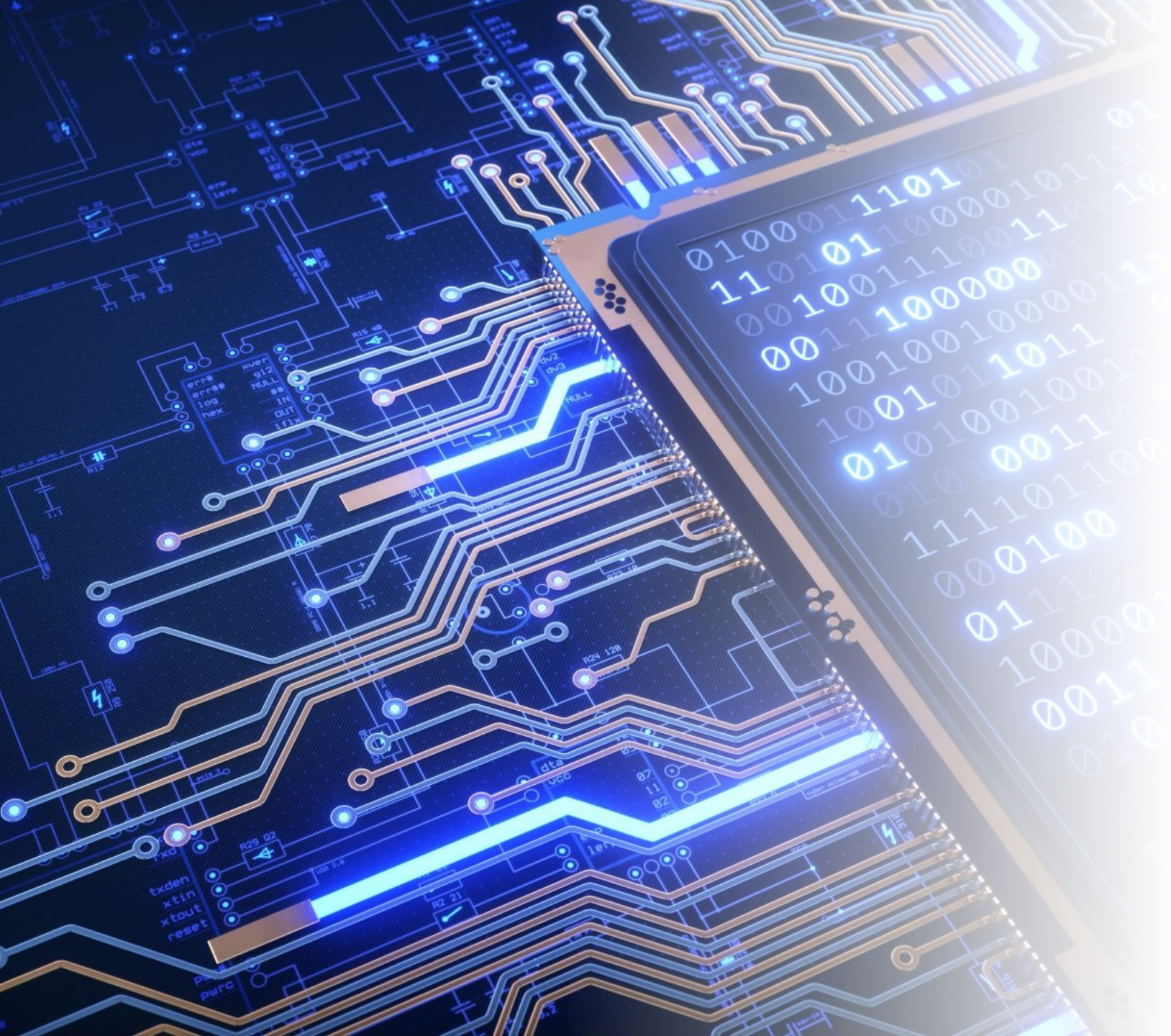


EEP 523: MOBILE APPLICATIONS FOR SENSING AND CONTROL

SUMMER 2021
Lecture 6

Tamara Bonaci

tbonaci@uw.edu

Agenda

- Review – practical introduction to deep learning
- TensorFlow Lite
- Introduction to Arduino
- Introduction to Circuit Playground board
- Android Connectivity
- Controlling Arduino with Android using:
  - Wi-FI
  - BLE

Review: Deep Learning Workflow in 7 steps

1. Decide on a goal
2. Collect a dataset
3. Design a model architecture
4. Train the model
5. Convert the model
6. Run inference
7. Evaluate and troubleshoot

1. Decide on the goal

- We can express our goal as a *classification* problem.

- Classification is a machine learning task that takes a set of input data and returns the probability that this data fits each of the known *classes*.

2. Collect Dataset

1. Select data

It is important that the data we choose is also available when you want to make predictions

2. Collect data

It is difficult to know exactly how much data is required to train an effective model. It depends on many factors, such as the complexity of the relationships between variables, the amount of noise, and the ease with which classes can be distinguished. However, there's a rule of thumb that is always true: the more data, the better!

3. Label data

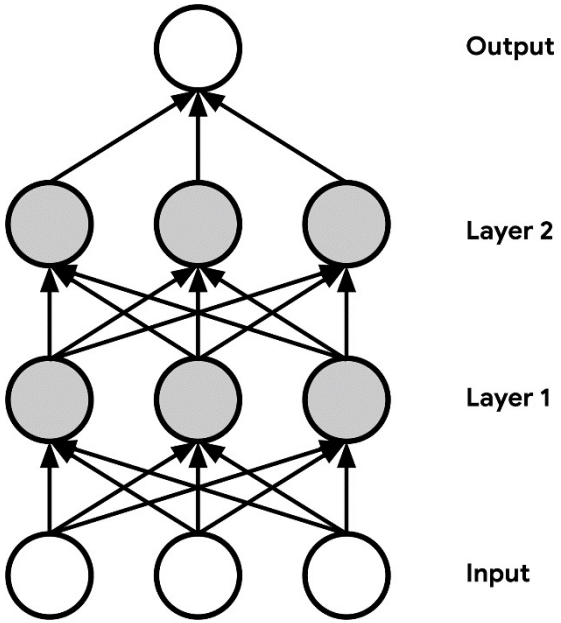
The process of associating data with classes is called labeling, and the "normal" and "abnormal" classes are our labels.

3. Design a Model Architecture

Model

- Network of simulated neurons represented by arrays of numbers arranged in layers.
- These numbers are known as ***weights*** and ***biases***, or collectively as the network's *parameters*.

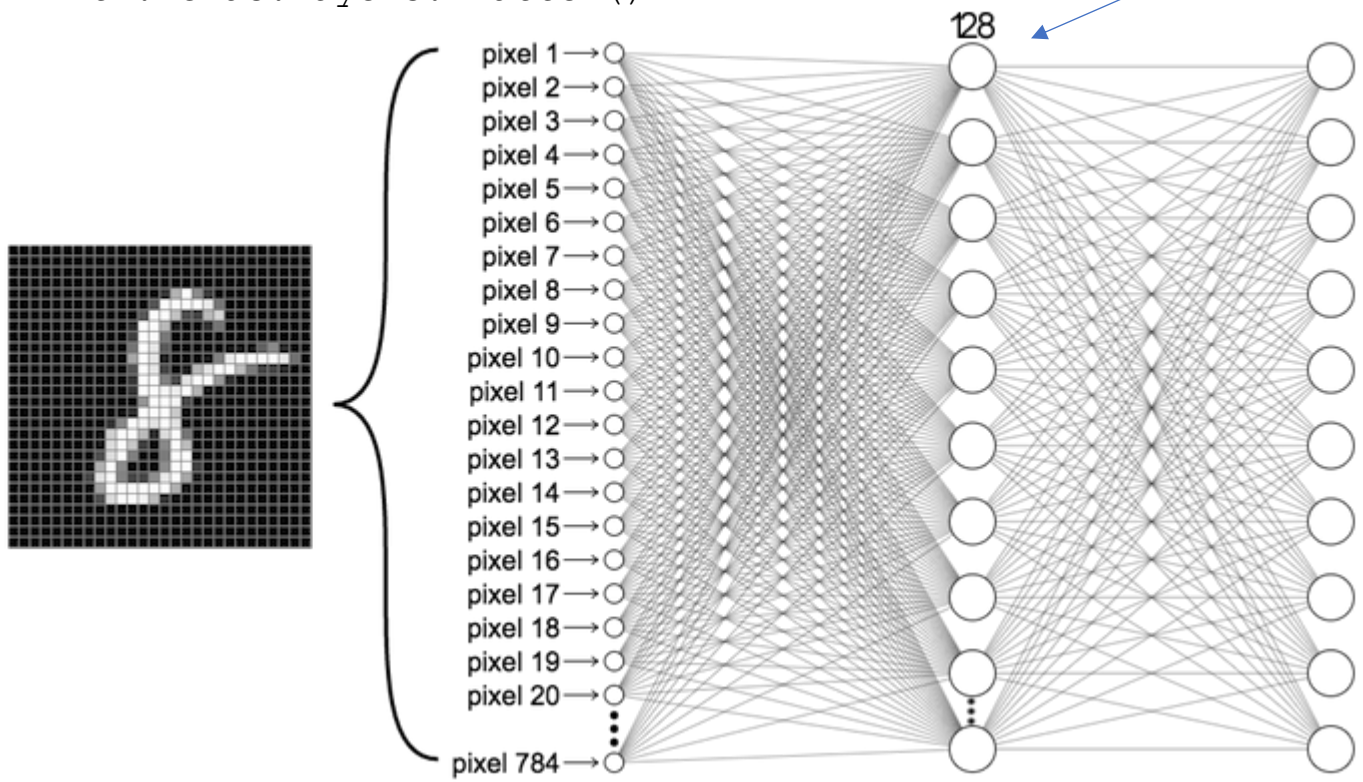
When data is fed into the network, it is transformed by successive mathematical operations that involve the weights and biases in each layer. The output of the model is the result of running the input through these operations.



