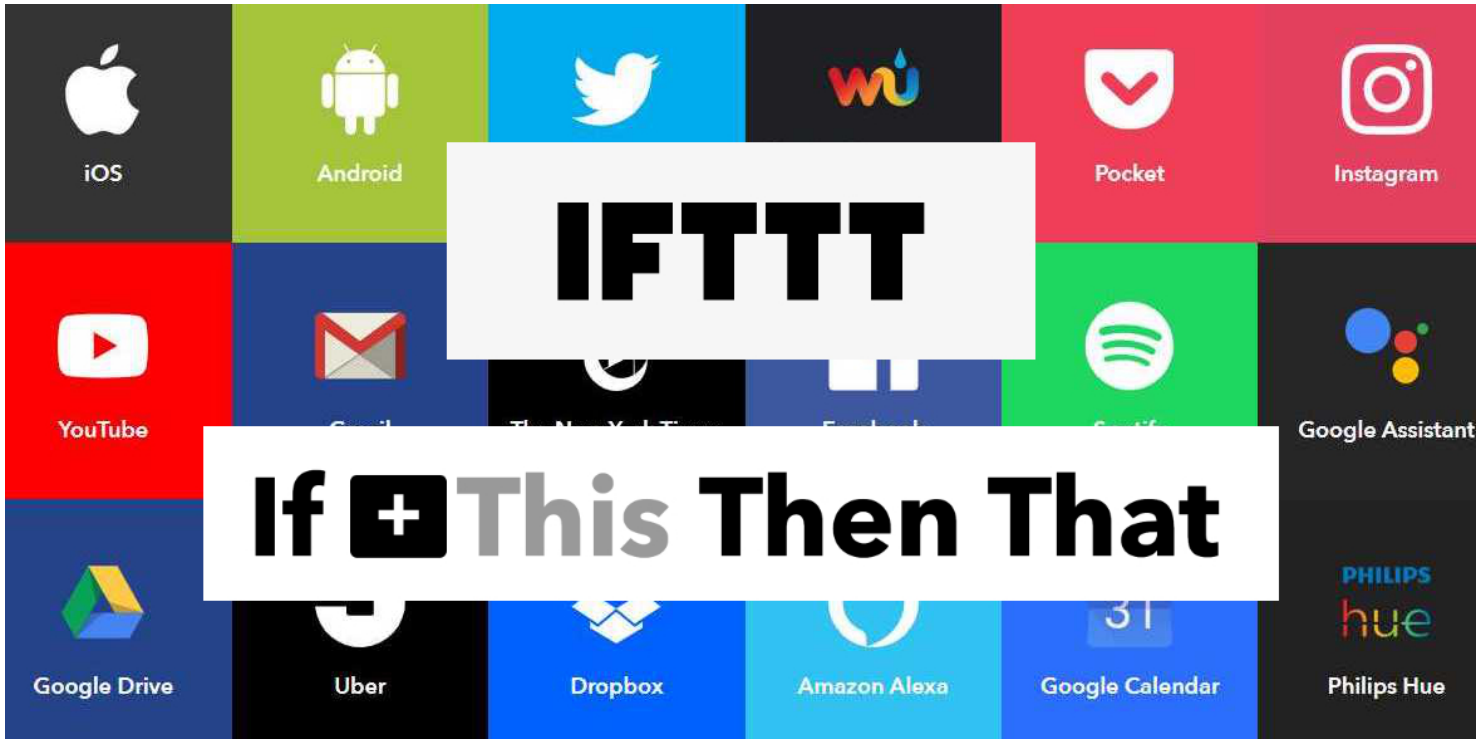
Assignment: L16_Motion (Learn to Calibrate)



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

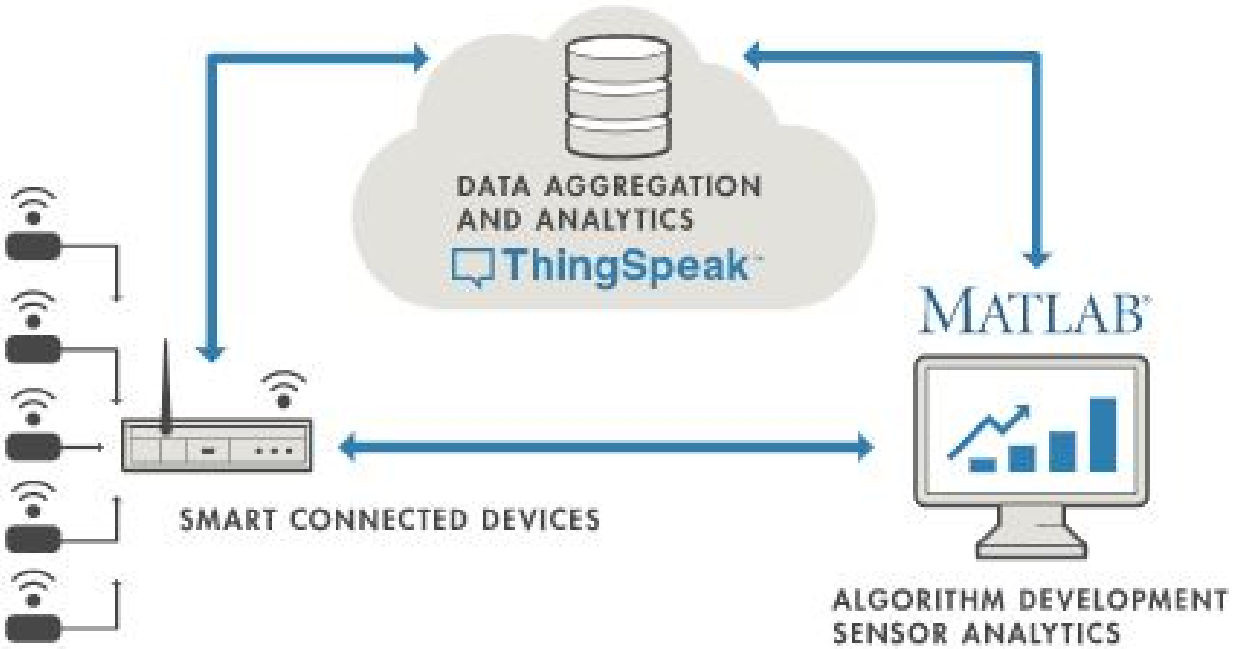
1. L16_10_Scale
   - Set initial CAL_FACTOR to 1000 and measure a known weight. (Note: one cup of water (in paper cup) is approx. 244 g).
   - Adjust CAL_FACTOR until you get the expected measurement in grams.
   - Post data to Adafruit.io and/or ThingSpeak™
   - Optional: Send text via IFTTT.

IFTTT



If This Then That, also known as IFTTT, is a freeware web-based service that creates chains of simple conditional statements, called applets.
An applet is triggered by changes that occur within other web services such as Gmail, Facebook, Telegram, Instagram, or Pinterest.

ThingSpeak Dashboard and Matlab Analysis



ThingSpeak™is an IoT analytics platform service from MathWorks®, the makers of MATLAB® and Simulink®. ThingSpeak allows you to aggregate, visualize, and analyze live data streams in the cloud.

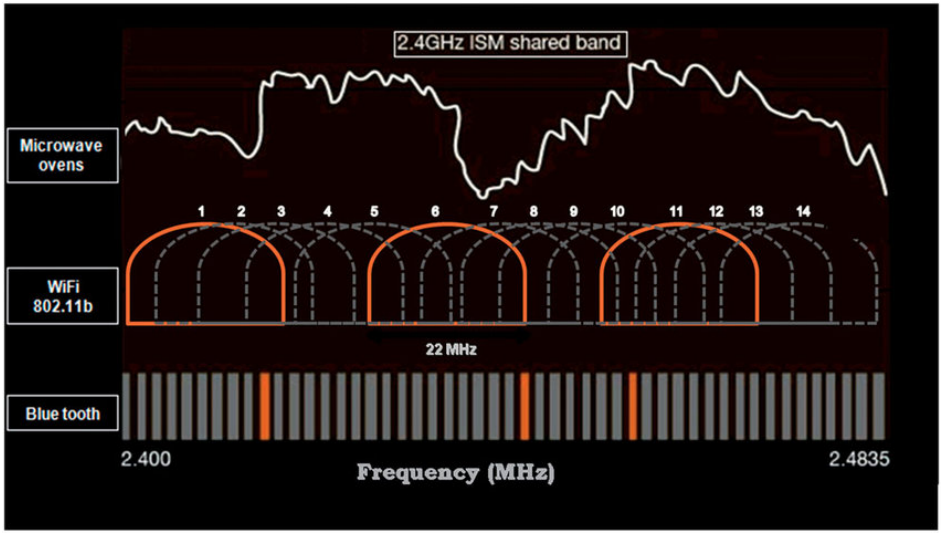
ThingSpeak.h

```
#include "ThingSpeak.h"
TCPClient client;

// Change the below to your channel number and APIKeys
unsigned int myChannelNumber = 31461;
const char * myWriteAPIKey = "LD79EOAAWRVYF04Y";
const char * myReadAPIKey = "NKX4Z5JGO4M5I18A";

void setup() {
  ThingSpeak.begin(client);
}

void loop() {

  float voltage = ThingSpeak.readFloatField(myChannelNumber, 1, myReadAPIKey);

  pinVoltage = analogRead(A1) * (3.3 / 4095.0);
  ThingSpeak.setField(1,pinVoltage);

  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

}
```

Information on ThingSpeak channels can be found at: `https://community.thingspeak.com/tutorials/thingspeak-channels/`

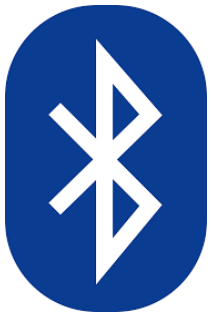
L17_DataXfer

Bluetooth



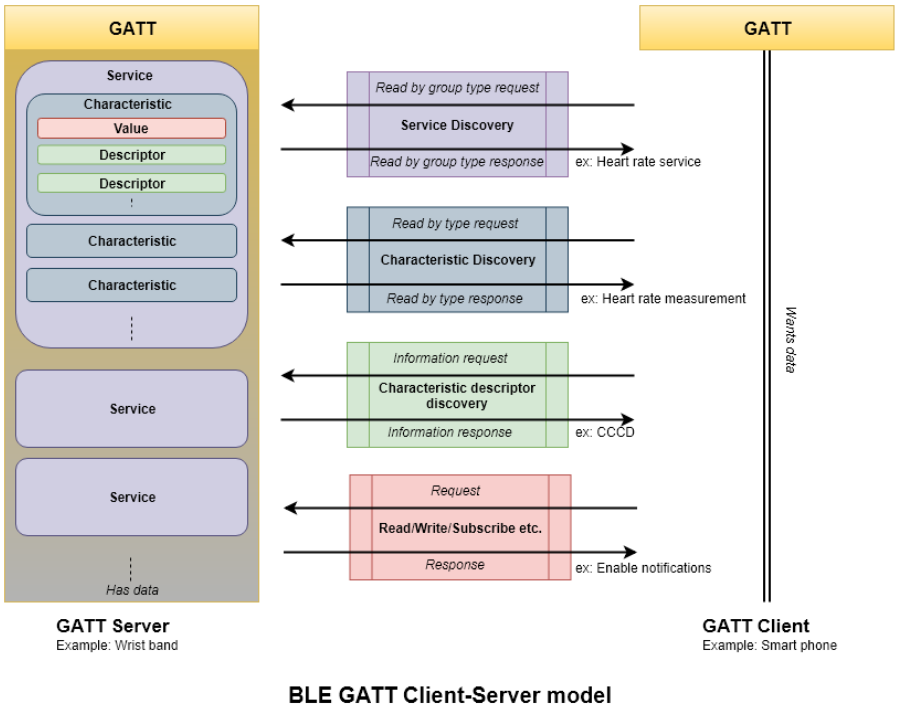
- The Bluetooth protocol operates at 2.4GHz in the same unlicensed ISM frequency band where RF protocols like ZigBee and WiFi also exist.
- Bluetooth networks (commonly referred to as piconets) use a master/slave model to control when and where devices can send data. In this model, a single master device can be connected to up to seven different slave devices. Any slave device in the piconet can only be connected to a single master.

Bluetooth - Generic Access Profile



- The Generic Access Profile (GAP) controls connections and advertising in Bluetooth. GAP is what makes your device visible to the outside world, and determines how two devices can (or can't) interact with each other.
- GAP defines various roles for devices, but the two key concepts to keep in mind are Central devices and Peripheral devices.
  - Peripheral devices are small, low power, resource constrained devices that can connect to a much more powerful central device. Peripheral devices are things like a heart rate monitor, a BLE enabled proximity tag, etc.
  - Central devices are usually the mobile phone or tablet that you connect to with far more processing power and memory.

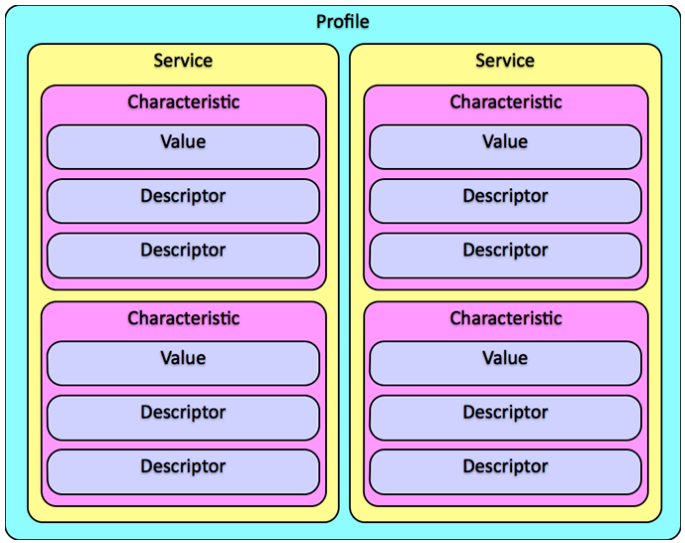
Bluetooth - Generic Attribute Profile (GATT)



BLE GATT Client-Server model

- Generic Attribute Profile defines the way that two BLE devices transfer data back and forth using concepts called Services and Characteristics. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table.
- GATT comes into play once a dedicated connection is established between two devices, meaning that you have already gone through the advertising process.

Bluetooth - Services and Profiles



- A Profile is a pre-defined collection of Services. The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service.
- Services break data up into logic entities, and contain specific chunks of data called characteristics. A service can have one or more characteristics, and each service distinguishes itself from other services with a unique numeric ID called a UUID, which can be either 16-bit (official BLE Services) or 128-bit (custom services).
- A Characteristic contains a single data point or an array of related data. For example: X/Y/Z values of an accelerometer.

ASCII Reminder

- ASCII characters (the symbols that we are use to reading) can be represented by a single byte (uint8_t).

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

ASCII: American Standard Code For Information Interchange

Argon BLE - UART Service

```
1  // These UUIDs were defined by Nordic Semiconductor and are now the defacto standard for
2  // UART-like services over BLE. Many apps support the UUIDs now, like the Adafruit
       Bluefruit app.
3  const BleUuid serviceUuid("6E400001-B5A3-F393-E0A9-E50E24DCCA9E");
4  const BleUuid rxUuid("6E400002-B5A3-F393-E0A9-E50E24DCCA9E");
5  const BleUuid txUuid("6E400003-B5A3-F393-E0A9-E50E24DCCA9E");
6
7  BleCharacteristic txCharacteristic("tx", BleCharacteristicProperty::NOTIFY, txUuid,
       serviceUuid);
8  BleCharacteristic rxCharacteristic("rx", BleCharacteristicProperty::WRITE_WO_RSP, rxUuid,
        serviceUuid, onDataReceived, NULL);
9  BleAdvertisingData data;
10
11 //onDataReceived is used to receive data from Bluefruit Connect App
12 void onDataReceived(const uint8_t* data, size_t len, const BlePeerDevice& peer, void*
       context) {
13   uint8_t i;
14
15   Serial.printf("Received data from: %02X:%02X:%02X:%02X:%02X:%02X \n", peer.address()
       [0], peer.address()[1],peer.address()[2], peer.address()[3], peer.address()[4], peer
       .address()[5]);
16   Serial.printf("Bytes: ");
17   for (i = 0; i < len; i++) {
18     Serial.printf("%02X ",data[i]);
19   }
20   Serial.printf("\n");
21   Serial.printf("Message: %s\n",(char *)data);
22 }
```

Argon BLE - UART Transmit Example

```
1  const int UART_TX_BUF_SIZE = 20;
2  uint8_t txBuf[UART_TX_BUF_SIZE];
3  uint8_t i;
4
5  SYSTEM_MODE(SEMI_AUTOMATIC); //Using BLE and not Wifi
6
7  void setup() {
8      Serial.begin();
9      waitFor(Serial.isConnected, 15000);
10
11     BLE.on();
12     BLE.addCharacteristic(txCharacteristic);
13     BLE.addCharacteristic(rxCharacteristic);
14     data.appendServiceUUID(serviceUuid);
15     BLE.advertise(&data);
16
17     Serial.printf("Argon BLE Address: %s\n", BLE.address().toString().c_str());
18 }
19
20 void loop() {
21     for(i=0;i<UART_TX_BUF_SIZE-1;i++) {
22       txBuf[i] = random(0x40,0x5B); //Captial ASCII characters plus @
23     }
24     txBuf[UART_TX_BUF_SIZE-1] = 0x0A;
25     txCharacteristic.setValue(txBuf, UART_TX_BUF_SIZE);
26     for(i=0;i<UART_TX_BUF_SIZE;i++) {
27         Serial.printf("%c",txBuf[i]);
28     }
29     delay(5000);
30 }
```

Formatted Print to a Buffer

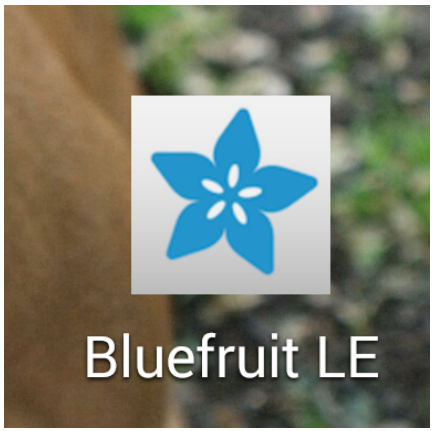
```
// sprintf does a formatted print to a buffer of type char[

const int BUFSIZE = 50;
byte buf[BUFSIZE];
int people;
float dogs,avg;

void setup() {
  Serial.begin(9600);
  people = 5;
  dogs = 13;
}

void loop() {
  avg = dogs / people;
  sprintf((char *)buf,"The %i people have on average %0.2f dogs \n",people, avg);
  Serial.printf("Buf contains the string: %s",(char *)buf);
}
```

The buffer can then be used to as the data payload between devices, for example, using BlueTooth.