Our Model

```
tf.keras.layers.Flatten()
```

```
tf.keras.layers
  .Dense(??, activation=tf.nn.relu)
```




```
tf.keras.layers.
          Dense(??, activation=tf.nn.softmax)
```

3. Design a Model Architecture

- For many common problems, we can find pretrained models available online for free.

- Deep learning models accept input and generate output in the form of **tensors**.
- For the purposes of this course, a tensor is essentially a list that can contain either numbers or other tensors; you can think of it as similar to an array.

4. Train the Model

- The model's weights start out with random values, and biases typically start with a value of 0.
- During training, batches of data are fed into the model, and the model's output is compared with the desired output (which in our case is the correct label, "normal" or "abnormal").
- An algorithm called *backpropagation* adjusts the weights and biases incrementally so that over time, the output of the model gets closer to matching the desired value.
- Training, which is measured in epochs (meaning iterations), continues until we decide to stop.

4. Train the Model: When to Stop?

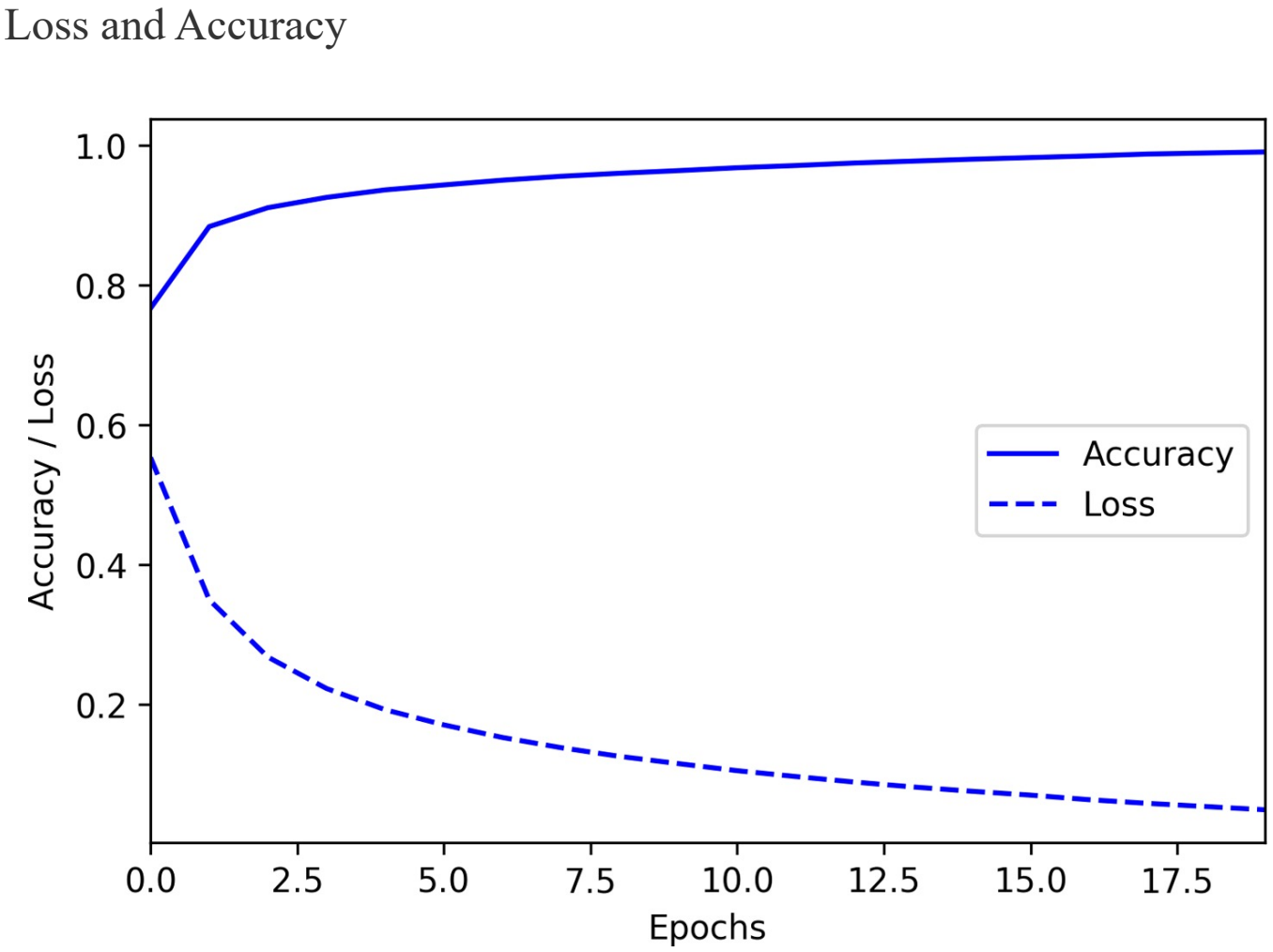
- We generally stop training when a model's performance stops improving. It is said to have ***converged***.

- Common performance metrics are ***loss*** and ***accuracy***.
  - **Loss**: numerical estimate of how far the model is from producing the expected answers

  - **Accuracy:** percentage of the time that it chooses the correct prediction.

  - A perfect model would have a loss of 0.0 and an accuracy of 100%, but real models are rarely perfect.

4. Train the Model:



4. Train the Model: Hyperparameters

- To attempt to improve the model's performance, we can change our model architecture, and we can adjust various values used to set up the model and moderate the training process.

- These values are collectively known as *hyperparameters*, and they include variables such as the number of training **epochs** to run and the number of neurons in each layer.

- Each time we make a change, we can retrain the model, look at the metrics, and decide whether to optimize further.

- Hopefully, time and iterations will result in a model with acceptable accuracy!

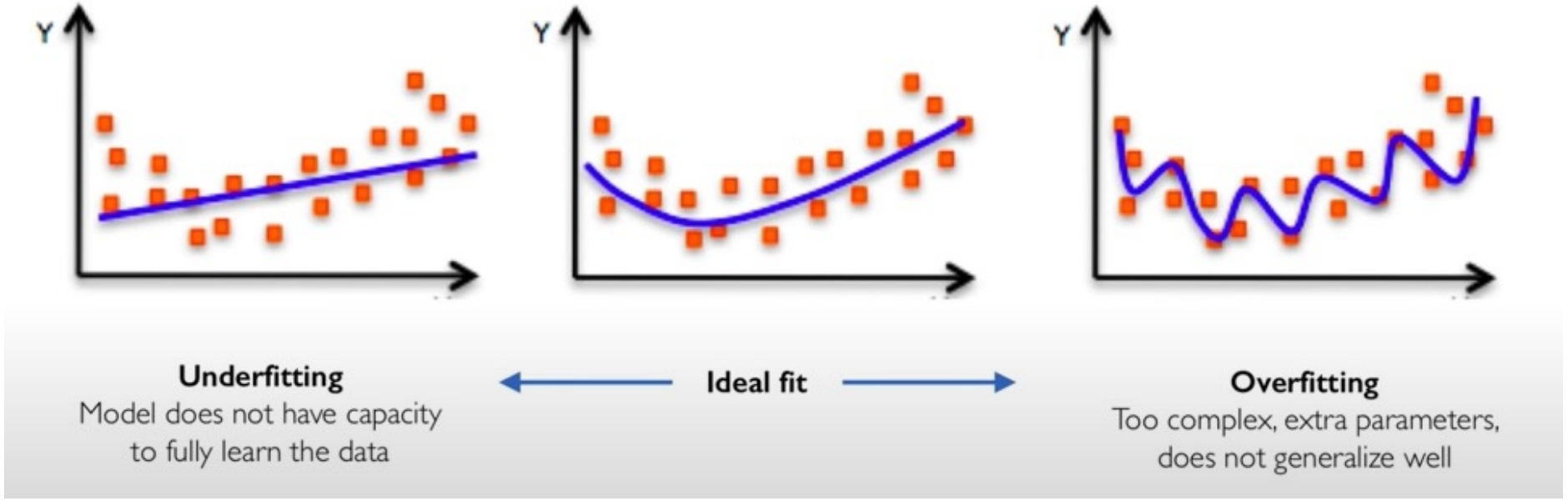
4. Train the Model: Why do Models Fail?

A neural network learns to *fit* its behavior to the patterns it recognizes in data

- ***Underfit***
  model has not yet been able to learn a strong enough representation of these patterns to be able to make good predictions.
  - X The architecture is too small to capture the complexity of the system it is supposed to model
  - X The model has not been trained on enough data.

- ***Overfit***
  it has learned its training data too well. The model is able to exactly predict the minutiae of its training data, but it is not able to generalize its learning to data it has not previously seen.
  - X the model has managed to entirely memorize the training data
  - X it has learned to rely on a shortcut present in the training data but not in the real world.

4. Train the model: Why do Models Fail?



4. Train the model: Preventing the model from failing

Goal: make deep learning models **less likely to overfit their training data**.

Regularization

- Generally, involves constraining the model in some way in order to prevent it from perfectly memorizing the data that it's fed during training.
- Example: *dropout* -> randomly cutting the connections between neurons during training.

Data augmentation,

- artificially expand the size of a training dataset.
- creating multiple additional versions of every training input,
  each transformed in a way that preserves its meaning
  but varies its exact composition

4. Train the model: train-validate-test

- need to *validate* the model using new data that wasn't used in training. It's common to split a dataset into three parts—***training*, *validation*, and *test*.**

- A typical split is **60% training data, 20% validation, and 20% test**. This splitting must be done so that each part contains the same distribution of information, and in a way that preserves the structure of the data.