Note: Windows treats a $\backslash n$ as a LF-CR (0x0D0A), if a LF is needed (e.g., for BLE) then it needs to be inserted manually:

```
buf[BUFSIZE-1] = 0x0A;
```

Assignment: L17_BlueTooth

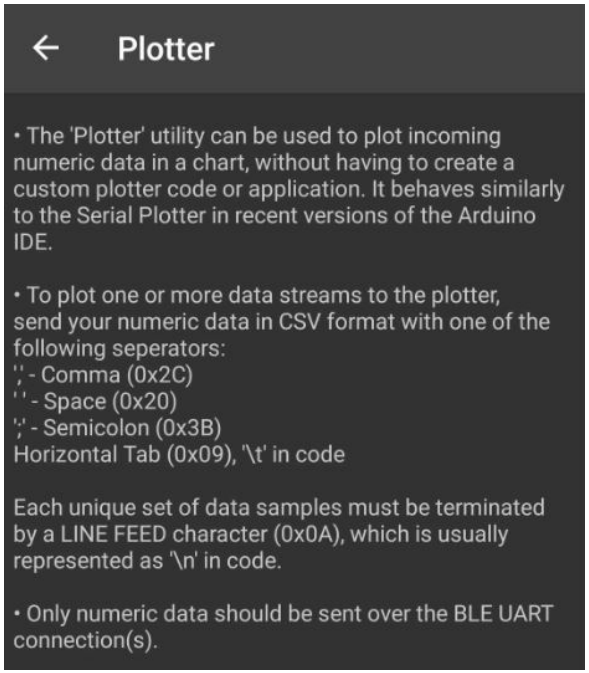


- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

L17_01_BlueTooth

- Load Bluefruit Connect on your smart device. Using the code on the proceeding slides, establish BLE UART communications
- Attach the encoder and NeoPixel ring to your Argon.
- Repeat the NeoPixel ring assignment (L06_02_NeoPixel) on your Argon (with only 12 pixels).
- When it changes, send the the encoder or NeoPixel position via BLE to Bluefruit Connect.
- Reset the encoder and NeoPixel ring to the appropriate state when values 0-11 are received from Bluefruit Connect.

Assignment: L17_BlueTooth - Colors



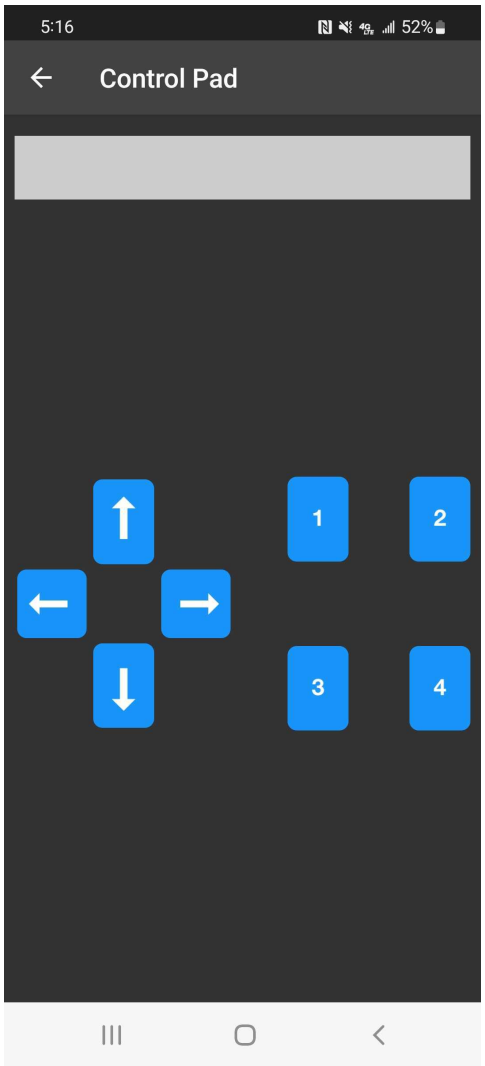
ColorPicker String
5 bytes plus CR

['!'] ['C'] [byte red] [byte green] [byte blue] [CRC]

L17_01_BlueTooth (Continued)

- Plotter function in Bluefruit Connect:
  1. Every time the encoder moves, generate and change the pixels to a random color (R,G,B format, not Hex).
  2. Plot the pixel number and three RGB components on the Bluefruit Plotter.
- Controller -> Color Picker screen on Bluefruit Connect:
  1. Send a color to the Argon.
  2. Identify in your code if ColorPicker string or general UART string is received.
  3. If ColorPicker, then using bitwise left shift and OR to convert string to hex color similar to L15_02_RetrieveShow
  4. Change your Neopixel color to match Color Picker

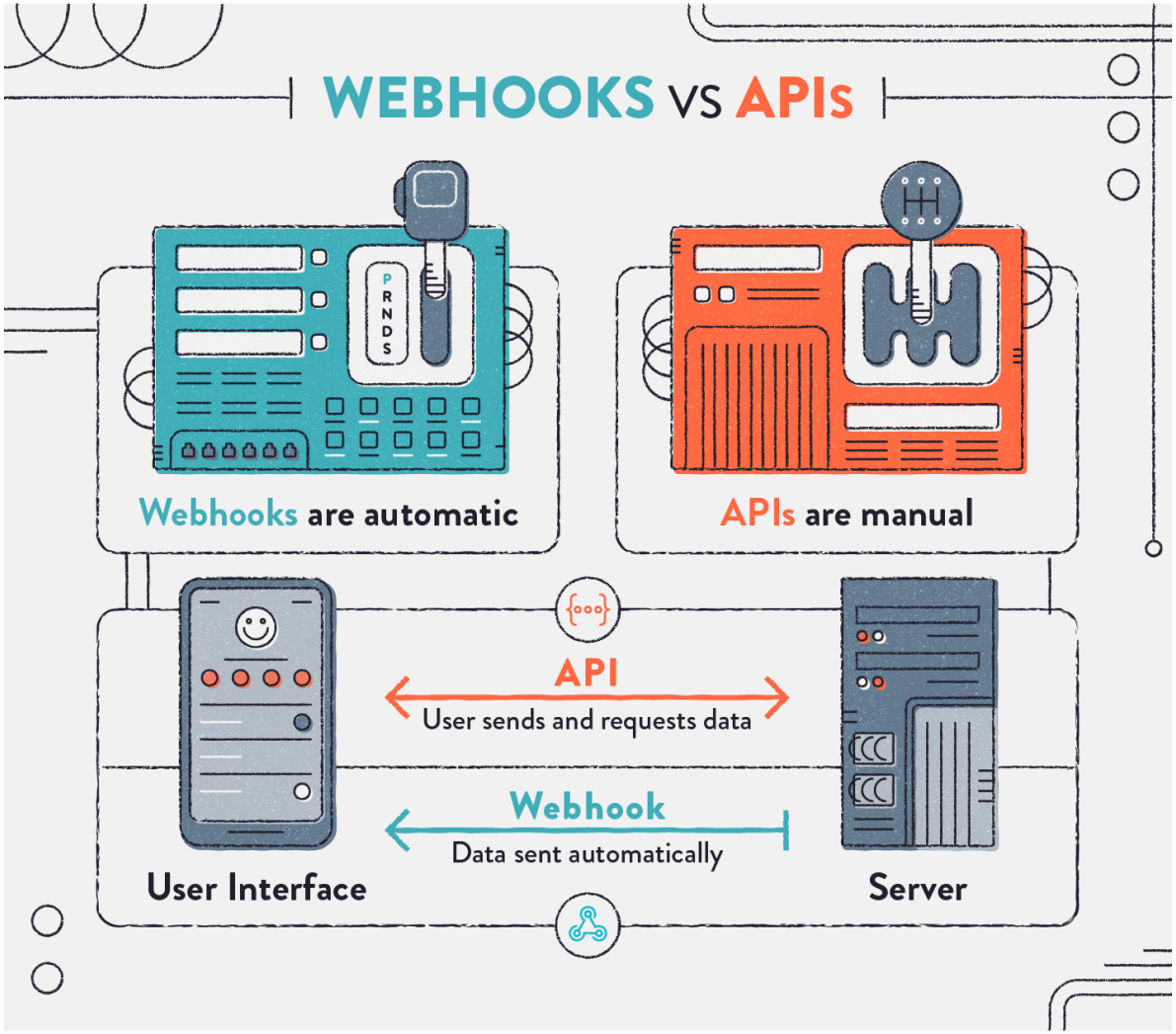
Assignment: L17_BlueTooth - Colors (EXTRA)



L17_01_BlueTooth (EXTRA)

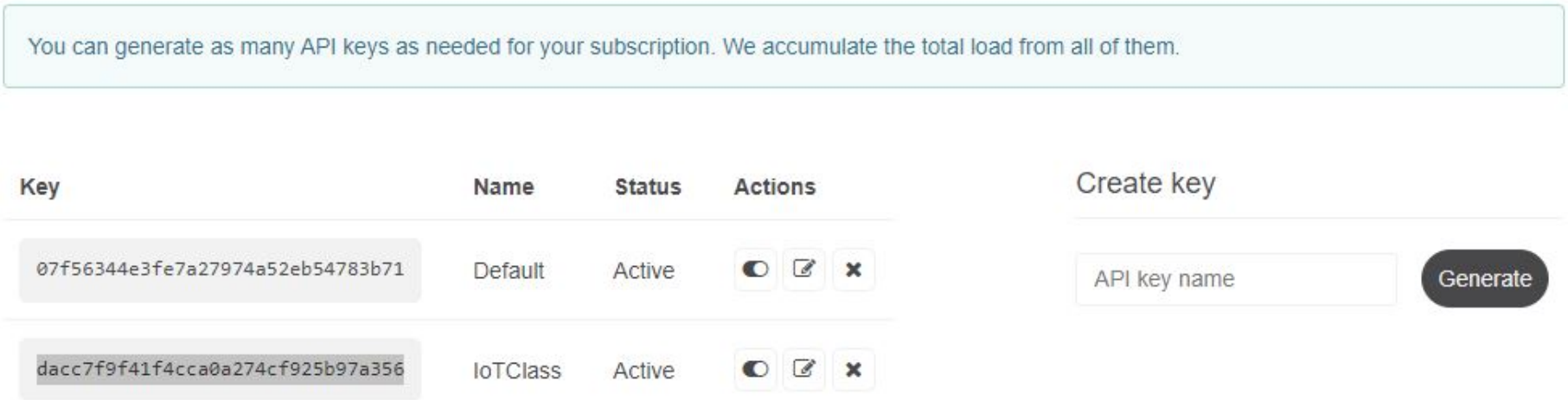
- Experiment sending signals from the Bluefruit Connect Control Pad
- Use the Up/Down arrows to change the neopixel brightness
- Use the Left/Right arrows to cycle through a rainbow
- Code in a different neopixel effect for each of the number keys

Webhooks


WEBHOOKS vs APIs
P
R
N
D
S
Webhooks are automatic
APIs are manual
API
User sends and requests data
Webhook
Data sent automatically
User Interface
Server

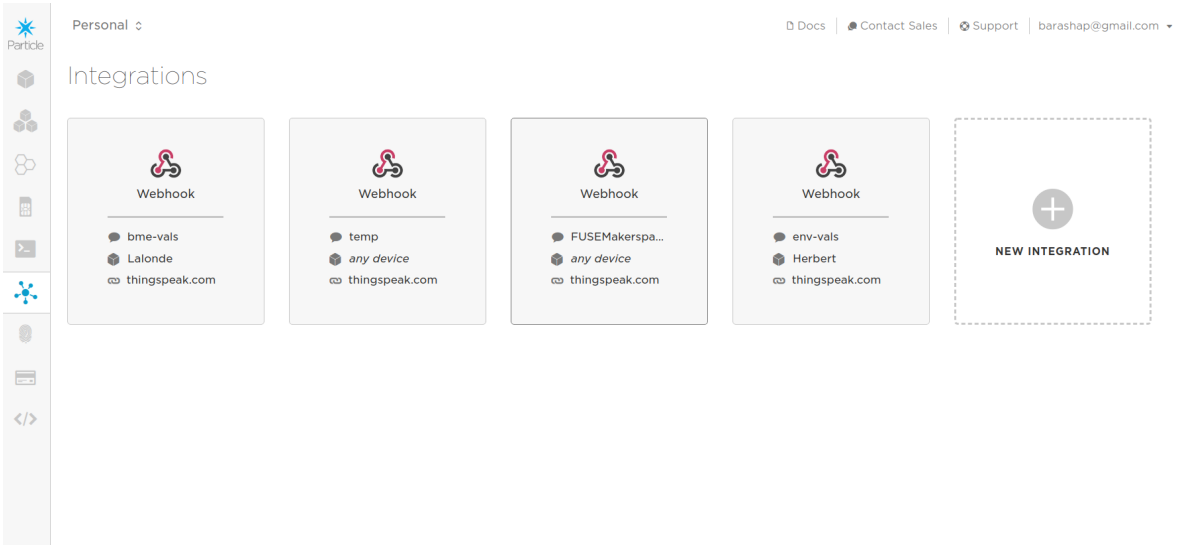
Step 1 - OpenWeather

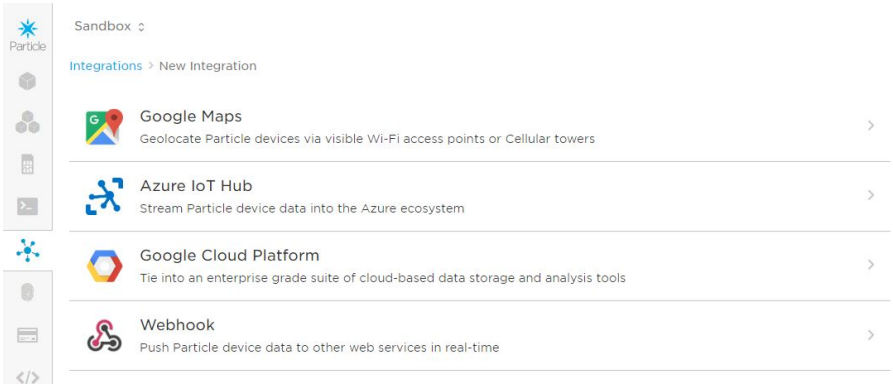
- Create an account at `openweathermap.org`
- Generate an API key at:
  `https://home.openweathermap.org/api_keys`



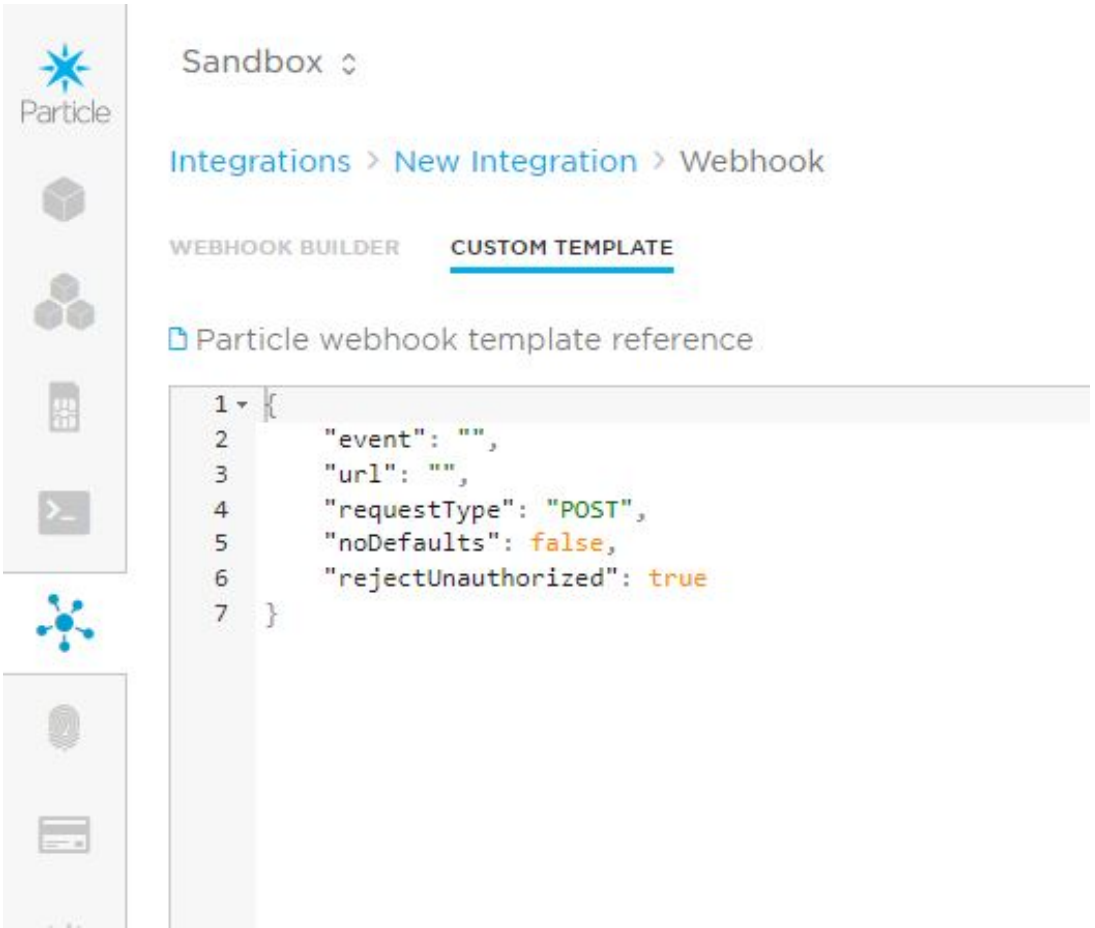
Step 2 - Create Webhook

From `console.particle.io`:





Step 3 - Select Custom Template



Step 4 - Update Custom Template with API format

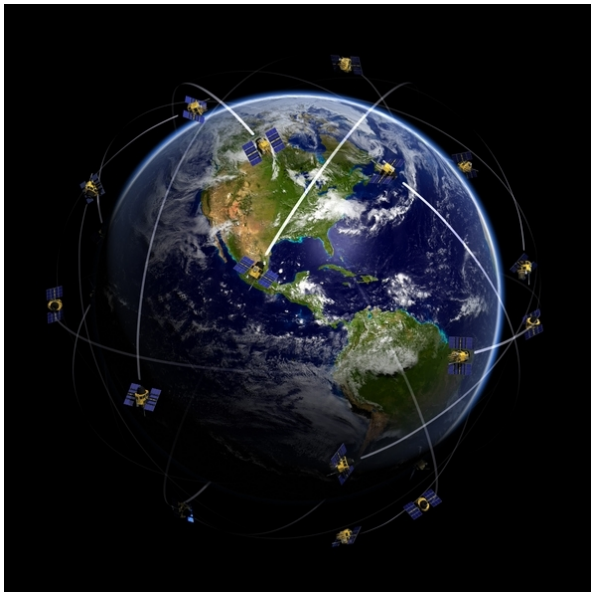
Adapted from `https://openweathermap.org/api/one-call-api` and `https://openweathermap.org/current`

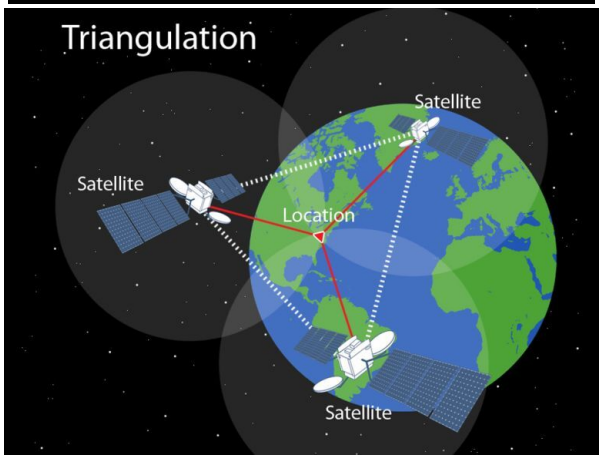
```
{
    "event": "GetWeatherData",
    "responseTopic": "{{PARTICLE_DEVICE_ID}}/{{PARTICLE_EVENT_NAME}}",
    "url": "https://api.openweathermap.org/data/2.5/onecall",
    "requestType": "GET",
    "noDefaults": true,
    "rejectUnauthorized": true,
    "responseTemplate": "{\"lat\":{{lat}},\"lon\":{{lon}},\"dt\":{{current.dt}},\"temp
     \":{{current.temp}},\"uvi\":{{current.uvi}},\"clouds\":{{current.clouds}},\"ws\":{{
     current.wind_speed}},\"wd\":{{current.wind_deg}} }",
    "query": {
        "lat": "{{lat}}",
        "lon": "{{lon}}",
        "exclude": "minutely,hourly,daily,alerts",
        "units": "metric",
        "appid": "dacc7f9f41f4cca0a274cf925b97a356"
    }
}
```

Step 5 - Particle Argon Code

```
const char *EVENT_NAME = "GetWeatherData";
unsigned int lastTime;
const float lat=35.0045, lon=-106.6465;  //update to your favorite location

void setup() {
    Serial.begin(9600);
    waitFor(Serial.isConnected,15000);
    String subscriptionName = String::format("%s/%s/", System.deviceID().c_str(),
     EVENT_NAME);
    Particle.subscribe(subscriptionName, subscriptionHandler, MY_DEVICES);
    Serial.printf("Subscribing to %s\n", subscriptionName.c_str());
}

void loop() {
    if((millis() - lastTime) > 60000) {
        Serial.printf("\n\nTime = %i\n",millis());
        Particle.publish(EVENT_NAME, "", PRIVATE);
        Particle.publish(EVENT_NAME, String::format("{\"lat\":%0.5f,\"lon\":%0.5f}", lat,
      lon), PRIVATE);
        lastTime = millis();
    }
}

void subscriptionHandler(const char *event, const char *data) {
    JSONValue outerObj = JSONValue::parseCopy(data);
    JSONObjectIterator iter(outerObj);
    while(iter.next()) {
            Serial.printf("key=%s value=%s\n",(const char *) iter.name(),(const char *)
     iter.value().toString());
    }
}
```

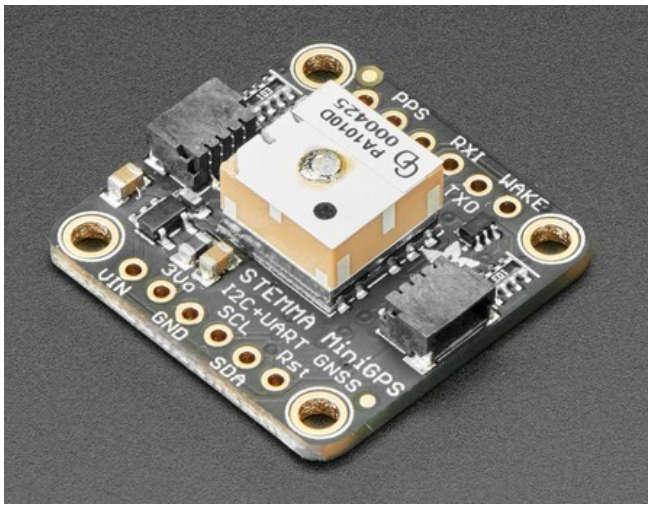
Global Positioning System





- 29 satellites (24 active plus 5 reserve) at an altitude of 12550 miles, circling the earth twice per day.
- Envisioned by Aerospace Corporation 1963
- First satellite 1973
- Open to commercial use 1985
- GPS receiver measures time it takes signal (at speed of light) to get from satellite to receiver.
- Uses triangulation from at least 4 satellites to obtain latitude, longitude, altitude, and time.
- GNSS is an international system of satellites that includes GPS and others

Global Positioning System



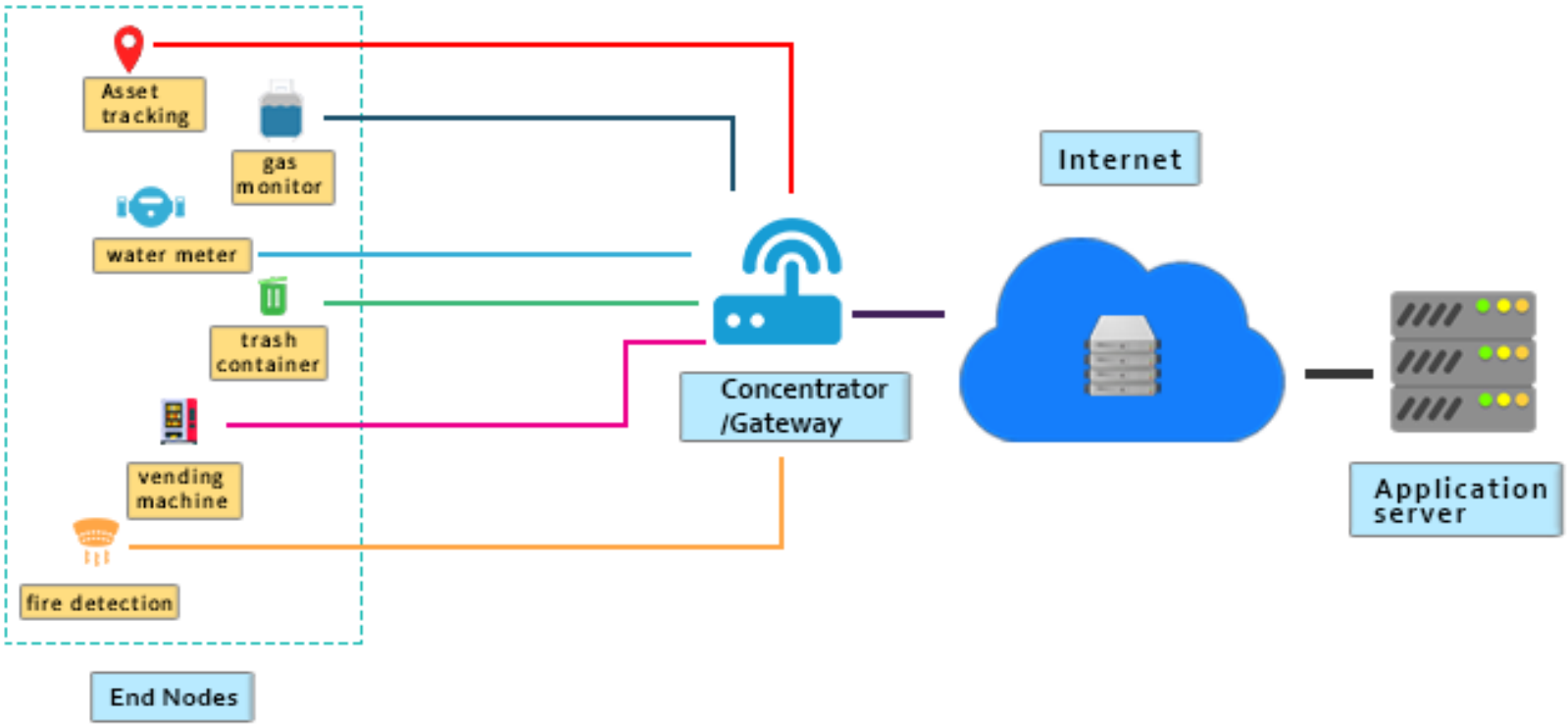
- Miniature GPS module
- Houses a complete GPS/GNSS solution
- Both I2C and UART interfaces
- STEMMA interface for easy prototyping

Adafruit_GPS Library

```
$GPGGA,202410.000,4042.6000,N,07400.4858,W,1,4,3.14,276.7,M,-34.2,M,,*63
$GPRMC,202410.000,A,4042.6000,N,07400.4858,W,0.08,161.23,160412,,,A*70
$GPGGA,202411.000,4042.5999,N,07400.4854,W,1,3,17.31,275.8,M,-34.2,M,,*5D
$GPRMC,202411.000,A,4042.5999,N,07400.4854,W,0.14,161.23,160412,,,A*7A

Time: 20:24:11.0
Date: 16/4/2012
Fix: 1 quality: 1
Location: 4042.5998N, 7400.4853W
Speed (knots): 0.14
Angle: 161.23
Altitude: 275.80
Satellites: 3
```

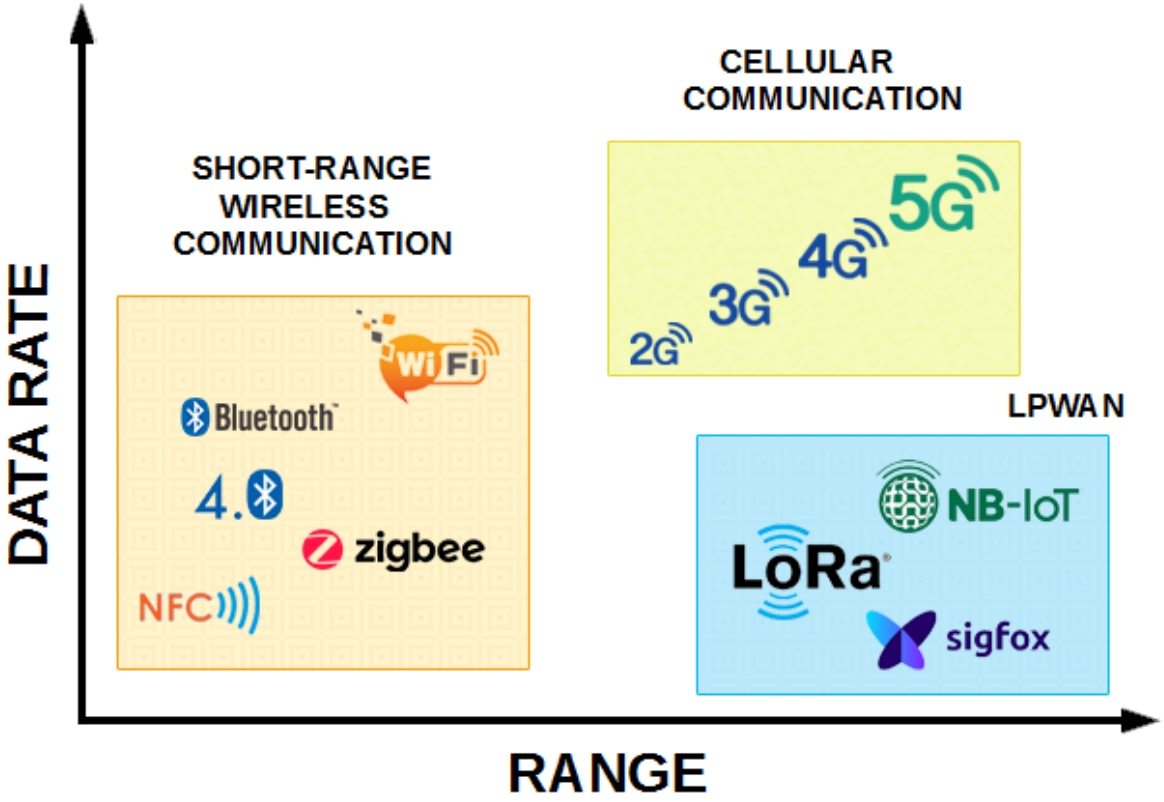
- Call gps.read() at the beginning of void loop()
- Then immediately call GPS.parse(GPS.lastNMEA()) to make the data available
- When you need it, you can than access GPS.latitude, GPS.longitude, GPS.speed, etc.

LoRa



LoRa is a long range, low power, inexpensive technology for Internet of Things

LoRa Range vs Data Rate



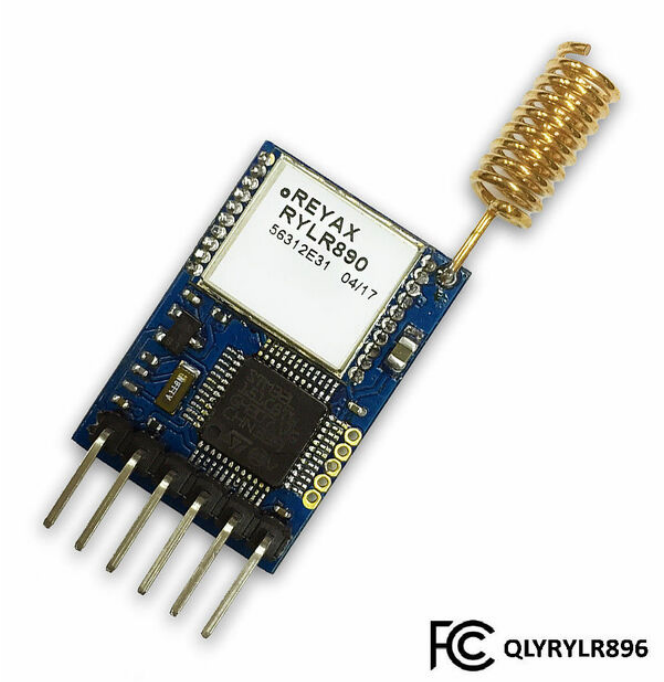
LoRa uses license-free sub-gigahertz radio frequency ISM bands in the deployed region such as 868 MHz in Europe and 915MHz in North America.