1) During training, the *training* dataset is used to train the model.

2) Periodically, data from the *validation* dataset is fed through the model, and the loss is calculated. Because the model has not seen this data before, its loss score is a more reliable measure of how the model is performing.

3) By comparing the training and validation loss (and accuracy, or whichever other metrics are available) over time, you can see whether the model is overfitting.

Deploying machine learning models on mobile and IoT devices

TensorFlow

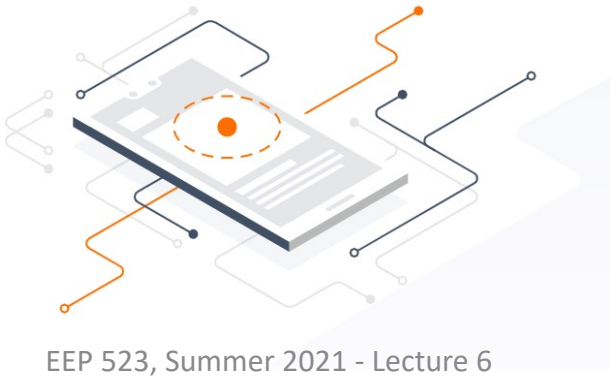
- a set of tools for building, training, evaluating, and deploying machine learning models.
- Originally developed at Google - now an open source project
- most popular and widely used framework for machine learning.

TensorFlow Lite

- set of tools for deploying TensorFlow models to mobile and embedded devices

Keras

- TensorFlow's high-level API that makes it easy to build and train deep learning networks. We'll also use.



5. Convert the model: Lite model

TensorFlow model:

- set of instructions that tell an interpreter how to transform data in order to produce an output.
- When we want to use our model, we just load it into memory and execute it using the **TensorFlow interpreter**.
- TensorFlow's interpreter is designed to run models on powerful desktop computers and servers
- TensorFlow Lite Converter. The converter can also apply special optimizations aimed at reducing the size of the model and helping it run faster, often without sacrificing performance.

6. Run inference

```
private fun classifyDrawing() {
    val bitmap = drawView?.getBitmap()

    if ((bitmap != null) && (digitClassifier.isInitialized)) {
        digitClassifier
            .classifyAsync(bitmap)
            .addOnSuccessListener { resultText -> predictedTextView?.text = resultText }
            .addOnFailureListener { e ->
                predictedTextView?.text = "classification error"
                Log.e(TAG, "Error classifying drawing.", e)
            }
    }
}
```

6. Run inference

STEPS TO USE THE TFLITE MODEL IN OUR APP

1. Load the TFLITE model from the asset folder – `loadModelFile()`

1. Initialize the interpreter - `initializeInterpreter()`

2. Prepare the data input in a byte buffer array

3. Run inference: interpreter.run(input, output)

See provided app for function implementation details

6. Run inference

- Since this is the part where our model meets our application code
- we need to write some code **that takes raw input data from our sensors and transforms it into the same form that our model was trained on**.
- We then pass this transformed data into our model and run inference.

TensorFlow Lite

EE P 523, Lecture 6

TensorFlow *Lite*

TensorFlow: end-to-end open-source platform for machine learning.

TensorFlow Lite: lightweight solution for mobile and embedded devices.

- It enables on-device machine learning inference with low latency and a small binary size.
- TensorFlow Lite also supports hardware acceleration with the Android Neural Networks API.

TensorFlow *Lite*

Why do we need a new mobile-specific library?

The next wave of machine learning applications will have significant processing on mobile and embedded devices.

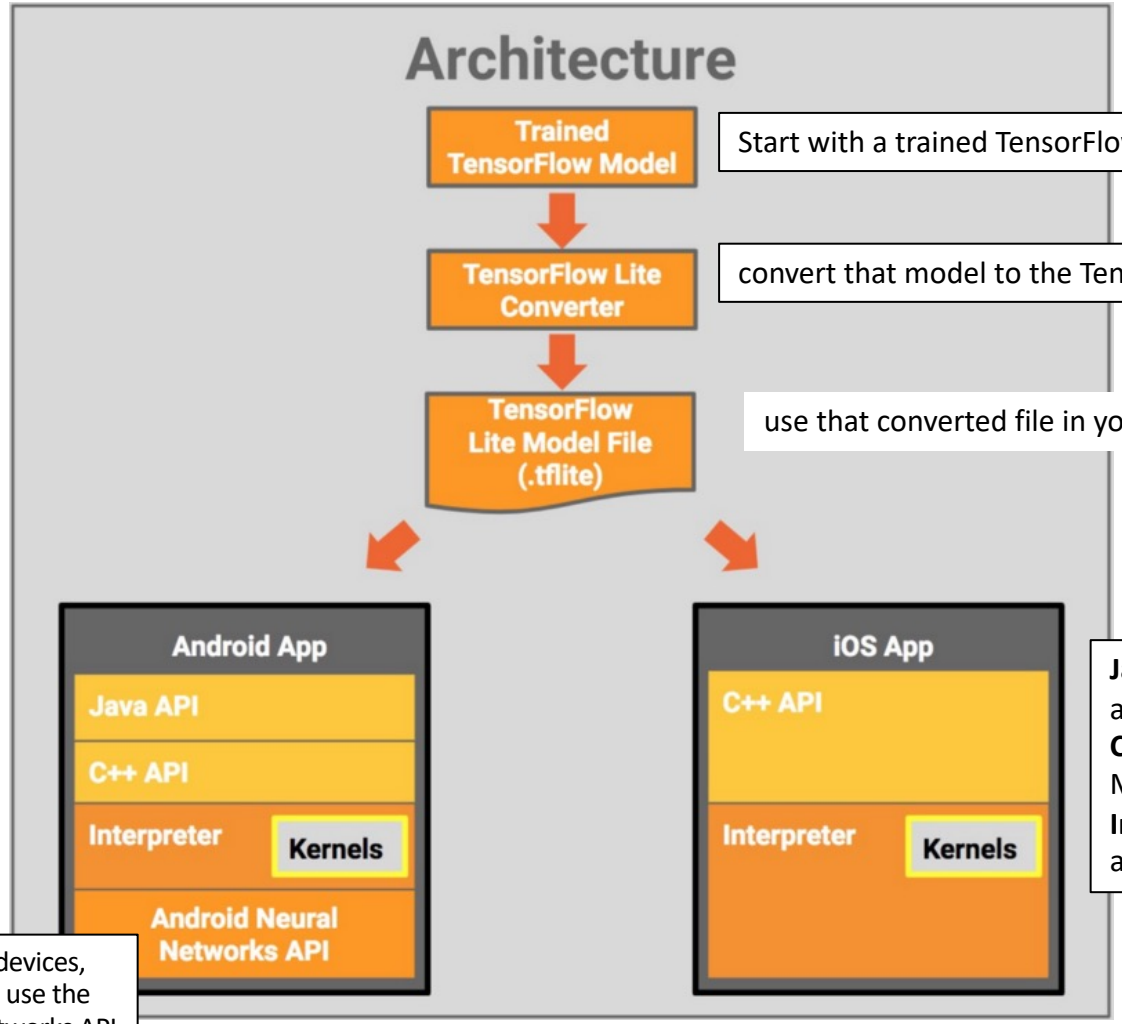
- Machine Learning is changing the computing paradigm.
- Consumer expectations are also trending toward natural, human-like interactions with their devices, driven by the camera and voice interaction models.

TensorFlow *Lite*

Why do we need a new mobile-specific library?

- Innovation at the *silicon layer* is enabling new possibilities for hardware acceleration, and frameworks such as the Android Neural Networks API make it easy to leverage these.
- Recent advances in real-time computer-vision and spoken language understanding have led to mobile-optimized benchmark models being open sourced (e.g., MobileNets, SqueezeNet).
- Widely-available smart appliances create new possibilities for on-device intelligence.
- Interest in stronger user data privacy paradigms where user data does not need to leave the mobile device.
- Ability to serve 'offline' use cases, where the device does not need to be connected to a network.

TensorFlow *Lite*



Start with a trained TensorFlow model on disk

convert that model to the TensorFlow Lite file format (.tflite)

use that converted file in your mobile application

Java API: A convenience wrapper around the C++ API on Android.
C++ API: Loads the TensorFlow Lite Model File and invokes the Interpreter
Interpreter: Executes the model using a set of kernels

On select Android devices, the Interpreter will use the Android Neural Networks API for hardware acceleration, or default to CPU execution if none are available.

Why is it Called TensorFlow?

- TensorFlow is called 'TensorFlow' because it handles the flow (node/mathematical operation) of tensors, which are data structures that you can think of as multi-dimensional arrays.

- Tensors are represented as n-dimensional arrays of base datatypes, such as a string or integer -- they provide a way to generalize vectors and matrices to higher dimensions.