LoRa Features

LoRa was many desirable features:

- It has very wide coverage range about 5 km in urban areas and 15 km in suburban areas
- Battery lifetime up to 15 years
- One LoRa gateway takes care of thousands of nodes.
- Easy to deploy and low cost.
- Enhanced the secure data transmission by embedded end-to-end AES128 encryption

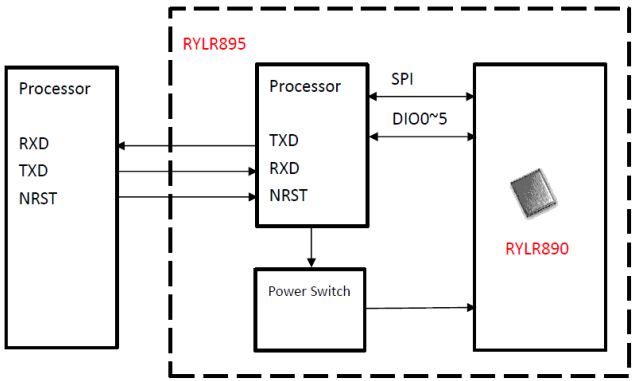
RXLR896 LoRa Module



Features:

- Semtech SX1276 Engine
- Excellent blocking immunity
- Low receive current
- High sensitivity
- Control easily by AT commands
- 127 dB Dynamic Range RSSI
- Designed with integrated antenna
- AES128 Data encryption

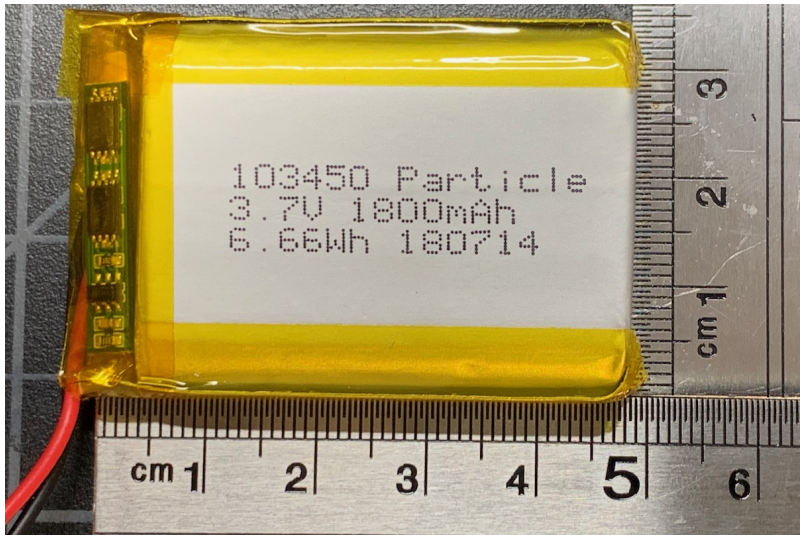
AT Commands



AT commands are commands which are used to control the modems where AT stands for Attention. These commands were derived from Hayes commands which were used by the Hayes smart modems. Every wireless modem requires an AT command to interact with a computer machine.

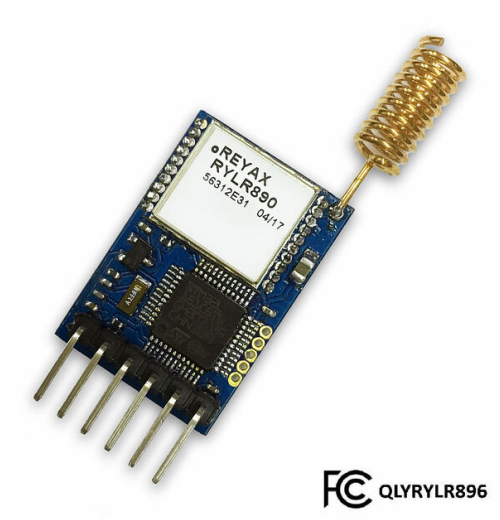
Communication between the Argon and RYLR896 takes place over UART (Serial1) using the same "Serial" commands that were used in Lesson 3 and Lesson 4.

Batteries - Going Mobile



1. All Particle platforms have JST-PH pins for a Lithium Polymer (LiPo) battery
   - Always check wiring polarity
2. Battery can be charged via USB port or V_{bus}.
3. Battery power is available on LiPo+ or 3.3V pins

L17_03_LoRaGPS



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

Features:

1. Using the L17_00_GPS code, obtain GPS coordinates
2. Modify L17_03_LoRaGPS:
   - Integrate the LoRa, GPS, and OLED
   - Get your own RADIOADDRESS from instructors
   - Display FUSE Sound and Particulates to OLED
   - Send live GPS coordinates back to LoRa base-station
   - Find the distance you can go N/S/E/W and get a LoRa signal back to FUSE.
   - Class Trip: repeat at ABQ Biopark

L18_PowerManagement

System.Sleep()

System.sleep() can be used to lower power consumption and increase battery life when the Particle is only needed intermittently to interact with the environment.

	STOP	ULTRA_LOW_POWER	HIBERNATE
Relative power consumption	Low	Lower	Lowest
Relative wake options	Most	Some	Fewest
Execution continues with variables intact	✓	✓	

Wake Mode	Gen 2	Gen 3
GPIO	✓	✓
Time (RTC)	✓	✓
Analog		✓
Serial		✓
BLE		✓
Cellular		✓
Wi-Fi		✓

Ultra Low Power example

The below code places the Argon in Ultra_Low_Power mode, enabling it to be awakened either by a RISING signal on Pin D0 or after the time specified by the variable sleepDuration.

```
void sleepULP() {
  SystemSleepConfiguration config;
  config.mode(SystemSleepMode::ULTRA_LOW_POWER).gpio(D0,RISING).duration(sleepDuration);
  SystemSleepResult result = System.sleep(config);
  delay(1000);
  if (result.wakeupReason() == SystemSleepWakeupReason::BY_GPIO) {
    Serial.printf("Awakened by GPIO %i\n",result.wakeupPin());
  }
  if (result.wakeupReason() == SystemSleepWakeupReason::BY_RTC) {
    Serial.printf("Awakened by RTC\n");
  }
}

/*
 * Other wake-up modes
 *
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).analog(pin, value, mode);
 *    mode can be: ABOVE, BELOW, or CROSS
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).usart(Serial1);
 *    note: Serial can not be used for wake-up
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).ble();
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).network(NETWORK_INTERFACE_WIFI_STA);
 */
```

Assignment: L18_01_Sleep

Using your L17_01_BlueTooth assignment

- Place the Argon in Ultra_Low_Power mode when the device hasn't recieved an input (either the Encoder or BLE) in the last minutes.
- Implement wake-up via
  1. RTC
  2. Encoder Button
  3. Signal from Bluefruit Connect
- Measure the power consumption in each state

L19_Lists and Trees

Struct datatype and Member Operators

struct enables the programmer to create a variable that structures a selected set of data.

```
struct Employee {
  char name[10];
  int idNumber;
  float salary;
}

Employee instructor;
Employee IoTengineers[10];

void setup {
  instructor.name = “Brian”;
  instructor.idNumber = “42”;
  instructor.salary = 212.47;
  IoTengineers[1].name = “Sally”;
}
```

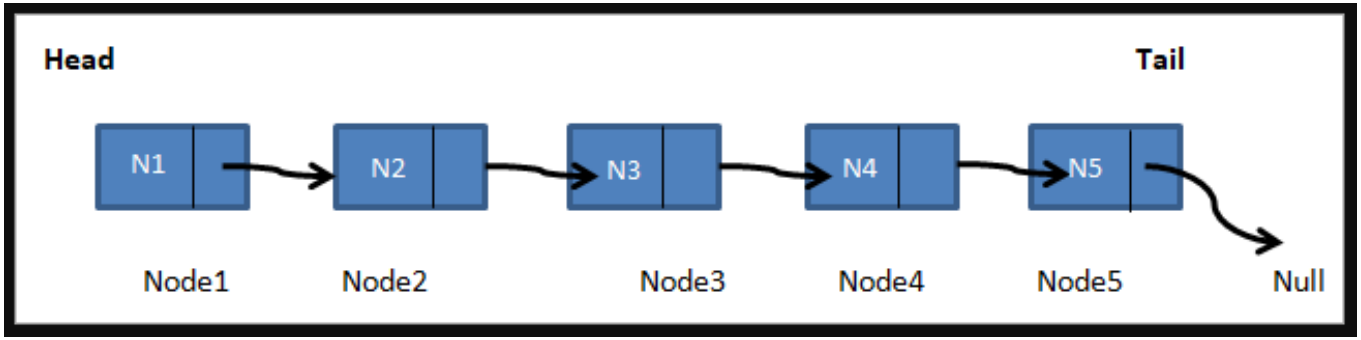
struct name

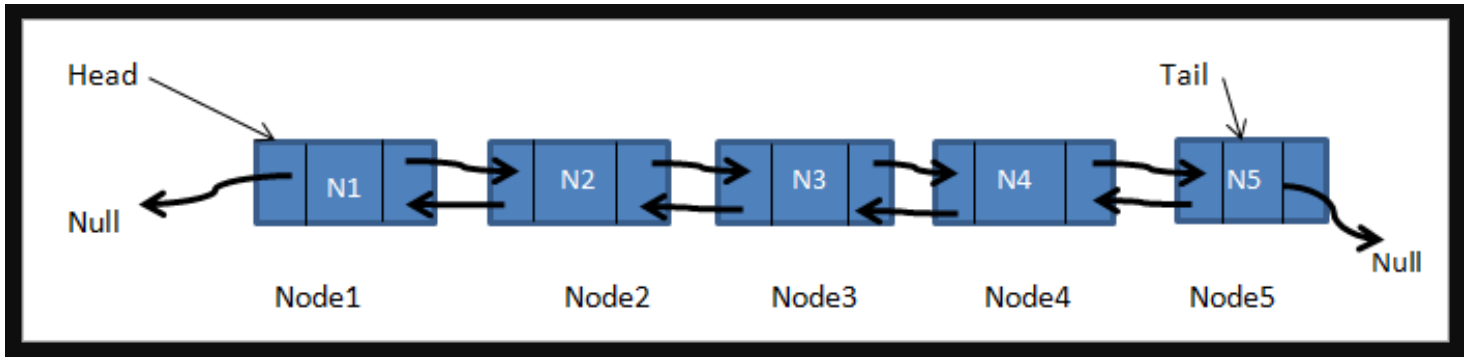
Members of the structure

Declare individual variable of data type Employee

Declare an array of data type Employee

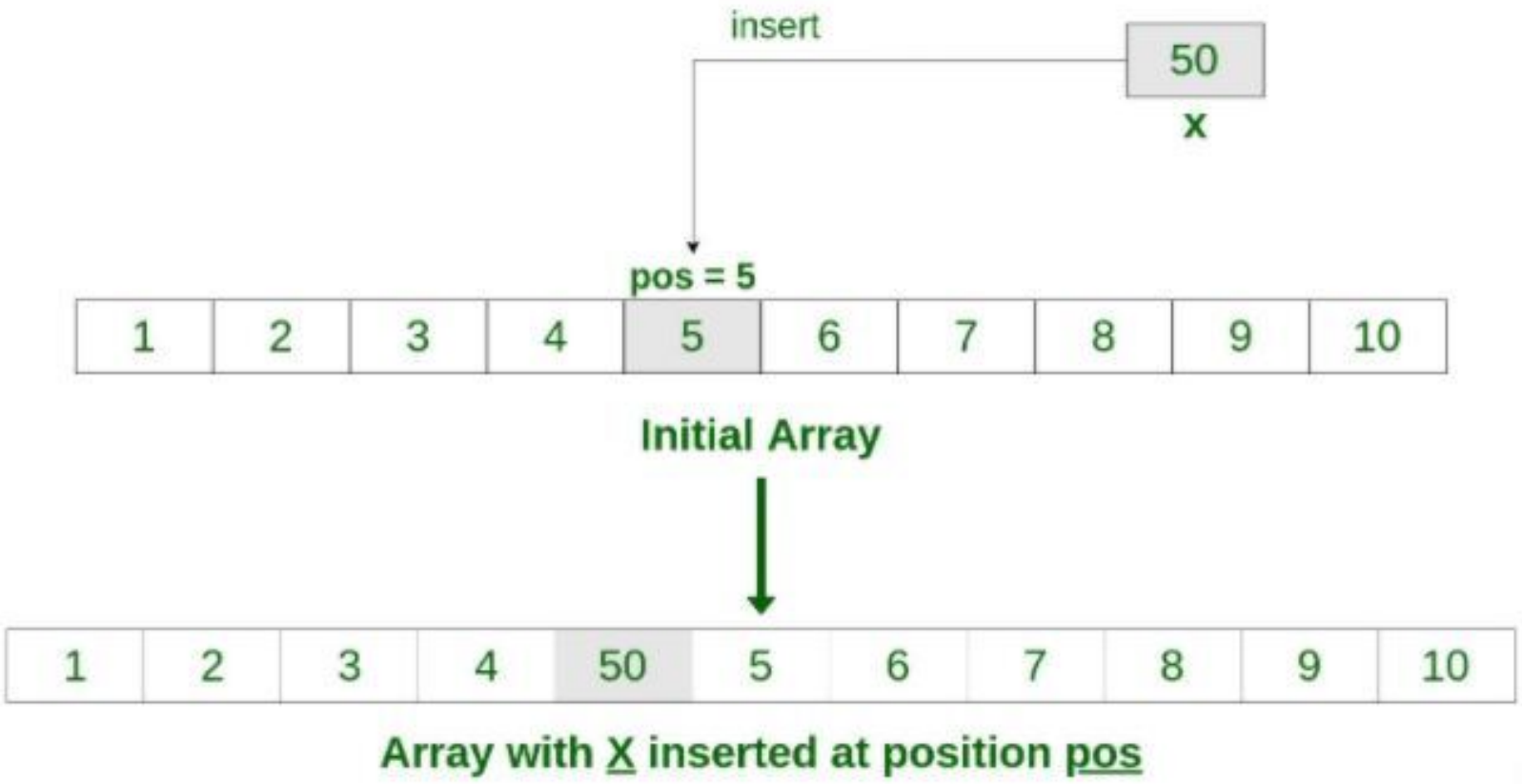
Linked Lists and Doubly Linked Lists



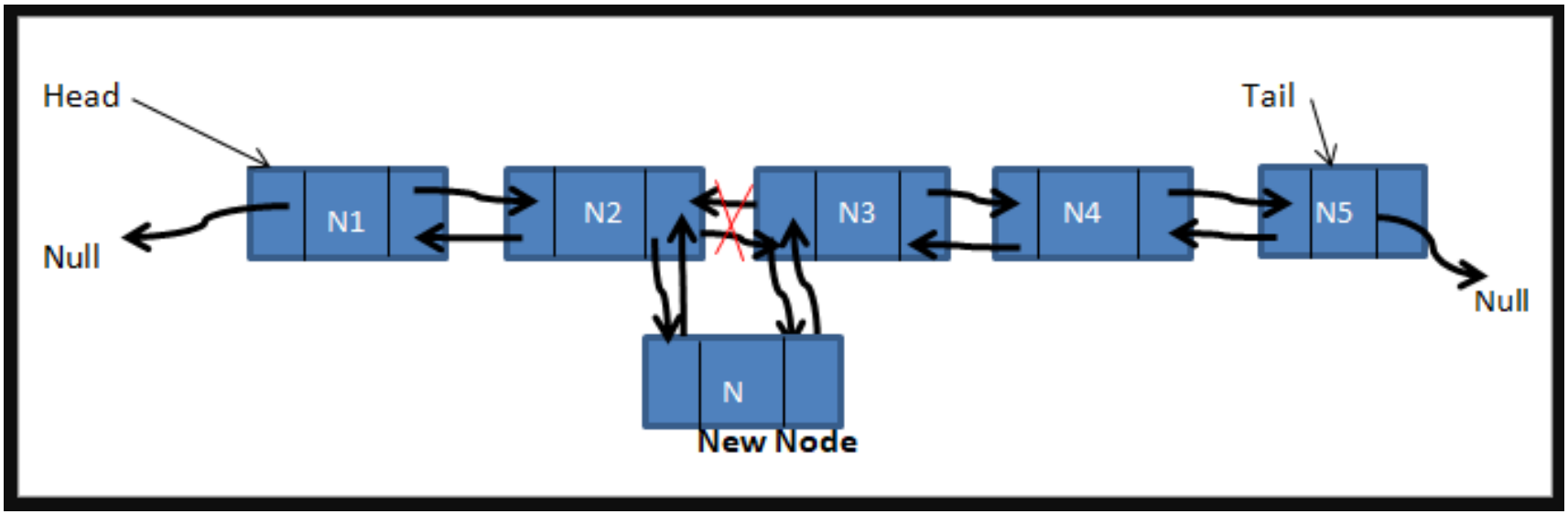



```
1 struct node {
2   struct node *prev;
3   int data;
4   struct node *next;
5 };
```

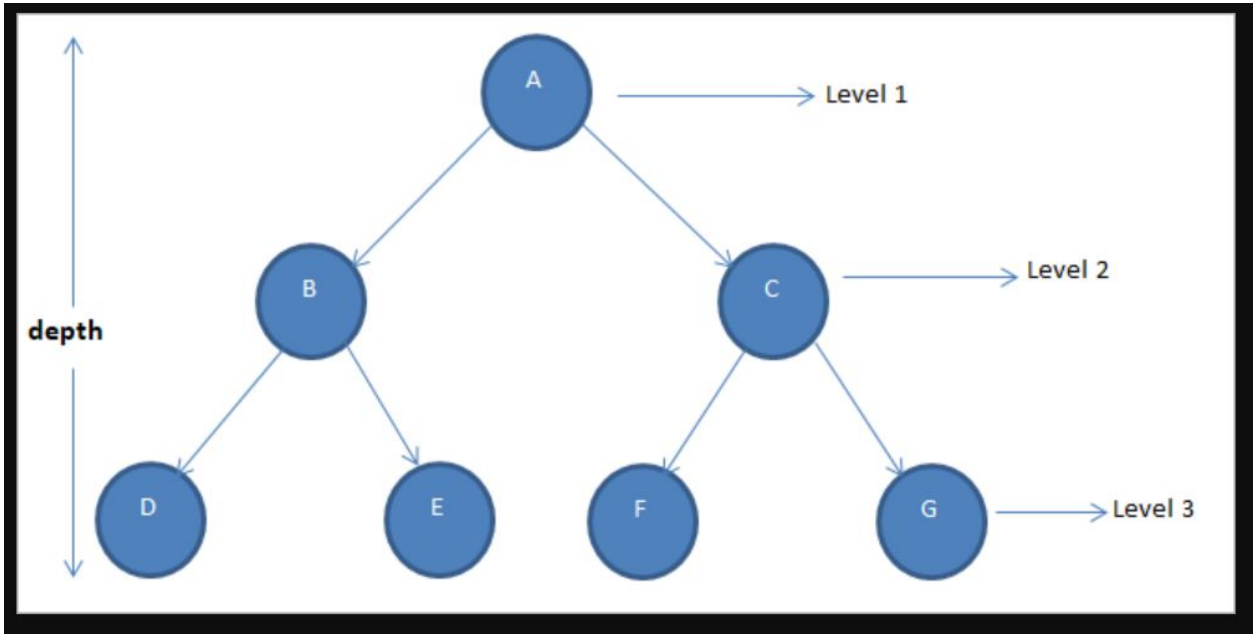
Inserting a "cell" into an Array



Inserting a "cell" into a Linked List

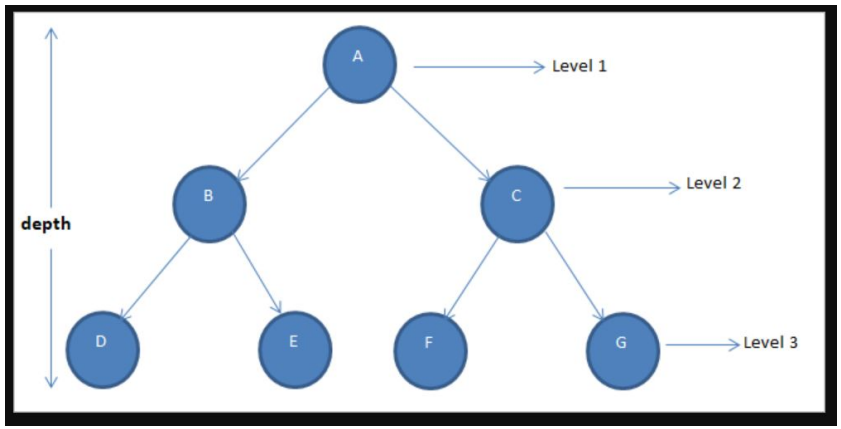


Binary Trees




```
1 struct bintree_node{
2     bintree_node *left;
3     bintree_node *right;
4     int data;
5 };
```