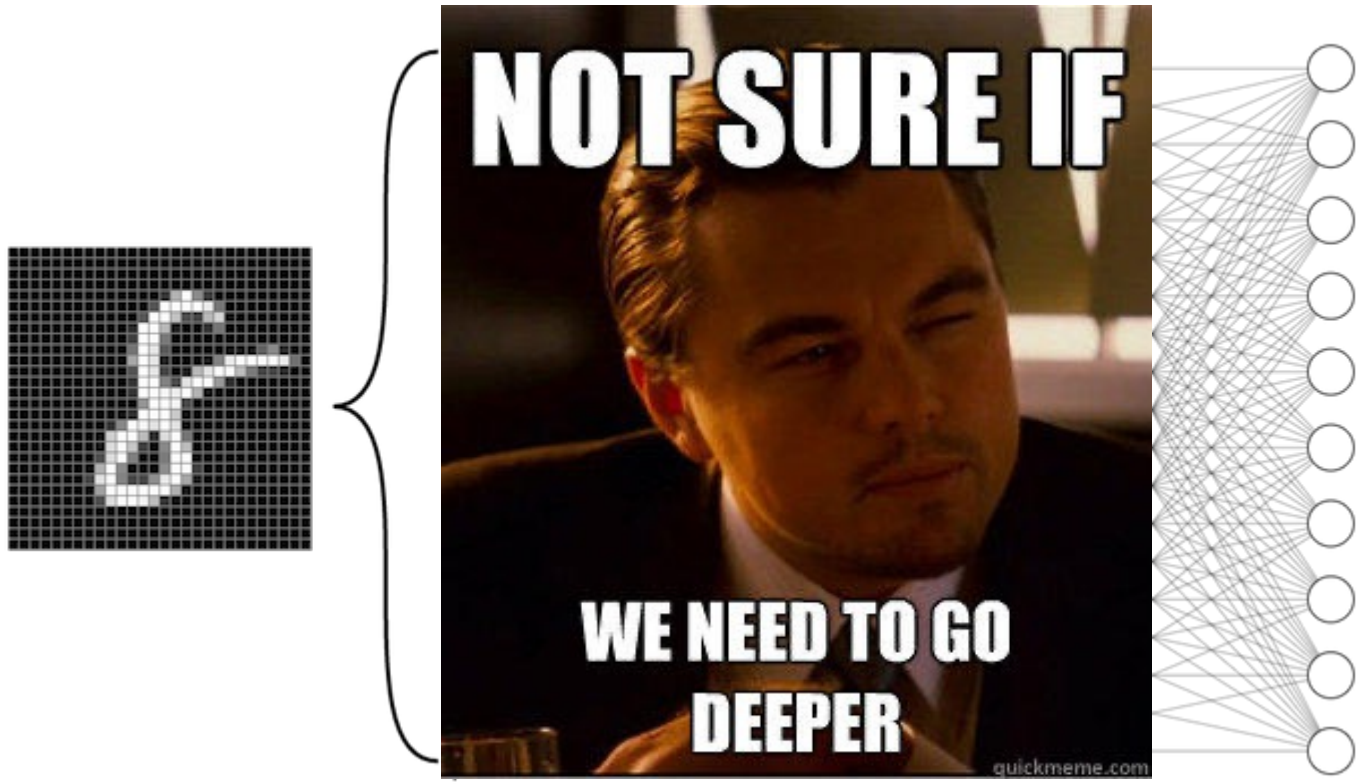
- The ```shape``` of a Tensor defines its number of dimensions, and the size of each dimension.

- The ```rank``` of a Tensor provides the number of dimensions (n-dimensions) -- you can also think of this as the Tensor's order or degree.Let's first look at 0-d Tensors, of which a scalar is an example:

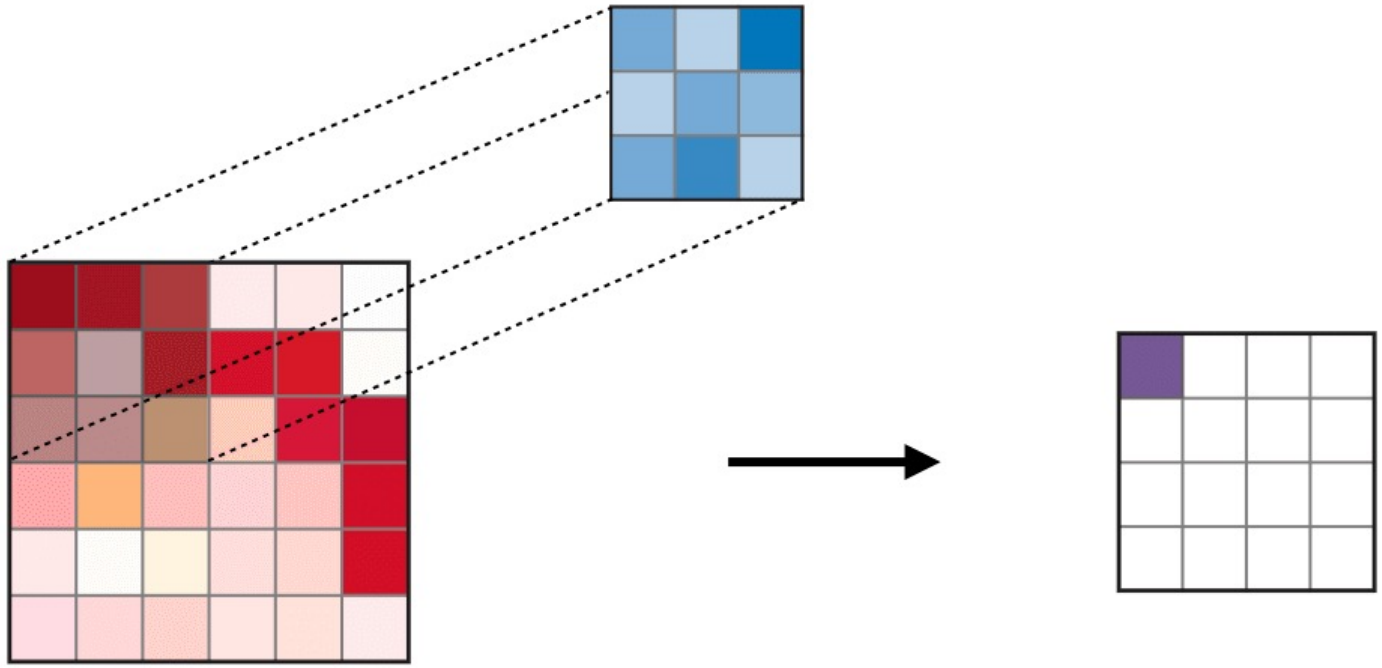
Improving Our Model

EE P 523, Lecture 6

How Do We improve the model?



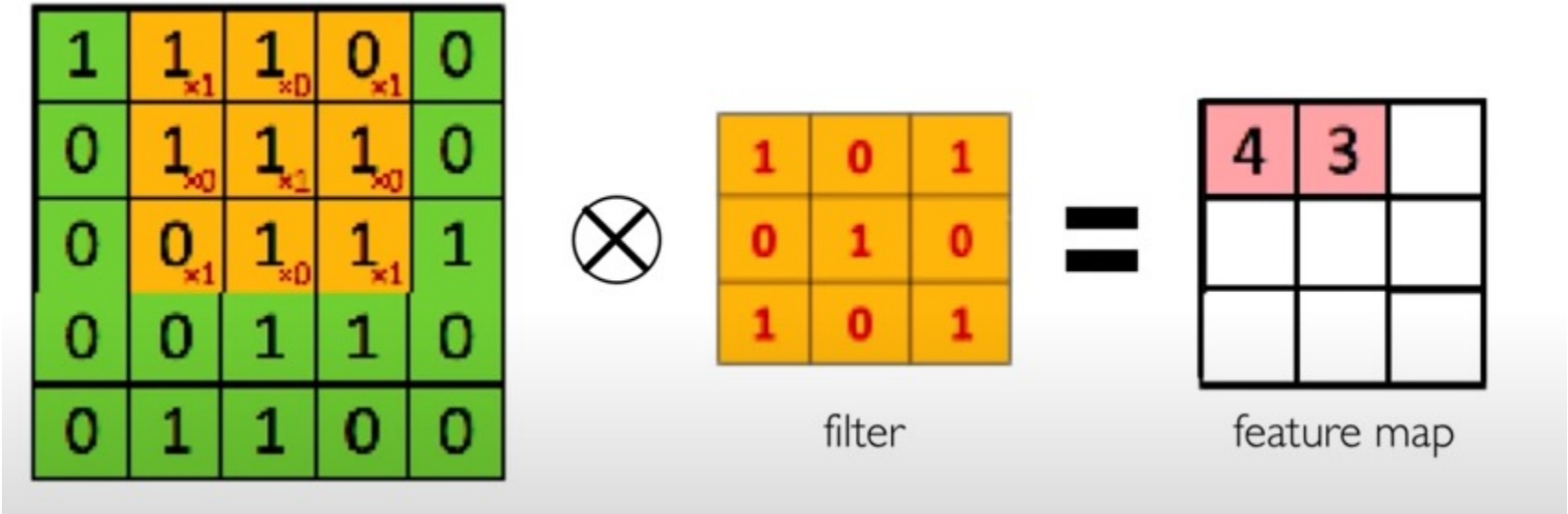
Convolution Operation



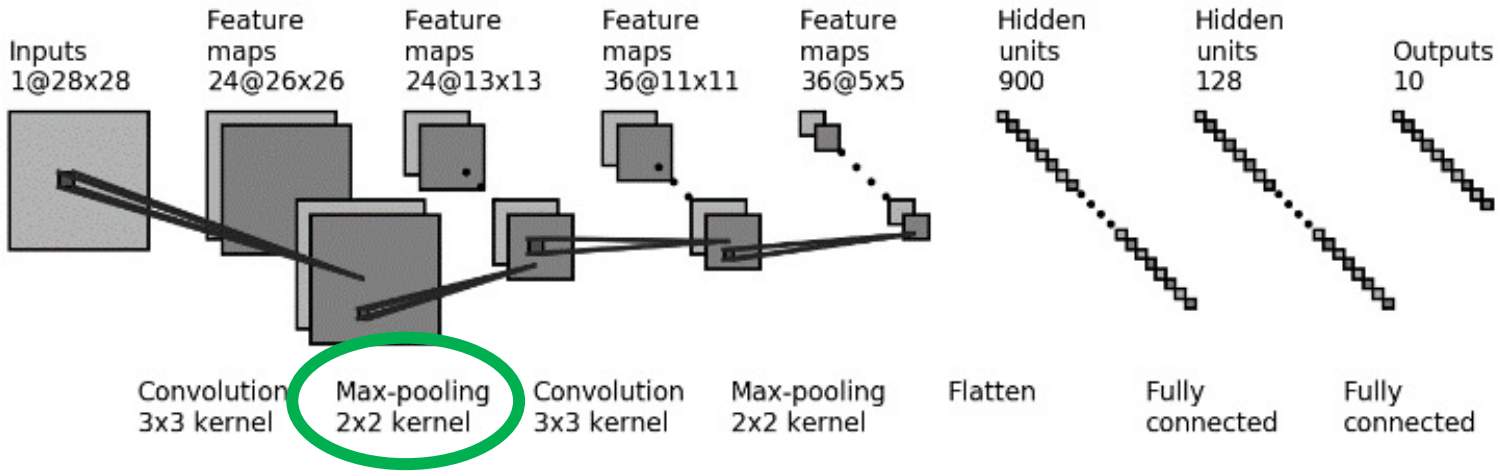
- Convolutional layers are used for **spotting 2D patterns in input images**.
- Each filter is a rectangular array of values that is moved as a sliding window across the input, and the output image is a representation of how closely

Convolution Operatoin

1)Slide a 3x3 filter 2)Element-wise multiply 3)Add the numbers

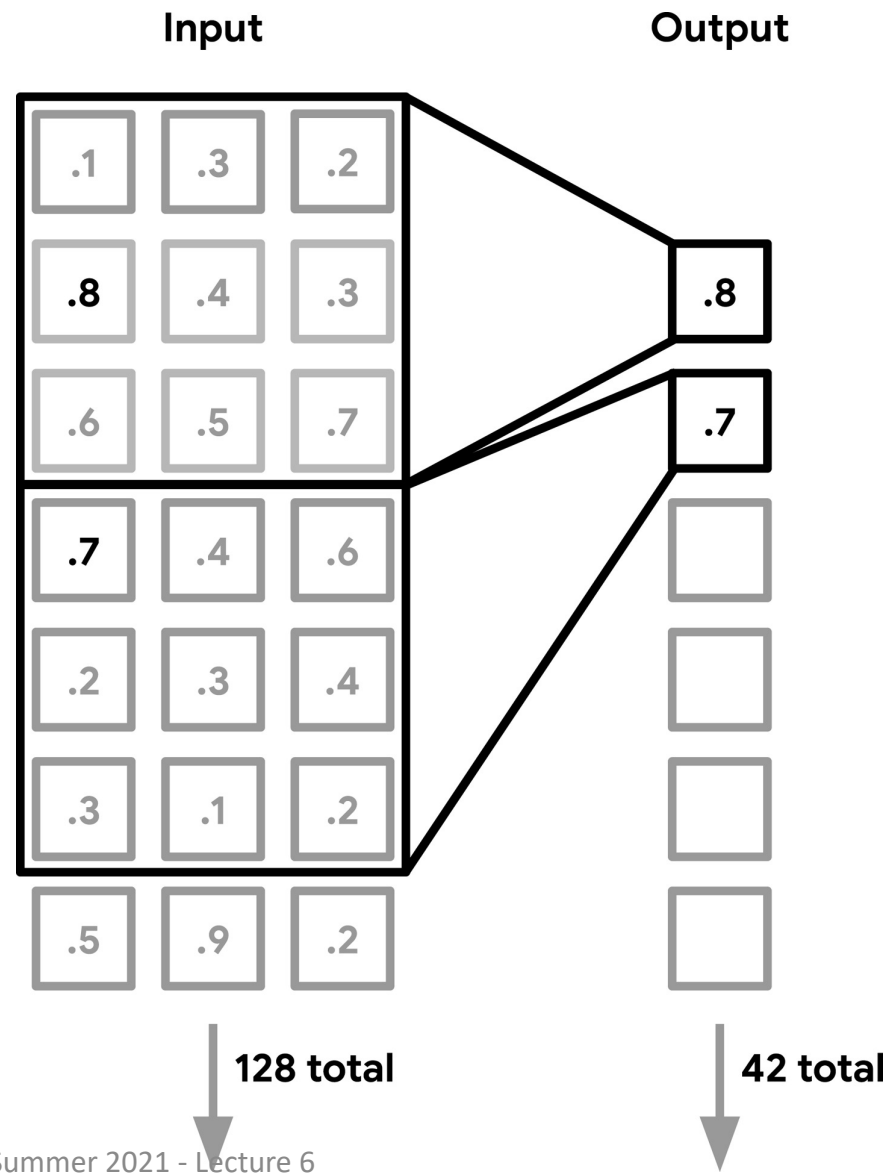


Convolutional Neural Network for digits recognition

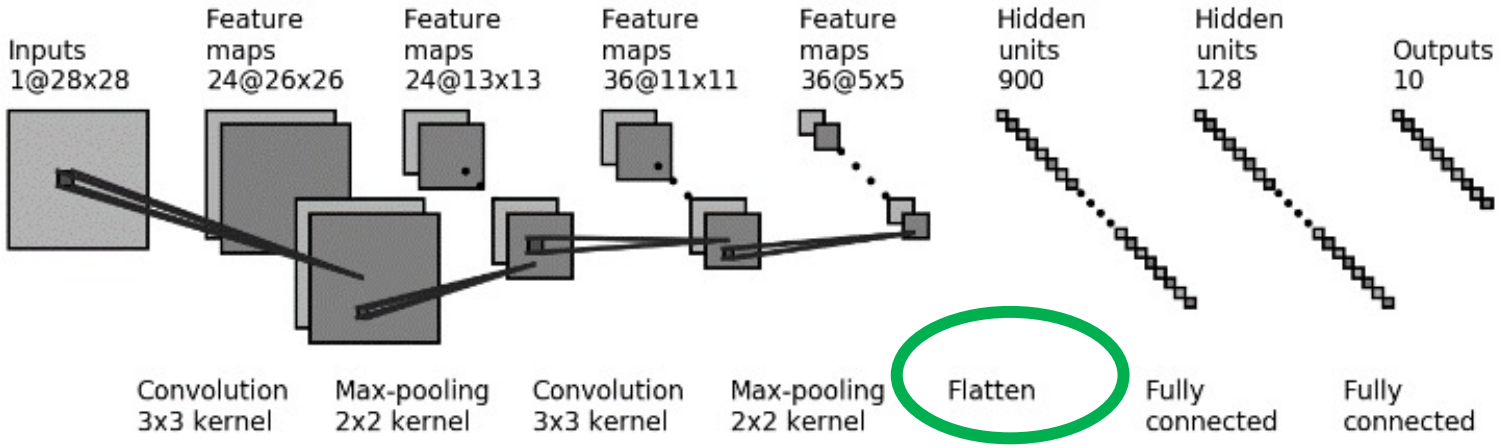


Max-pooling

- The pooling layer (POOL) is a **downsampling operation**, typically applied after a convolution layer, which does some spatial invariance.
- In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively

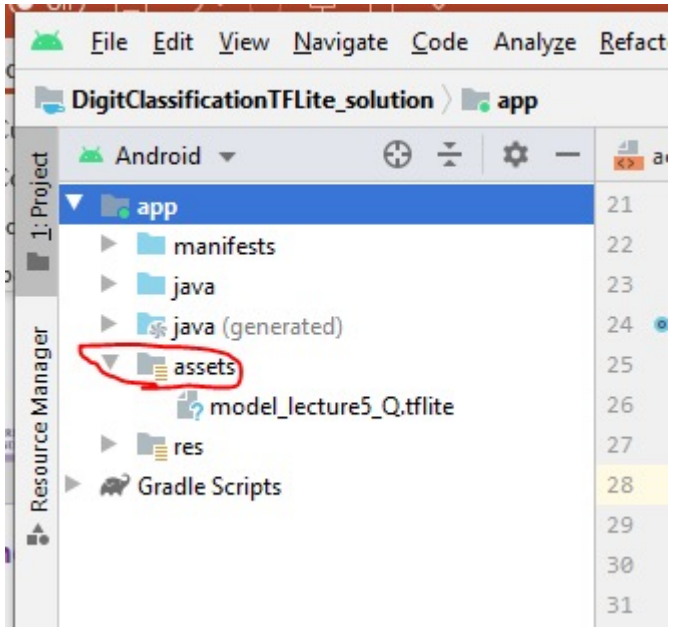


Convolutional Neural Network for digits recognition



- The **Flatten layer** is used to **transform a multidimensional tensor into one with a single dimension**.
- In this case, our (36, 5, 5) tensor is squished down into a single dimension with shape (900).