Binary Trees



Uses of Binary Trees

- Binary Search
- Hash Trees
- Heaps
- Huffman Coding
- Syntax Tree

```
1 struct bintree_node{
2     bintree_node *left;
3     bintree_node *right;
4     int data;
5 };
```

Non-Embedded C++

HelloWorld in C++

```cpp
// include the standar input-output library
// most include statements need the .h, but iostream is an exception
#include <iostream>

// namespace is used to declare regions with the global space
// std enable the standard console (monitor) and input (keyboard)
using namespace std;

char myName[20];

//main is the first function executed in your cpp code
int main()
{
    cout << "Hello World!!!\n";    // cout = output to console
    cout << "What is your name: ";
    cin >> myName;                 // cin = input from console
    cout << "Hello " << myName << ", I hope you are having a nice day.\n";

    return 0;  // denotes successfully executed
}
```

Instructions for installing and writing C++ code in VSCode:
https://code.visualstudio.com/docs/languages/cpp

The Big Question: What about main()

```
#include <Arduino.h>

extern "C" int main(void)
{
#ifdef USING_MAKEFILE

  // To use Teensy 3.0 without Arduino, simply put your code here.
  // For example:

  pinMode(13, OUTPUT);
  while (1) {
    digitalWriteFast(13, HIGH);
    delay(500);
    digitalWriteFast(13, LOW);
    delay(500);
  }


#else
  // Arduino's main() function just calls setup() and loop()....
  setup();
  while (1) {
    loop();
    yield();
  }
#endif
}
```

Hello World - What a microcontroller sees

HelloWorld.ino

```
1 void setup() {
2   Serial.begin(9600);
3   Serial.printf("Hello World! \n");
4 }
5
6 void loop() {}
```

Hex Code and Assembly Language

```
1  HelloWorld.bin:     file format binary
2  Disassembly of section .data:
3  00000000 <.data>:
4      101c: bd10          pop {r4, pc}
5      101e: 4402          add r2, r0
6      1020: 4603          mov r3, r0
7      1022: 4293          cmp r3, r2
8      1024: d002          beq.n 0x102c
9      1026: f803 1b01     strb.w  r1, [r3], #1
10     102a: e7fa          b.n 0x1022
11     102c: 4770          bx  lr
12     102e: 0000          movs  r0, r0
13     1030: b538          push  {r3, r4, r5, lr}
```

Useful BASH commands

Redirect from stdout (standard output)

The > and >> signs are used for redirecting the output of a program to something other than stdout (standard output, which is the terminal by default).

- The >> appends to a file or creates the file if it doesn't exist.
- The > overwrites the file if it exists or creates it if it doesn't exist.

Examples:

```
1
2  # Create a file called "allmyfiles.txt" and fill with the directory listing
3
4  ls > allmyfiles.txt
5
6  # Adds "End the directory listing" to the end of "allmyfiles.txt"
7
8  echo "End of directory listing" >> allmyfiles.txt
9
10 # Create a zero-byte file with the name "newfile.txt"
11
12 > newfile.txt
13
14 # Redirect Particle Serial Monitor output to the file "filename.csv"
15
16 Particle serial monitor --follow >> filename.csv
```

GitHub - Part 3

Review: The .gitignore file

Remember to add the following items to your .gitignore file for your Capstones.

- credentials.h
- target

Note: The target folder contains a binary version of your code that has your API keys in it. So you want to .gitignore the target folder!

Collaborating with Github

Using Git inside an IDE - Visual Studio Code

You can run Git commands from the Visual Studio Code terminal. Advanced IDEs (Interactive Development Environments) usually integrate with a VCS like Git.

1. Open a project folder in VS Code.
2. Open a terminal window in VS Code.
3. Choose your preferred Git command tool.
   - Windows Command
   - Windows Powershell
   - Git Bash $\rightarrow$ like Linux
4. Now type your Git and file/directory commands in the terminal.

This is easier than Powershell because you are already in your project folder so VS Code can show your Git status in the IDE.

Typical GitHub workflow and branches

- Master or Main is the good code. This is generally the code that is also available on a production system.
- A production system is a live system that people are actually using.
- You don't want to push unfinished, untested code to your master or main branch.

Typical workflow:

Feature → development → staging/testing/release → master (main)/production

`https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`

Practice: Branching and merging code

1. Create a test repo in GitHub and clone it to your local system.
2. Create a new VS Code project in your new repo. Write some code and push to your new repo.
3. Create a branch called "dev"
   - git branch dev → create the new branch
   - git checkout dev → switch to the new branch
   - git branch → confirm the branch you are now in.
4. Make changes to your test file.
5. commit and push your changes.
6. git checkout main → to switch back to the main branch.
7. git branch → to see what branch you are in.
8. git merge dev → to merge your dev branch changes into main.
9. git push → to push the merged changes to main
10. Look at your commit history to see how the merge was handled.

Review: Git commands

git clone → to copy a complete repo to your local machine.
git log → shows commit history for a repo.
git show <commit number> → shows details for a specified commit.
git diff <file name> → shows differences between file versions.
git status → shows what is staged for commit so you know what will be committed *BEFORE* you commit it.
git branch → to create a new branch or see what branch you are currently in.
git checkout <file name> → to rollback files to the current Git version.
git checkout <branch name> or **git switch** <branch name> → to change branches.
git pull → to get the latest code from a repo.
git add → to stage files to be committed.
git commit → to apply timestamp and version to files in a local repo.
git push → to upload commits to a remote repo.

Collaboration in GitHub

- Collaboration allows multiple people (GitHub accounts) to work on the same project.
- For your capstones, you will need to add classmates as collaborators to your repo.
- Applying rules to the GitHub workflow helps avoid merge conflicts and problem code.
  - Communicate with other team members about changes you are making.
  - Segregate assignments if possible to avoid changing the same code at the same time.
  - **Always do a git pull from a branch** before merging to avoid having your branch ahead of a higher level branch.
  - Use a separate branch for development to avoid breaking the master or main branch of your teammates' code.
  - Thoroughly test your code before committing and pushing to GitHub.
  - Have a teammate do a code review before merging to a higher level branch.

Collaboration workflow

1. Share a project repo with all team members in GitHub.
2. Clone the shared repo to your local machine.
3. If you do not have a development or working branch, create one.
4. Switch to the new branch in the repo.
5. Do your development and testing.
6. Commit and push your branch to GitHub.
7. Notify the repo owner that your code is ready for review and merging to the next higher level branch.

Note: design a workflow that works for your team. Avoid working on the master or main branch. Reserve the master or main branch for final production-level, presentation-worthy code.

Practice: Collaborating

1. Add your teammate as a collaborator to your repo.
   - Settings > Manage Access > Invite Collaborator
2. Approve the invitation to collaborate in your email.
3. Confirm that you can now see your teammate's repo.
4. Clone your teammate's repo into your IoT folder.
5. Create a new branch for your development.
   - git branch yourname-dev
   - git checkout yourname-dev
6. Make a change to a file on your dev branch.
7. Commit and push your change.
8. Have your teammate confirm that they can see your new branch and change in their repo.

Pull Requests

A pull request is a safety feature in GitHub that enforces a code review before merging code from a downstream branch into an upstream branch.

A pull request is initiated in the GitHub GUI.

To enforce pull requests before merge

1. Go to Settings > Branches > Add Rule.
2. Choose "Require pull request reviews before merging."
3. Click "Create" to save the rule.

Note: some collaborator features may only be available with a GitHub Pro account.

Practice: Approving and merging pull requests

1. Push your changes to your branch in GitHub.
2. Create a pull request for your branch in GitHub.
3. Have your teammate review and approve your pull request.
4. Have your teammate merge your pull request into an upstream branch in the repo.
5. View the result in GitHub.

Handling merge conflicts

In spite of your best efforts, you will eventually encounter a merge conflict. This occurs when Git cannot figure out how to merge two files because the changes are too disparate.

1. Open the conflicted file in a text editor or VS Code or whatever IDE you are using.
2. Notice the merge conflict markers in the file.
   - <<<<<<< HEAD text in HEAD version ======= text in your version >>>>>>> your-branch
3. Select the text you want to keep and
4. Remove the text you want to delete.
5. Delete the conflict markers.
6. Save the adjusted file.
7. Commit and push to GitHub.
8. Go to GitHub and confirm that your branch now has the corrected file.

Additional Git resources

- **Installing git:** `https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`
- **git cheat sheet:** `https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf`
- **GitHub documentation:** `https://docs.github.com/en/github`
- **Git workflow:** `https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`
- **What to do if it goes all wrong:** `https://dangitgit.com/`
- **Resolving merge conflicts:** `https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line`

Capstone

Capstone Projects

Intent:

- Based on a direct observation or need expressed by a guest speakers.
- Original work demonstrating the skills obtained in this class.
- Demonstrate the ability to work as part of a team.
- A pitch to potential employers or investors.

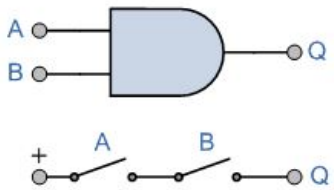
Guidelines:

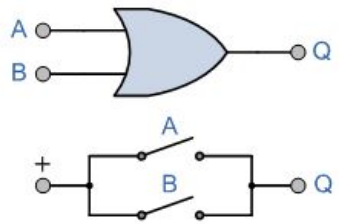
- Break class into 3 or 4 teams.
- Practical application of smart home, manufacturing, community environment, or immersive entertainment.
- Code MUST follow the IoT Style Guide
- Needs to include a Cloud Dashboard component.
- Project will include a video, presentation, GitHub, and Hackster.io.

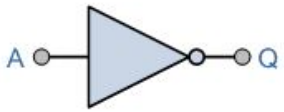
Previous capstone projects can been found at: `https://www.youtube.com/playlist?list=PL0t2Pk5ETDgxfVptdyr6xbL6MW1-5CJey`

BONUS - CMOS

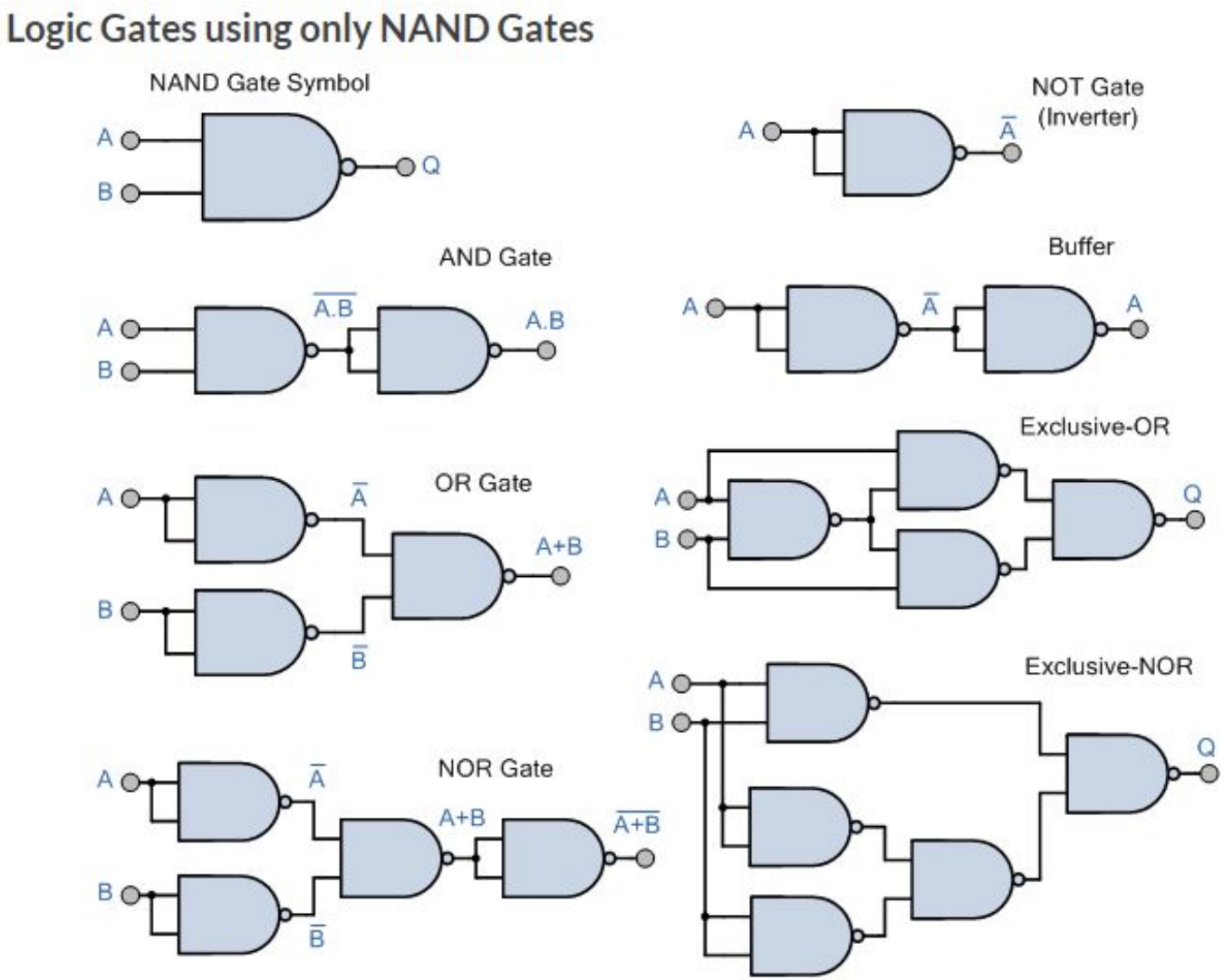
AND, OR, and NOT

Symbol	Truth Table		
	B	A	Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1

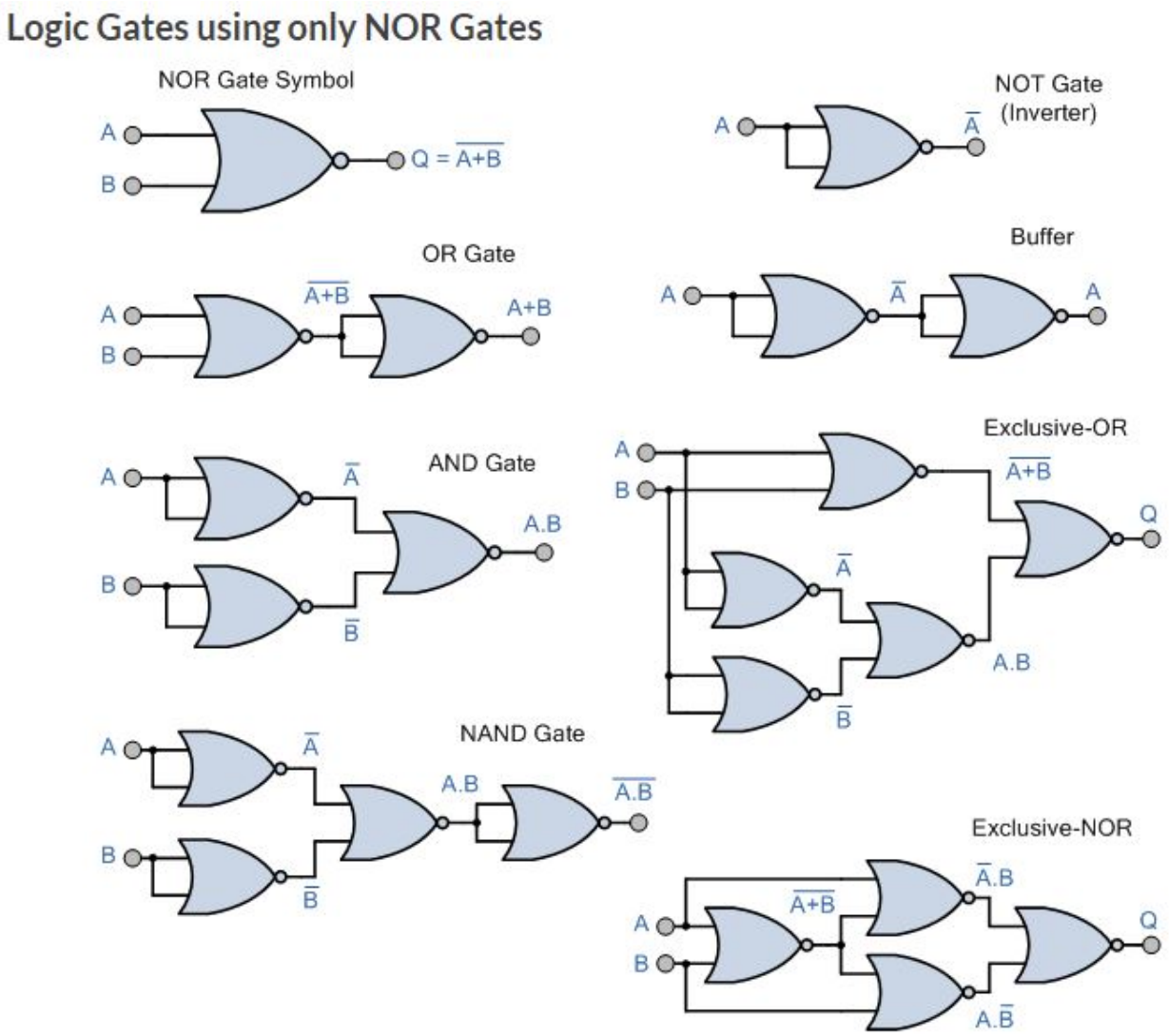
Symbol	Truth Table		
	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	1