Include the model into our Android App



Include in the build.gradle(App)

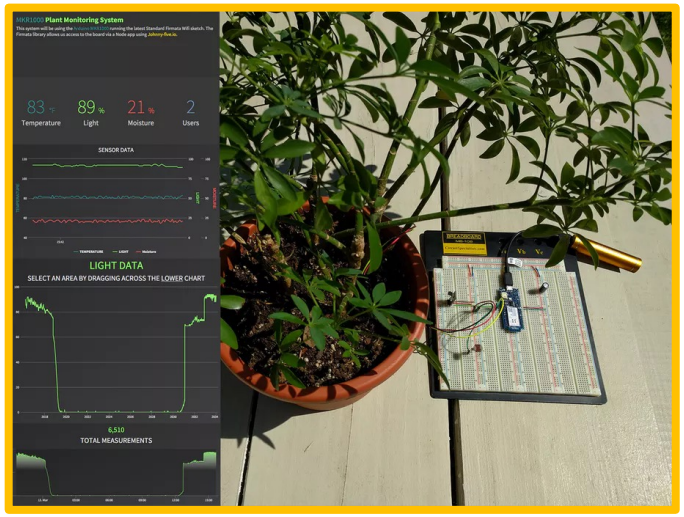
```
android{

aaptOptions {
    noCompress "tflite"
}

}
```

```
dependencies {

    // Task API - required only if you are running inference in a background task
    implementation "com.google.android.gms:play-services-tasks:17.0.0"

    // TF Lite
    implementation 'org.tensorflow:tensorflow-lite:0.0.0-nightly'
}
```

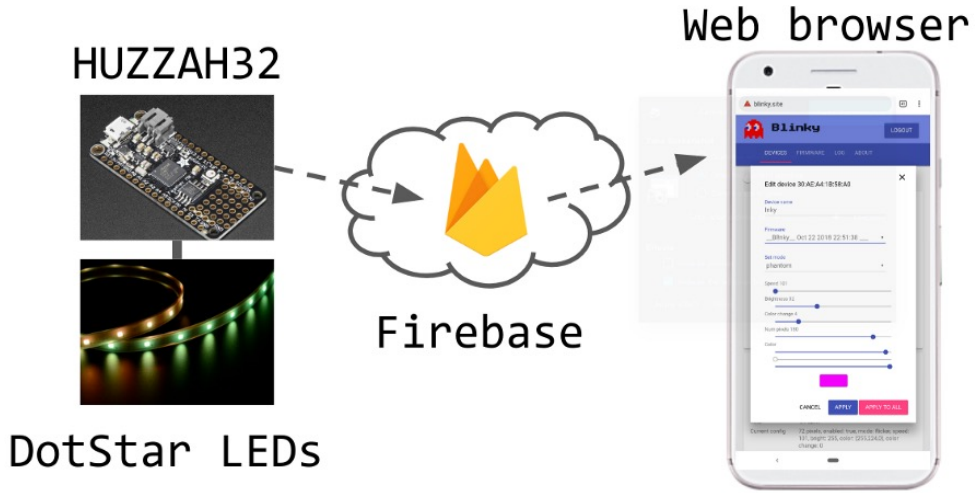
Android-Arduino Control

EE P 523, Lecture 6

Why Smartphones Aren’t enough?



Why Smartphones Aren’t enough?



https://medium.com/firebase-developers/using-firebase-to-control-your-arduino-project-over-the-web-ba94569d172c

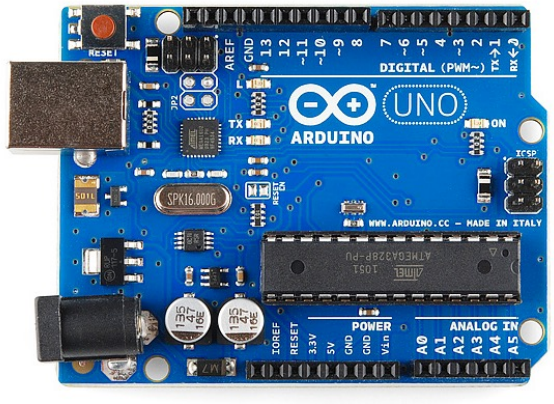
What Can We do with an Arduino?

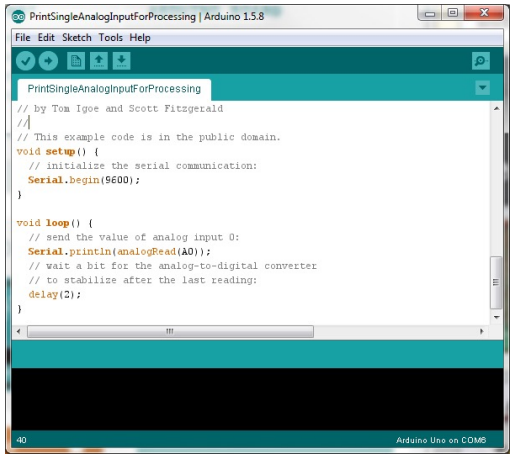
- Read inputs
  - light on a sensor
  - a finger on a button
  - a Twitter message
- Turn it into an output
  - activate a motor
  - turn on an LED
  - publishing something online

What is an Arduino?

- An open-source electronics platform based on easy-to-use hardware and software.

- Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming.



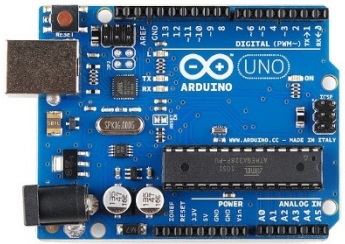


What is a Microcontroller?

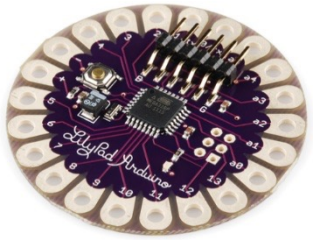
- A **microcontroller** (*µ-controller*) is a small computer on a single integrated circuit
- A microcontroller contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals

- Originally programmed only in assembly language
- Various high-level programming languages, such as C, Python and JavaScript, are now also in common use to target microcontrollers and embedded systems.

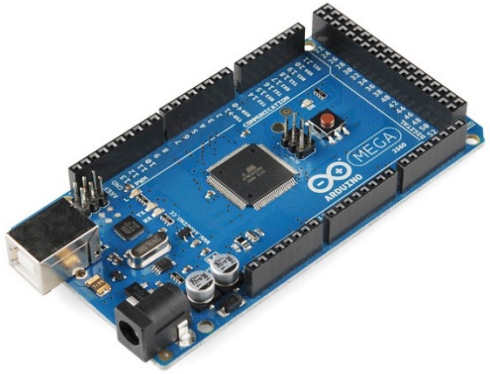
Some Popular Models



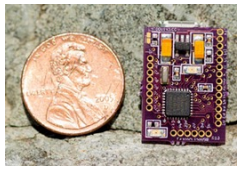
Arduino Uno (R3)
14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a USB connection, a power jack, a reset button



LilyPad Arduino
Very similar to the Arduino Uno but built for wearable e-textiles. The LilyPad also has its own family of input, output, power, and sensors built specifically for e-textiles.



Arduino Mega (R3)
The Arduino Mega is like the UNO's big brother. It has lots (*54!*) of digital input/output pins (14 can be used as PWM outputs), 16 analog inputs, a USB connection, a power jack, and a reset button



FemtoduinoUSB
The smallest Arduino board with micro-USB, a voltage regulator, same power and pin count as an Arduino UNO

How do We Program an Arduino?

- Arduino language is basically a set **of C/C++ functions** that can be called from your code.
- Your sketch undergoes minor changes (e.g., automatic generation of function prototypes) and then is passed directly to a C/C++ compiler (avr-g++).

How do We Program an Arduino?

The Arduino IDE supports the languages **C and C++** using special rules of code structuring.

A minimal Arduino C/C++ program consist of only two functions:

setup():

- Called once when a sketch starts after power-up or reset.
- Used to initialize variables, input and output pin modes, and other libraries needed in the sketch.

loop()

- After setup() function ends, the loop() function is executed **repeatedly** in the main program.
- Controls the board until the board is powered off or is reset.

Arduino Programming Language

Functions

Digital I/O
- digitalRead()
- digitalWrite()
- pinMode()

Analog I/O
- analogRead()
- analogReference()
- analogWrite()

Zero, Due & MKR Family
- analogReadResolution()
- analogWriteResolution()

Advanced I/O
- noTone()
- pulseIn()
- pulseInLong()
- shiftIn()
- shiftOut()
- tone()

Time
- delay()
- delayMicroseconds()
- micros()
- millis()

Math
- abs()
- constrain()
- map()
- max()
- min()
- pow()
- sq()
- sqrt()

Trigonometry
- cos()
- sin()
- tan()

Characters
- isAlpha()
- isAlphaNumeric()
- isAscii()
- isControl()
- isDigit()
- isGraph()
- isHexadecimalDigit()
- isLowerCase()
- isPrintable()
- isPunct()
- isSpace()
- isUpperCase()

Random Numbers
- random()
- randomSeed()

Bits and Bytes
- bit()
- bitClear()
- bitRead()
- bitSet()
- bitWrite()
- highByte()
- lowByte()

External Interrupts
- attachInterrupt()
- detachInterrupt()

Interrupts
- interrupts()
- noInterrupts()

Communication
- Serial
- Stream

USB
- Keyboard
- Mouse

Arduino Programming Language

Variables

Constants

Floating Point Constants

Integer Constants

HIGH | LOW

INPUT | OUTPUT | INPUT_PULLUP

LED_BUILTIN

true | false

Conversion

byte()

char()

float()

int()

long()

word()

Data Types

array

bool

boolean

byte

char

double

float

int

long

short

size_t

string

unsigned char

unsigned int

unsigned long

void

word

Variable Scope & Qualifiers

const

scope

static

volatile

Utilities

PROGMEM

sizeof()

Arduino Programming Language

Structure

Sketch

loop()
setup()

Control Structure

break
continue
do...while
else
for
goto
if
return
switch...case
while

Further Syntax

#define (define)
#include (include)
/* */ (block comment)
// (single line comment)
; (semicolon)
{} (curly braces)

Arithmetic Operators

% (remainder)
* (multiplication)
+ (addition)
- (subtraction)
/ (division)
= (assignment operator)

Comparison Operators

!= (not equal to)
< (less than)
<= (less than or equal to)
== (equal to)
> (greater than)
>= (greater than or equal to)

Boolean Operators

! (logical not)
&& (logical and)
|| (logical or)

Pointer Access Operators

& (reference operator)
* (dereference operator)

Bitwise Operators

& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

Compound Operators

%= (compound remainder)
&= (compound bitwise and)
*= (compound multiplication)
++ (increment)
+= (compound addition)
-- (decrement)
-= (compound subtraction)
/= (compound division)
^= (compound bitwise xor)
|= (compound bitwise or)

Blinking LED

a) Using delays

b) Using millis()

c) Using an interrupt

Blinking LED (a)

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  //turn the LED on (HIGH is the voltage level)
  delay(1000);                      //wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

Blinking LED without delay?

```
const long interval = 1000;              // interval at which to blink (milliseconds)

void setup(){
  pinMode(ledPin, OUTPUT); // set the digital pin as output:
}

void loop() {
   unsigned long currentMillis = millis();
}
```

Blinking LED without delay (b)

```
const int ledPin =  LED_BUILTIN;          // the number of the LED pin
int ledState = LOW;                       // ledState used to set the LED
unsigned long previousMillis = 0;         // will store last time LED was updated
const long interval = 1000;               // interval at which to blink (milliseconds)

void setup(){
  pinMode(ledPin, OUTPUT); // set the digital pin as output:
}

void loop() {
   unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {

     previousMillis = currentMillis; // save the last time you blinked the LED

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
        digitalWrite(ledPin, ledState); // set the LED with the ledState of the variable:

  }
}
```

Interrupts: When/When not

- Used for important, occasional events that should trigger as soon as possible (e.g., a button press)
- ISR = Interrupt Service Routine

- I interrupted my television viewing to take a bath.
- I was reading a book when visitors arrived and interrupted me. We then went to the movies.
- The user pressed a red button to interrupt the robot walking around.