Symbol	Truth Table	
	A	Q
	0	1
	1	0

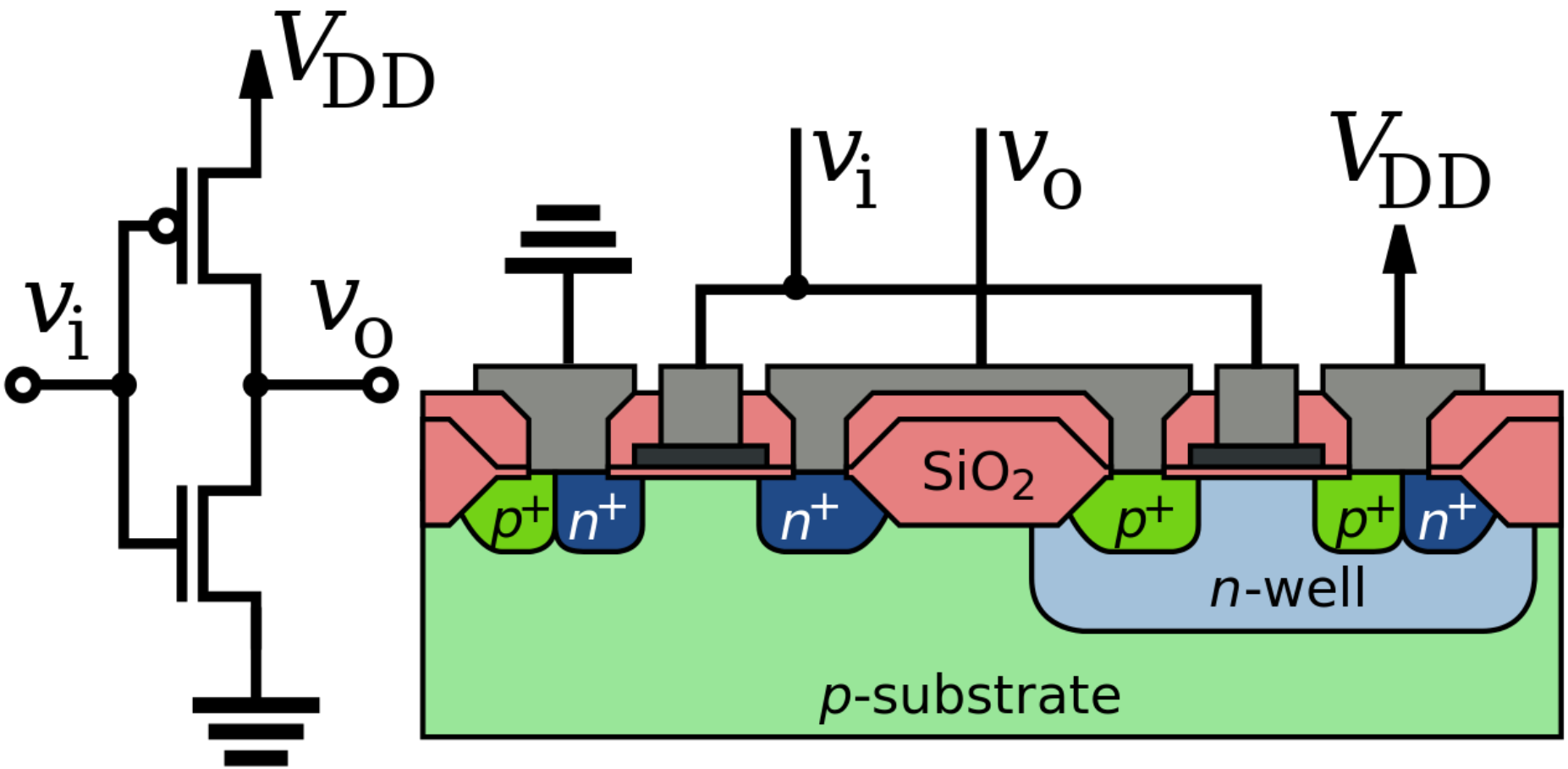
NAND


Logic Gates using only NAND Gates
NAND Gate Symbol
A
B
Q
NOT Gate
(Inverter)
A
$\overline{A}$
AND Gate
A
B
$\overline{A.B}$
A.B
Buffer
A
$\overline{A}$
A
OR Gate
A
B
$\overline{A}$
$\overline{B}$
A+B
Exclusive-OR
A
B
Q
NOR Gate
A
B
$\overline{A}$
$\overline{B}$
A+B
$\overline{A+B}$
Exclusive-NOR
A
B
Q

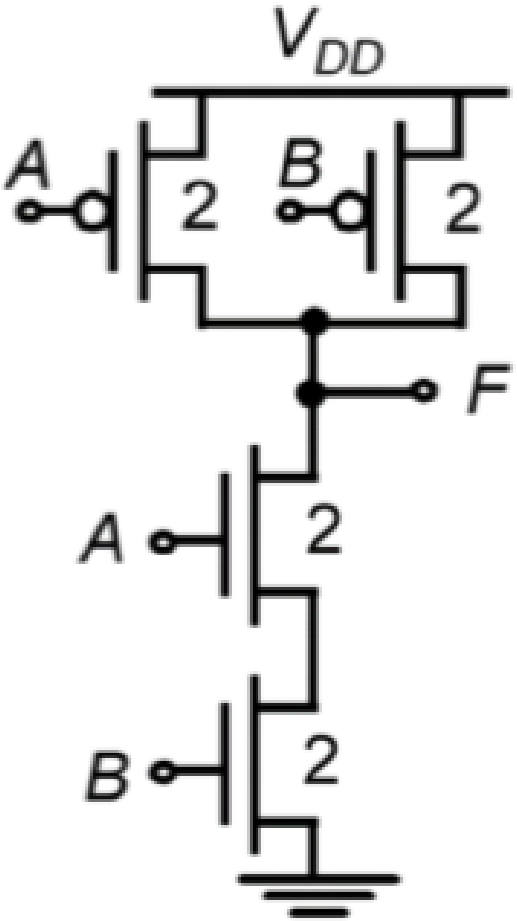
NOR

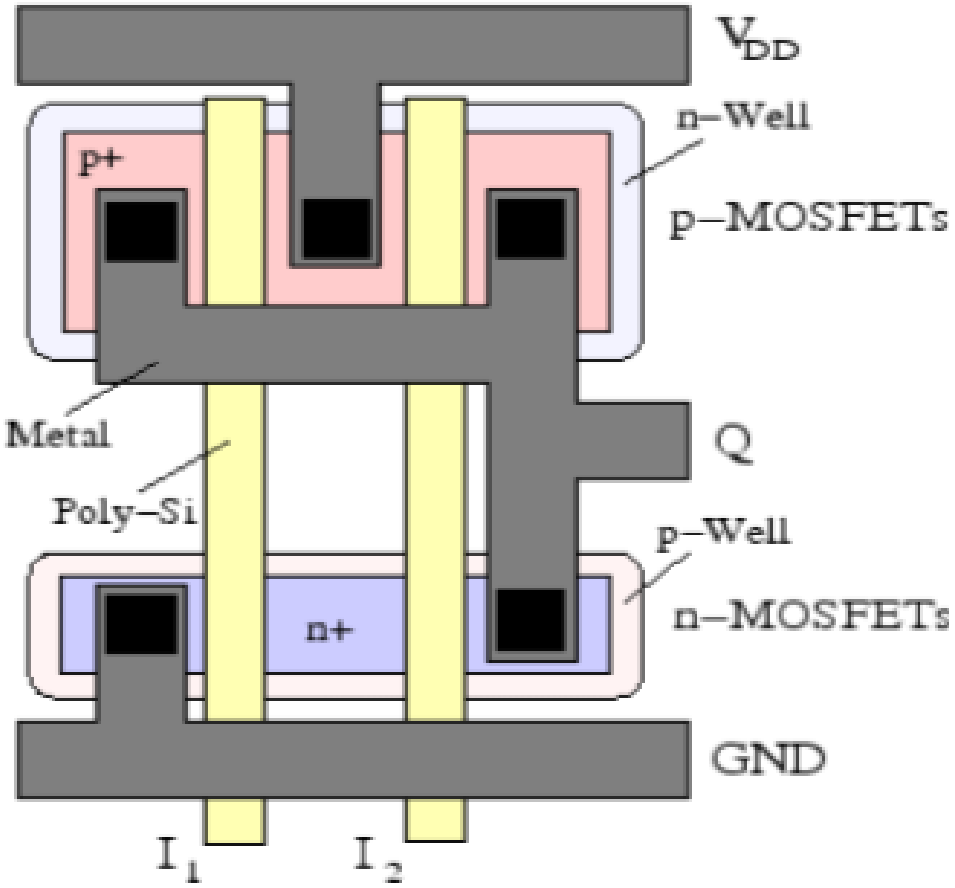

Logic Gates using only NOR Gates
NOR Gate Symbol
A
B
Q = A+B
NOT Gate
(Inverter)
A
Ā
OR Gate
A
B
A+B
A+B
Buffer
A
Ā
A
AND Gate
A
B
Ā
B̄
A.B
Exclusive-OR
A
B
A+B
Ā
B̄
A.B
Q
NAND Gate
A
B
Ā
B̄
A.B
A.B
Exclusive-NOR
A
B
A+B
A.B
A.B
Q

CMOS Inverter



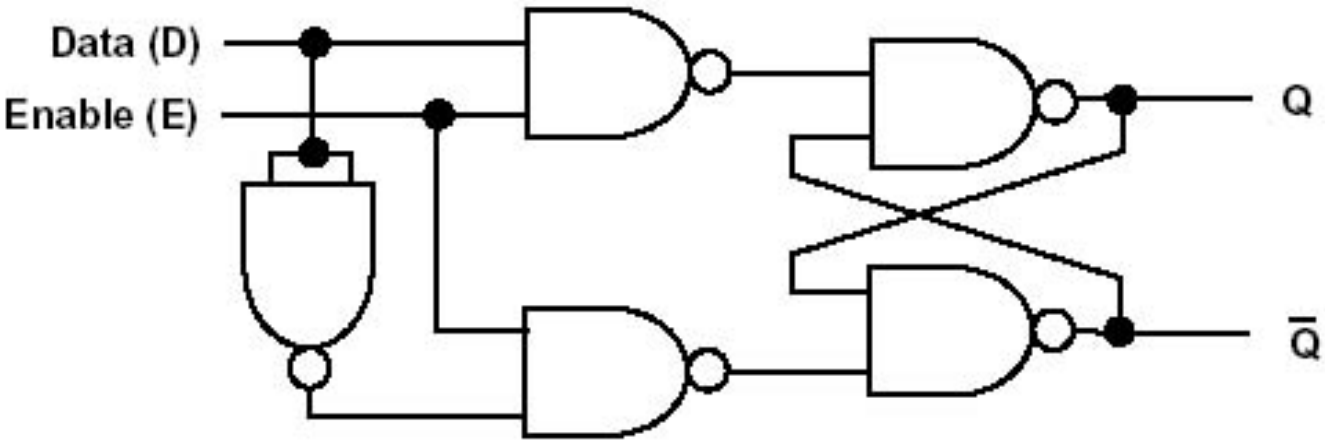
CMOS NAND


V_{DD}
A
B
2
2
F
A
2
B
2


V_{DD}
n–Well
p+
p–MOSFETs
Metal
Q
Poly–Si
p–Well
n+
n–MOSFETs
GND
I_1
I_2

CPU Registers - D Flip Flop

Remember the circuit you made on Day 1. Here's a more complex version that is used a registers (ultra-fast memory in the CPU):



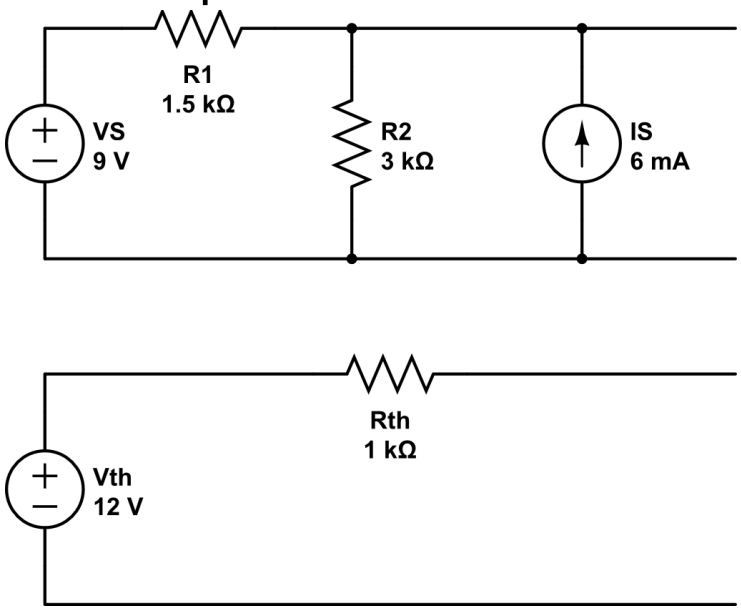

WORK
IN PROGRESS

Thevenin Equivalent Circuits

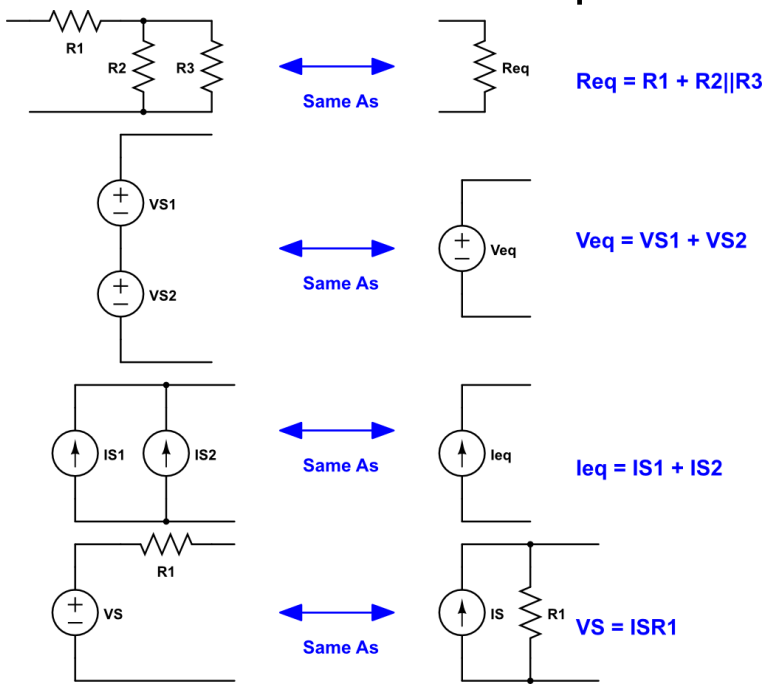
Léon Charles Thévenin (30 March 1857 – 21 September 1926) was a French telegraph engineer who extended Ohm's law to complex circuits.

Any combination of batteries and resistances with two terminals can be replaced by a single voltage source V_{th} and a single series resistor R_{th}.

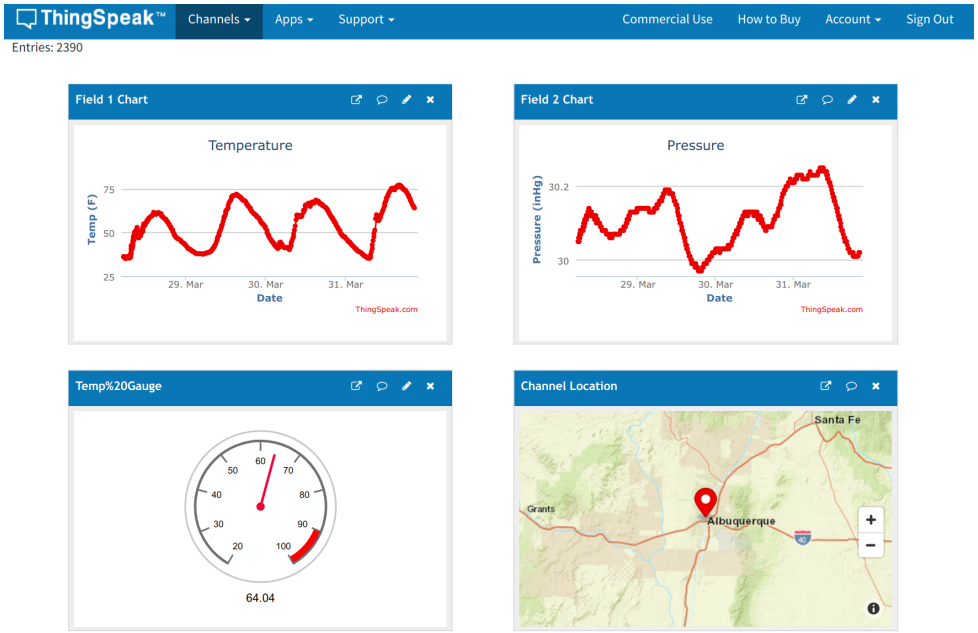
Equivalent Circuit



Useful relationships



ThingSpeak Via Webhooks



We will be learning how to use Particle Webhooks to publish between cloud services. For this, you will need:

1. Particle Argon
2. BME280

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
JSON is built on two structures:

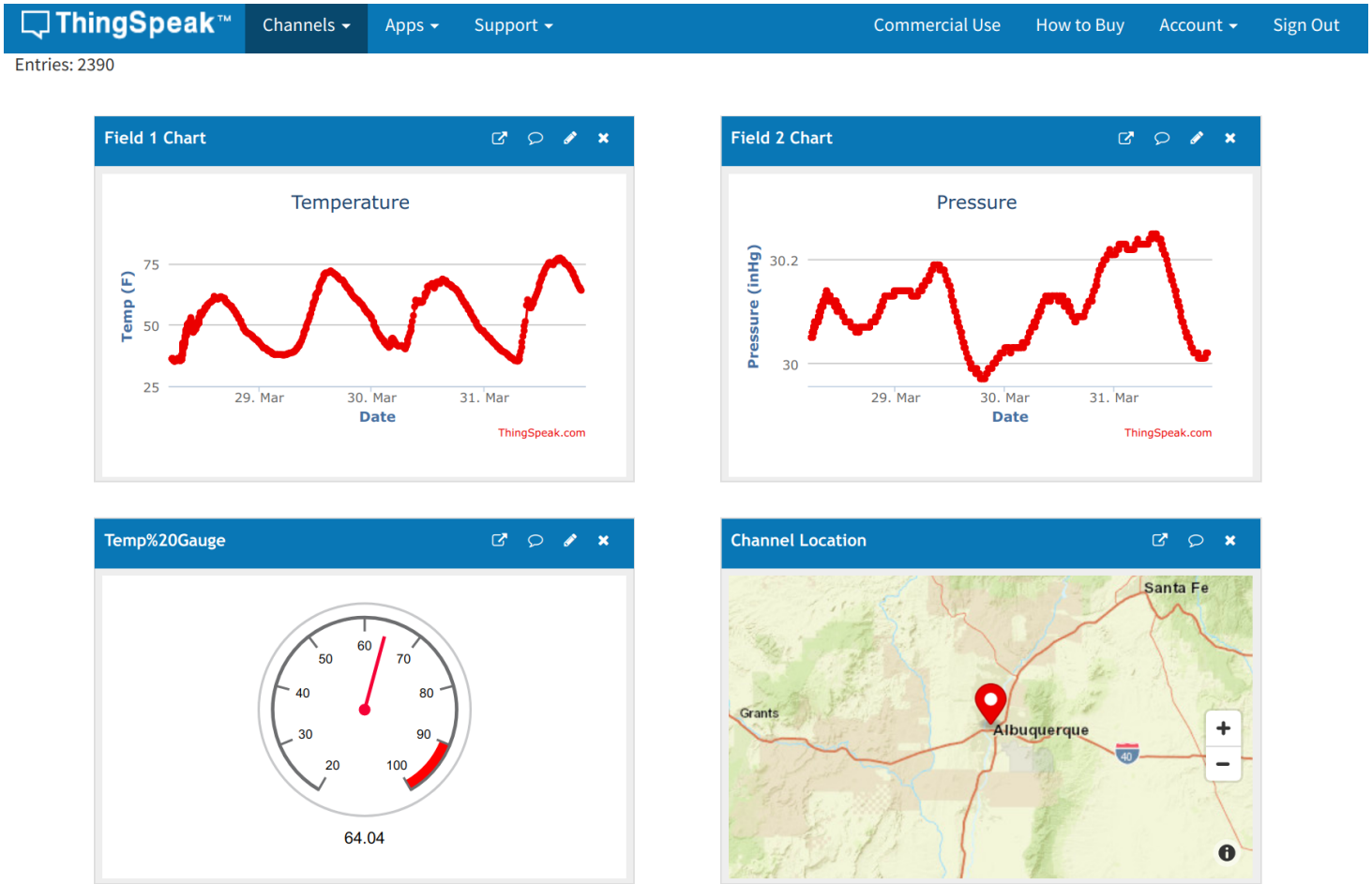
- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

JSON Parser Generator

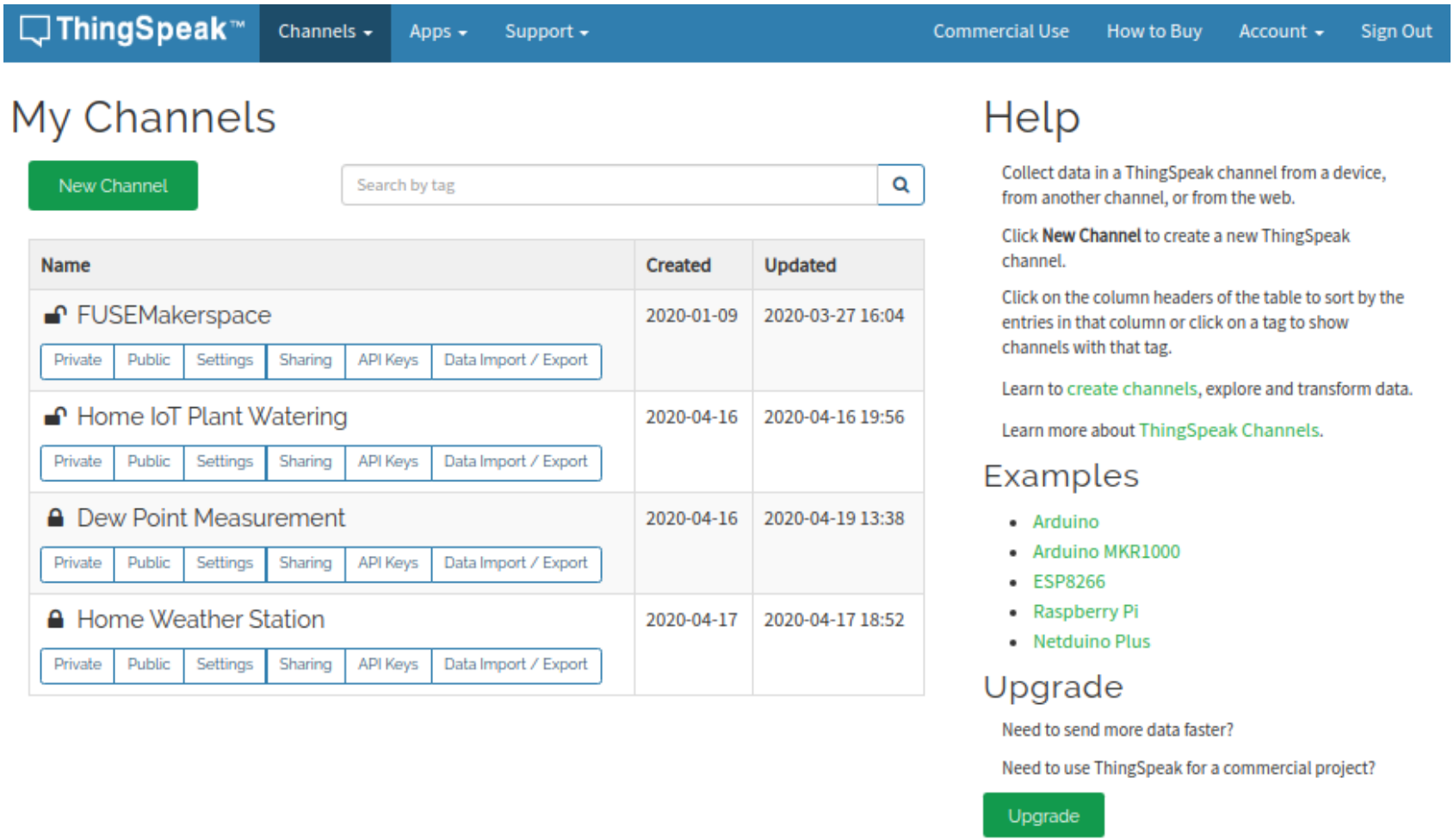
Creating objects in JSON are straightforward but can be tedious. There is a JSON Parser available to simplify the process.

```
#include <JsonParserGeneratorRK.h>

void createEventPayLoad(int moistValue, float tempValue, float presValue, float humValue,
      int waterED) {
  JsonWriterStatic<256> jw;
  {
    JsonWriterAutoObject obj(&jw);

    jw.insertKeyValue("Moisture", moistValue);
    jw.insertKeyValue("Temperature", tempValue);
    jw.insertKeyValue("Pressure", presValue);
    jw.insertKeyValue("Humidity", humValue);
    jw.insertKeyValue("Plant Watered", waterED);
  }
  Particle.publish("env-vals",jw.getBuffer(), PRIVATE);
}
```

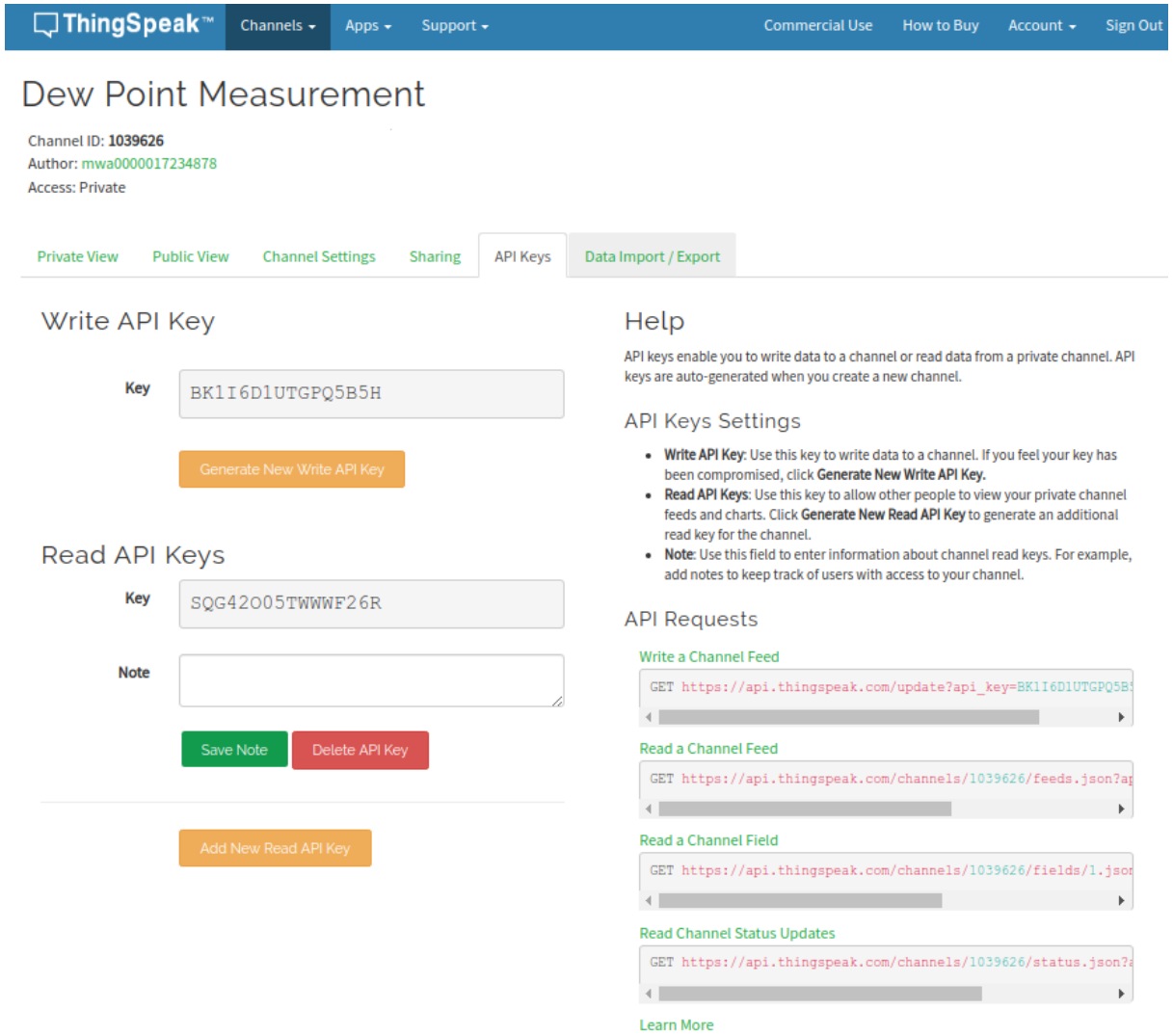
ThingSpeak Dashboard



Step 1 - Create ThingSpeak Channel

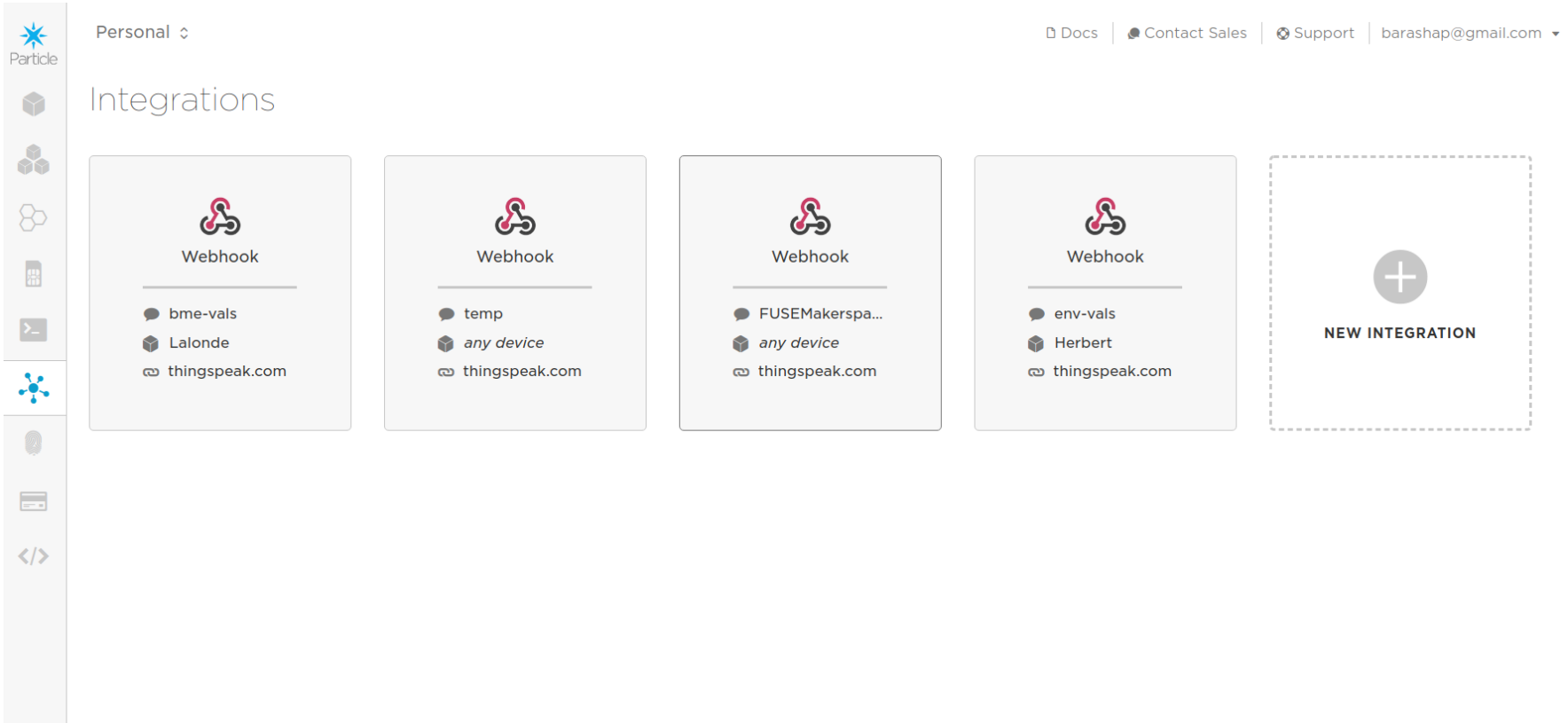


Step 2 - Get API Key

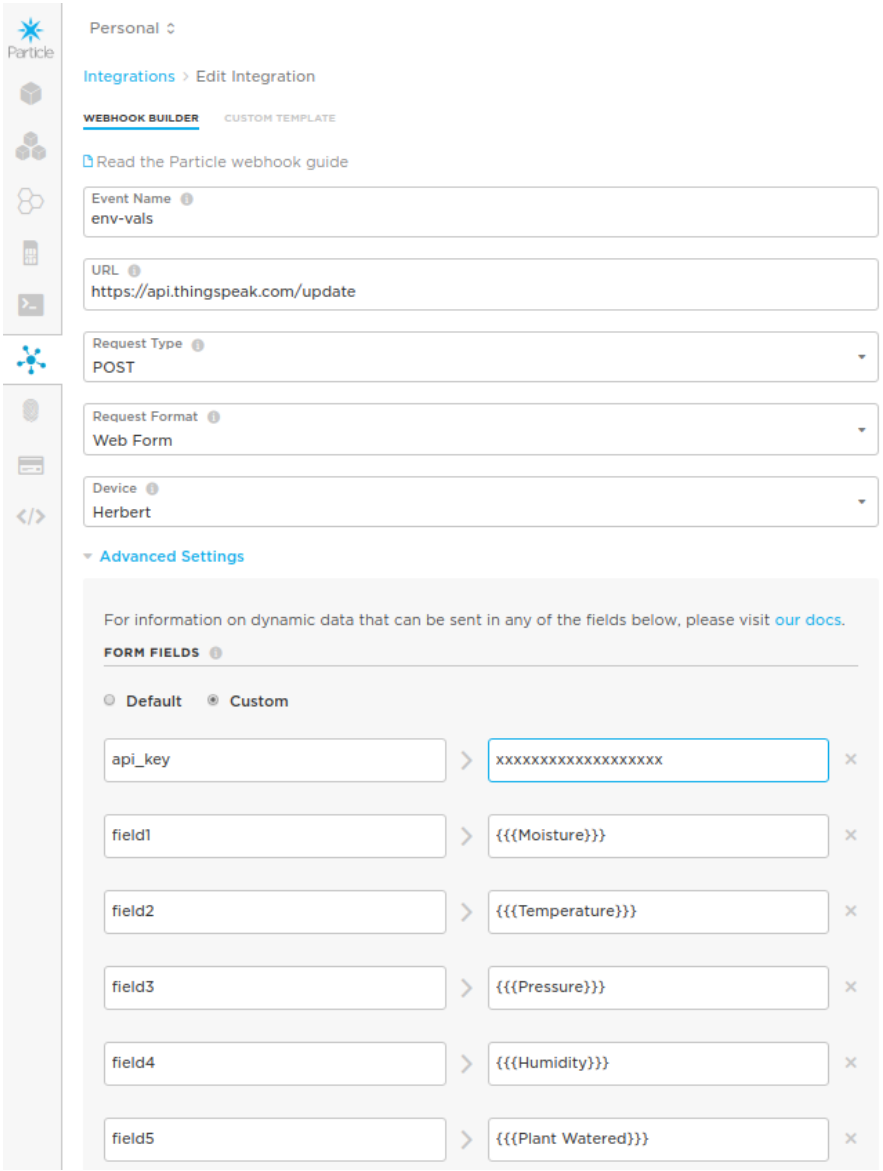


Step 3 - Create Webhook

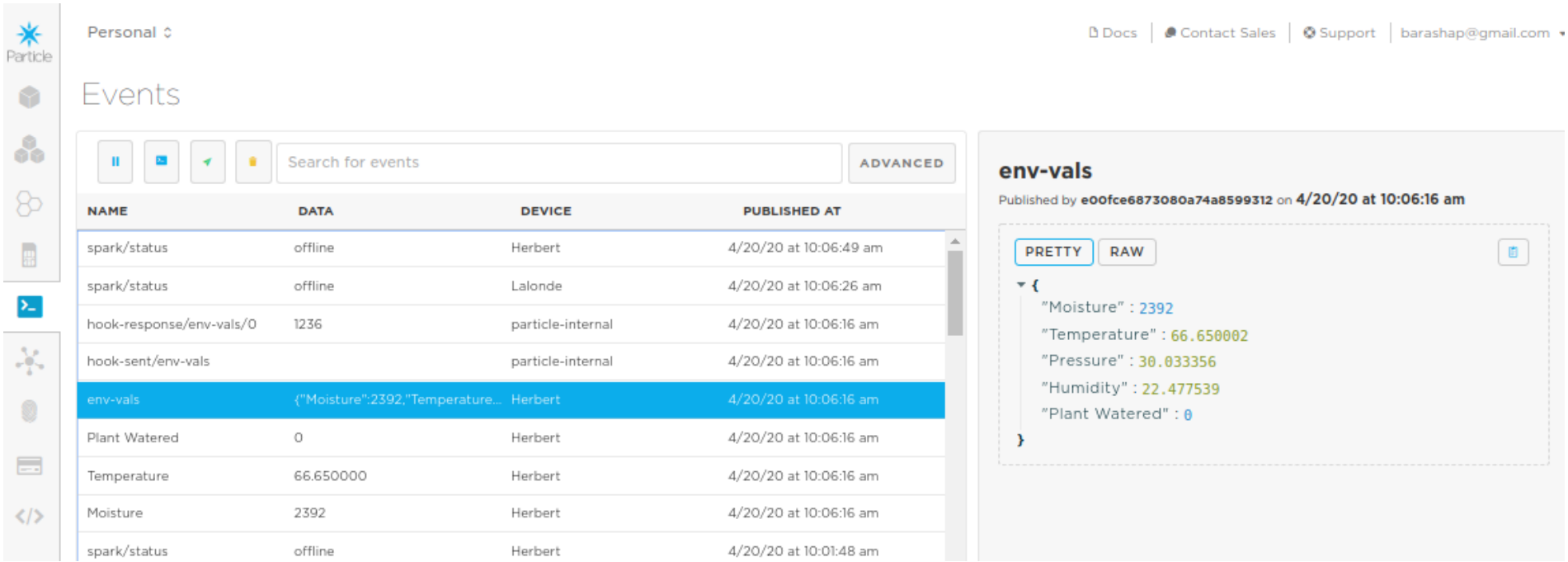
From `console.particle.io`:



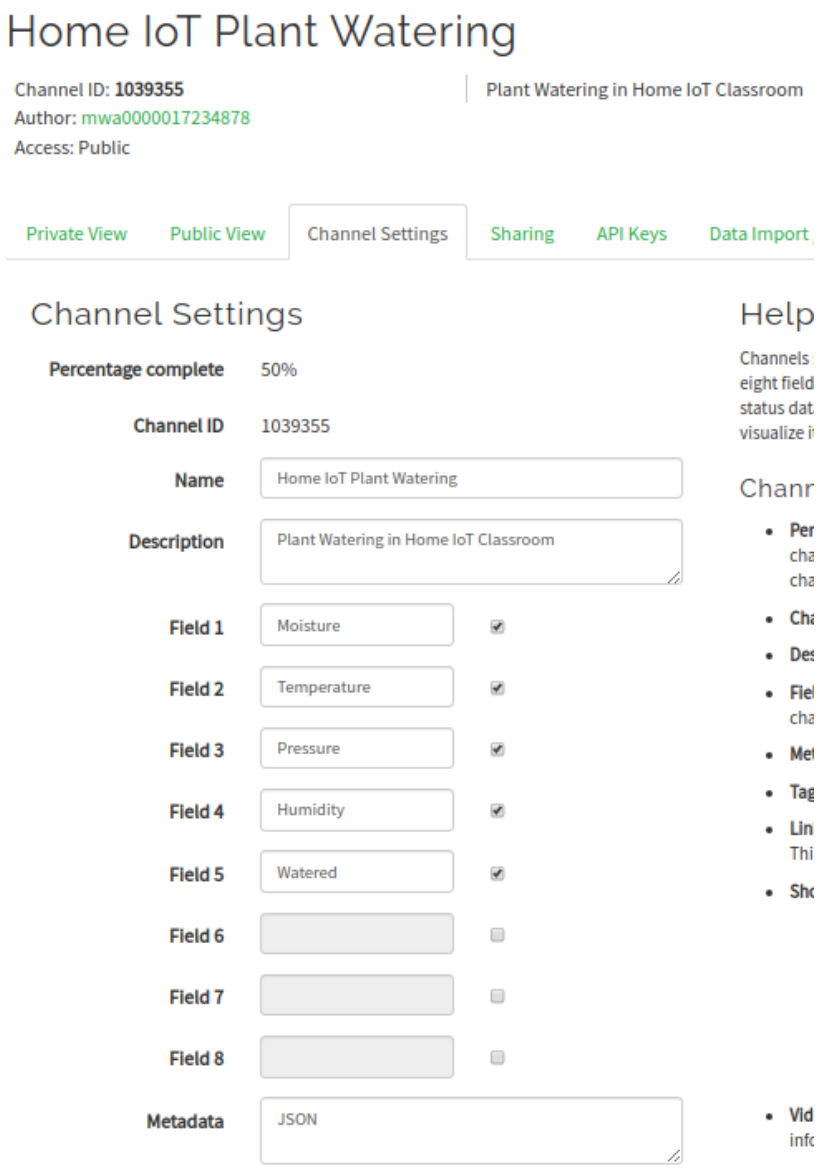
Step 4 - Add JSON Data



Step 5 - Particle Cloud Events

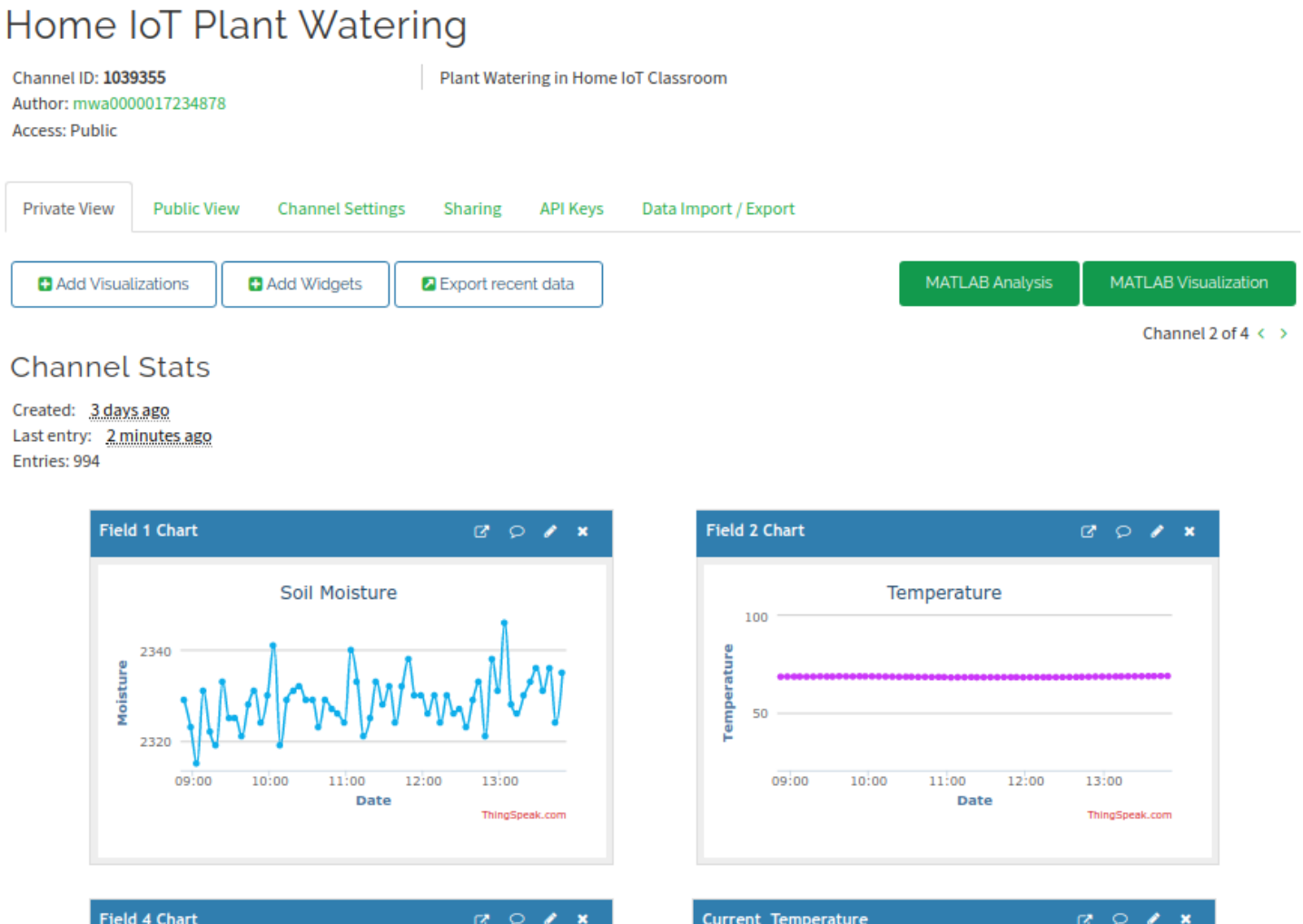


Step 6 - Create Channel

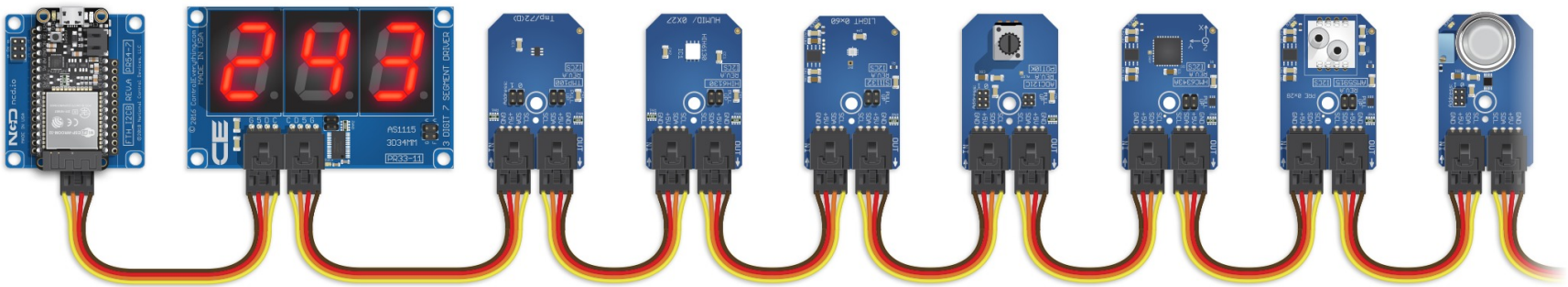


Step 7 - Create Dashboard

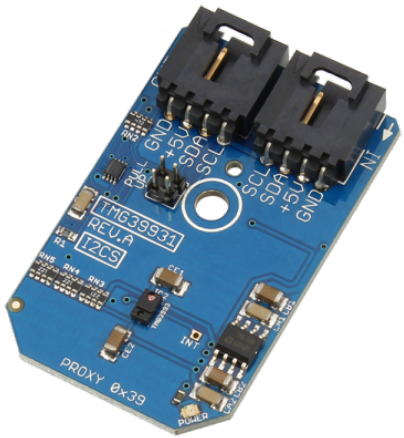
You can change the colors of your lines, by editing the graph. The hex codes are found at `https://htmlcolorcodes.com/color-picker/`



NCD.io ControlEverywhere



NCDio TMG3993 Proximity/Color Sensor



- From Address 0x94
- Request 9 bytes
  - Byte 0 / 1 - LSB / MSB of Infrared Luminance
  - Byte 2 / 3 - LSB / MSB of Red Luminance
  - Byte 4 / 5 - LSB / MSB of Green Luminance
  - Byte 6 / 7 - LSB / MSB of Blue Luminance
  - Byte 9 - Proximity

TMG3993 Initialization

```
  Serial.println("Initializing TMG3993");
  Wire.beginTransmission(Addr);
  // Select Enable register
  Wire.write(0x80);
  // Power ON, ALS enable, Proximity enable, Wait enable
  Wire.write(0x0F);
  Wire.endTransmission();

  Wire.beginTransmission(Addr);
  // Select ADC integration time register
  Wire.write(0x81);
  // ATIME : 712ms, Max count = 65535 cycles
  Wire.write(0x00);
  Wire.endTransmission();

  Wire.beginTransmission(Addr);
  // Select Wait time register
  Wire.write(0x83);
  // WTIME : 2.78ms
  Wire.write(0xFF);
  Wire.endTransmission();

  Wire.beginTransmission(Addr);
  // Select control register
  Wire.write(0x8F);
  // AGAIN is 1x
  Wire.write(0x00);
  Wire.endTransmission();
}
```

NCDio ACD121C MQ9 CO Sensor

12-Bit Analog to Digital Conversion

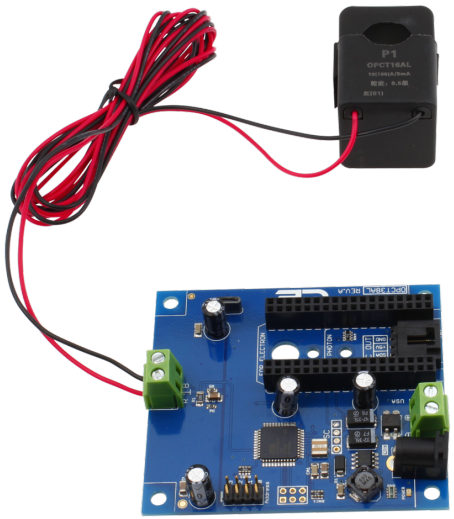
- From Address 0x00
- Request 2 bytes (raw_adc)
  - Byte 0 - MSB
  - Byte 1 - LSB
- CO (ppm) $= \frac{1000}{4096} * raw\_adc + 10$

NCDio ACD121C MQ131 Ozone Sensor

12-Bit Analog to Digital Conversion

- From Address 0x00
- Request 2 bytes (raw_adc)
  - Byte 0 - MSB
  - Byte 1 - LSB
- O_3 (ppm) $= \frac{1.99}{4095} * raw\_adc + 0.01$

NCDio PECMAC Current Sensor



- 1 to 8 channels
- Full range between 10 and 100 amps
- Simple to use. Simply run an AC power wire through the opening of the current sensor. This controller will read the magnetic field inducted onto the current sensor and provide you with a real-world current measurement value that is 98 percent accurate (prior to calibration).