- To detect pin changes (e.g., rotary encoders, button presses)
- Watchdog timer (e.g., if nothing happens after 8 seconds, interrupt me)
- Timer interrupts - used for comparing/overflowing timers
- SPI data transfers
- I2C data transfers
- USART data transfers
- ADC conversions (analog to digital)
- EEPROM ready for use
- Flash memory ready

Interrupts

Syntax

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);  //(recommended)
```

- digitalPinToInterrupt(pin): translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use digitalPinToInterrupt(3)
- ISR: to call when the interrupt occurs; this function takes no parameters and returns nothing. This function is sometimes referred to as an interrupt service routine.
- mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

  LOW to trigger the interrupt whenever the pin is low,
  CHANGE to trigger the interrupt whenever the pin changes value
  RISING to trigger when the pin goes from low to high,
  FALLING for when the pin goes from high to low.
  HIGH to trigger the interrupt whenever the pin is high. (some boards only)

Note about ISR function

- Generally, an ISR should be as short and as fast as possible.
- If Multiple ISRs: only one can run at a time, other interrupts will be executed after the current one finishes in an order that depends on the priority they have.
- Typically, global variables are used to pass data between an ISR and the main program.
- To make sure variables shared between an ISR and the main program are updated correctly, declare them as ***volatile***.

Interrupts: Example (c)

```
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
volatile int buttonState = 0;         // variable for reading the
 pushbutton status
```

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

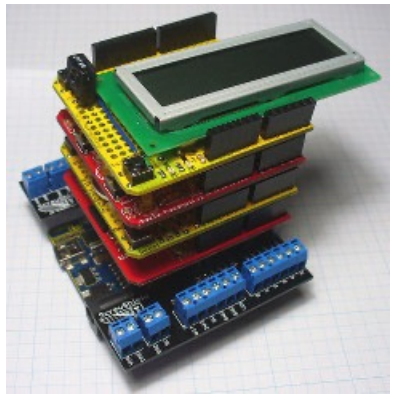
```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  // Attach an interrupt to the ISR vector
  attachInterrupt(0, pin_ISR, CHANGE);
}

void loop() {
  // Nothing here!
}

void pin_ISR() {
  buttonState = digitalRead(buttonPin);
  digitalWrite(ledPin, buttonState);
}
```

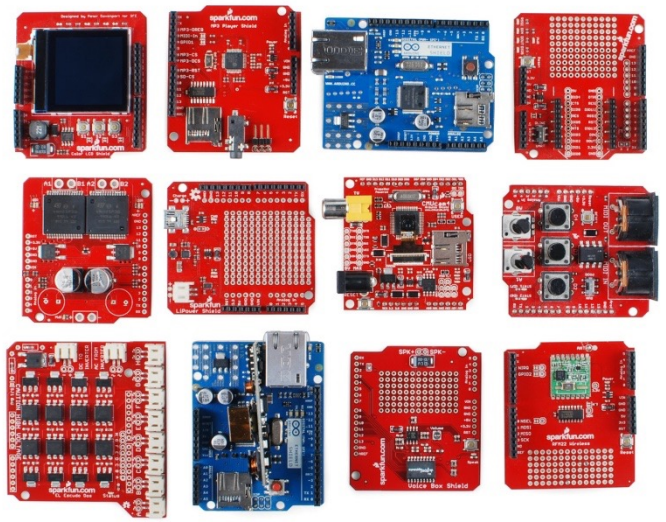
ISR: take aways

- Keep it *short*
- Don't use delay ()
- Don't do serial prints
- Make variables shared with the main code **volatile**
- Variables shared with main code may need to be protected by "critical sections" (see below)
- Don't try to turn interrupts off or on

Arduino Shields

- Many boards will come with stackable attachments (called ***shields***) that proved extra functionality
  - Ex: WiFi, controlling motors, cellular modems, controlling LCD screens, etc.





Getting Input from Sensors

Sensors use the following methods to provide information:

Digital on/off

- Some devices, such as the tilt sensor and the motion sensor in, simply switch a voltage on and off

Analog

- Other sensors provide an analog signal (a voltage that is proportional to what is being sensed, such as temperature or light level). All of them use the analogRead command

Pulse width

- Distance sensors, such as the PING, provide data using pulse duration proportional to the distance value.
- Applications using these sensors measure the duration of a pulse using the pulseIn command.

Getting Input from Sensors

Sensors use the following methods to provide information:

Serial

- Some sensors provide values using a serial protocol. For example, the GPS communicates through the Arduino serial port
- Most Arduino boards only have one hardware serial port

Synchronous protocols: I2C and SPI

- The I2C and SPI digital standards were created for microcontrollers like Arduino to talk to external sensors and modules
- These protocols are used extensively for sensors, actuators, and peripherals

How do We Get Analog Values?

- Computers can only have two voltages: on and off.

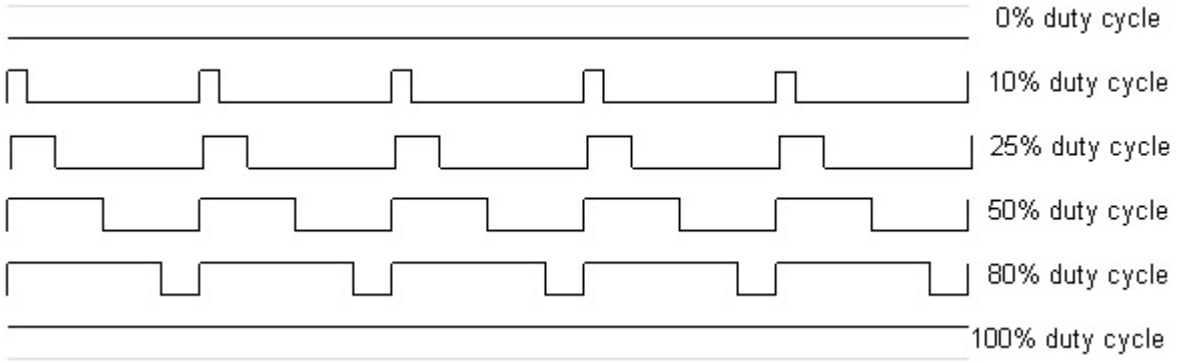
- Pulse Width Modulation, or PWM, is a technique for **getting analog results with digital means**.

Pulse-Width Mouldation (PWM)

PWM has several uses:

- Dimming an LED
- Providing an analog output; if the digital output is filtered,
  it will provide an analog voltage between 0% and 100% .
- Generating audio signals.
- Providing variable speed control for motors.
- Generating a modulated signal, for example to drive an infrared LED for a remote control.

Pulse-Width Mouldation (PWM)



PWM signal is a digital square wave, where the frequency is constant, but that fraction of the time the signal is on (the duty cycle) can be varied between 0 and 100%

```
analogWrite(pin, dutyCycle)
```

PWM with Arduino

1. **AnalogWrite:**

```
analogWrite(pin, dutyCycle)
```

   + dutyCycle from 0 to 255
   + simple interface to hardware PWM
   - No control over frequency

2. **Time delays**:
   repeatedly turning the pin on and off for the desired times (example)
   + Use any digital output pin, full control of duty cycle and frequency
   - Any interrupts will affect the timing
   - Can't leave the output running while the processer does something else

3. **Microcontroller timers**:
   by manipulating the chip's timer registers directly, you can obtain more control than the analogWrite function provides.
   (example)

Fading with PWM

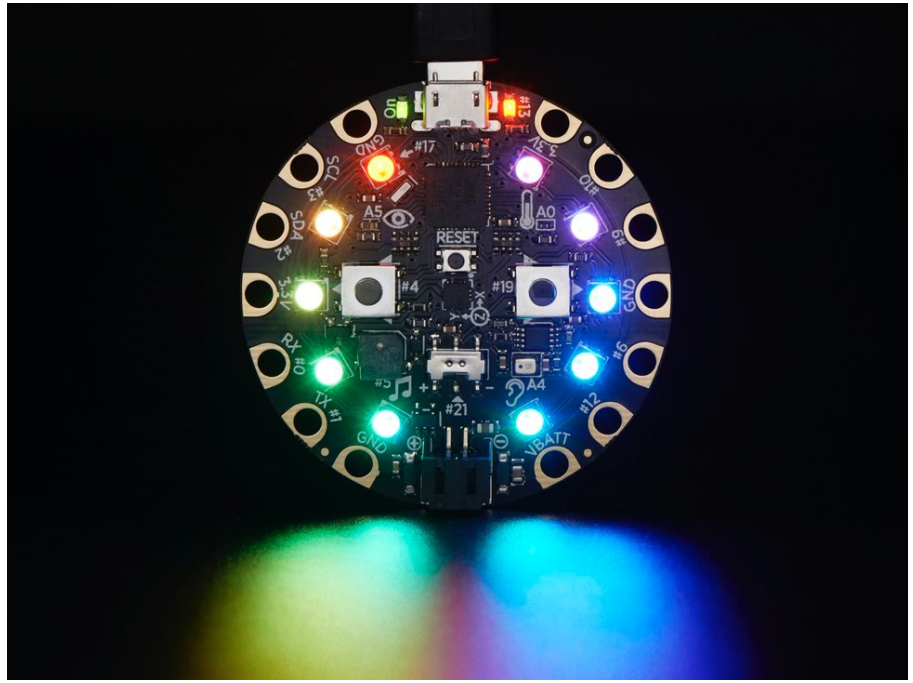
```
int ledPin = 13;  // Build in LED in CircuitPlayground

void setup() {
}

void loop() {
  // fade in from min to max in increments of 5 points:
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5)
{
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }

  // fade out from max to min in increments of 5 points:
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5)
{
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}
```

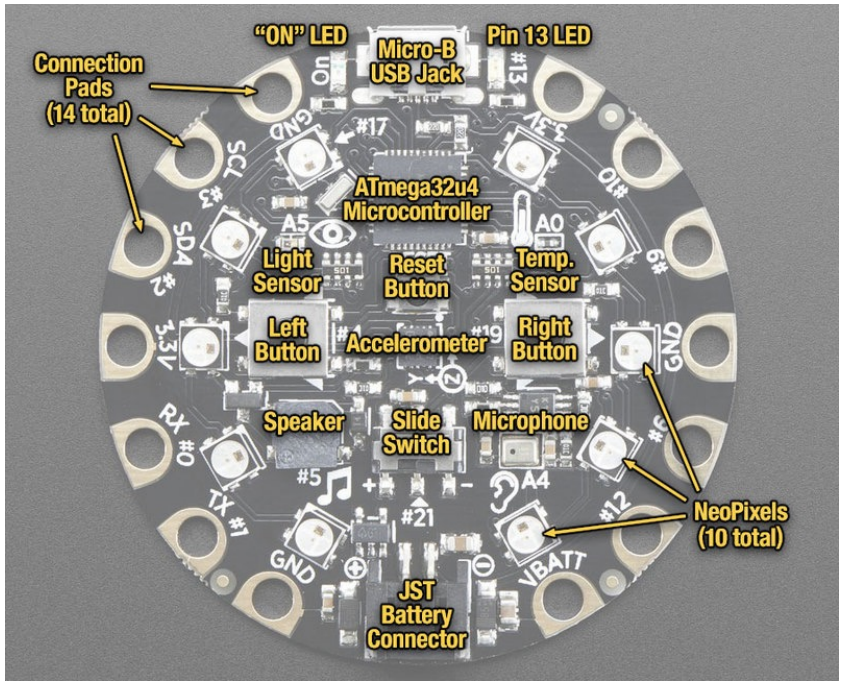
To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

Circuit Playground

EE P 523, Lecture 6

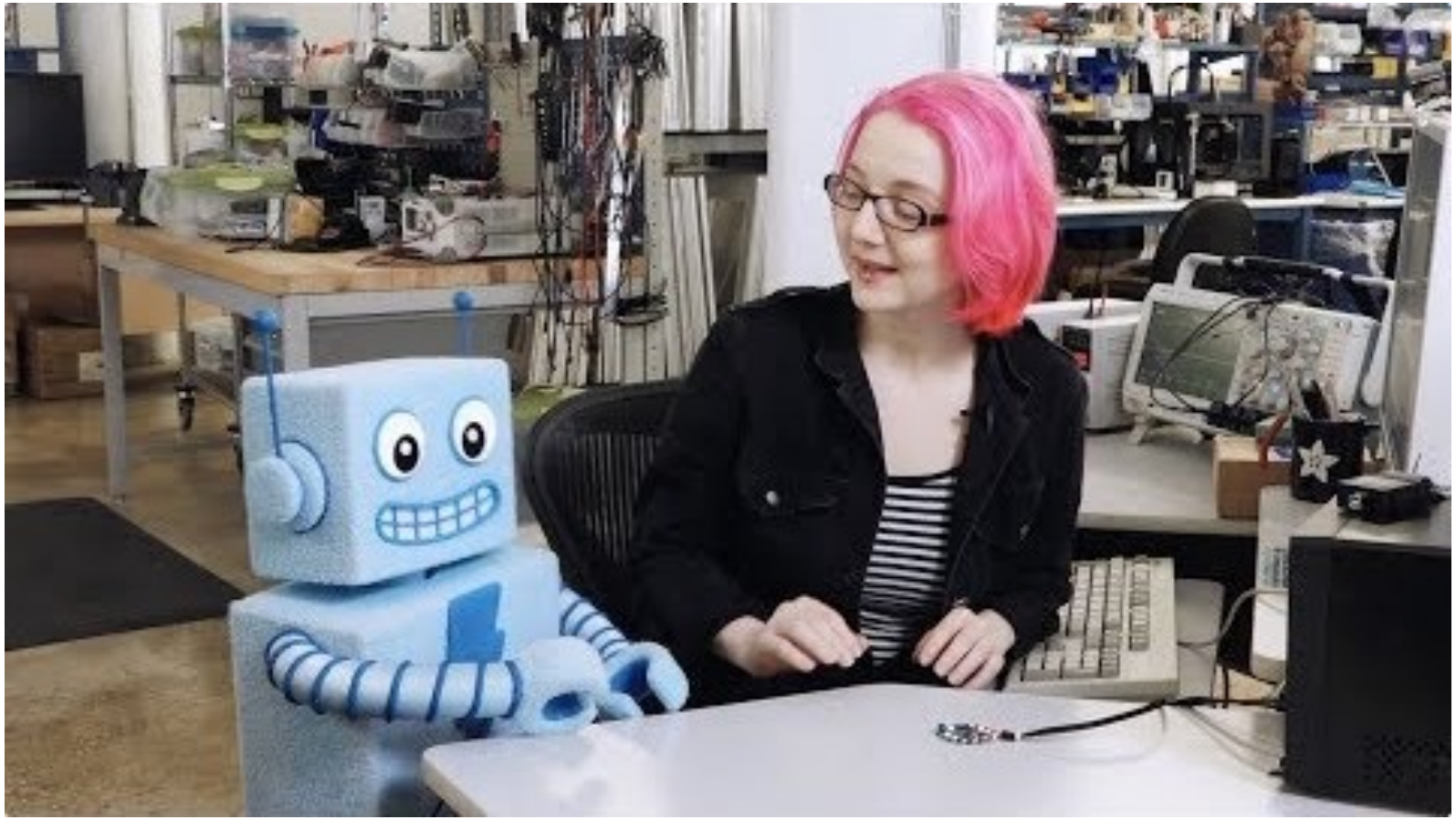
Circuit Playground





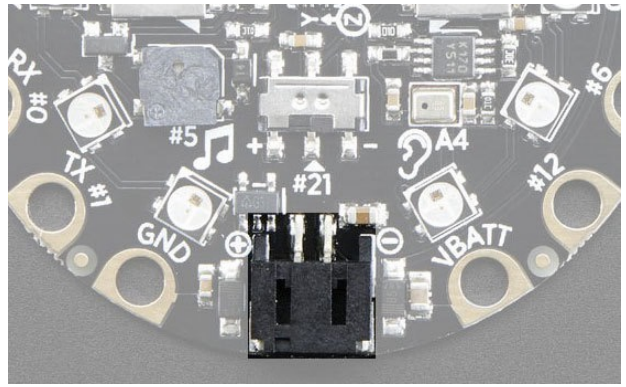
ATmega32u4 Processor, running at 3.3V and 8MHz

Circuit Playground

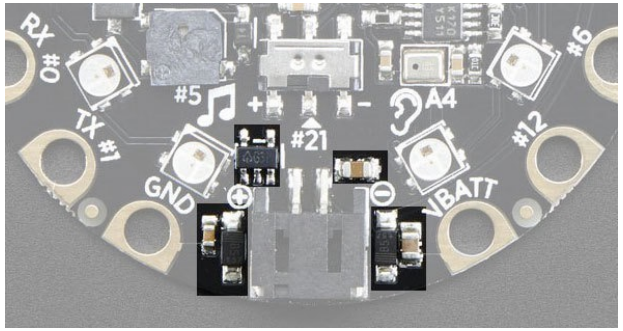


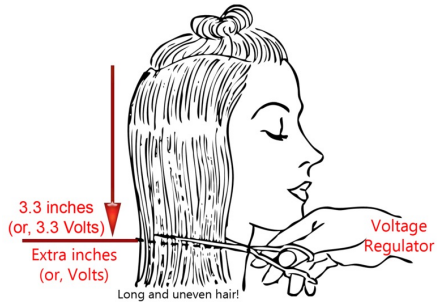
Power option 1: battery power jack

1. Battery pack must be 3.5-6.5 V DC
   - 3 × 1.5V AAA batteries or lithium-ion/LiPo battery

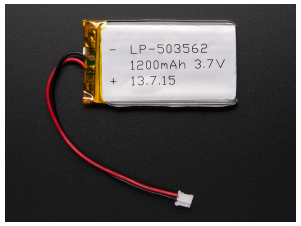


- Provides polarity protection diode
- Voltage regulator to provide consistent 3.3 V

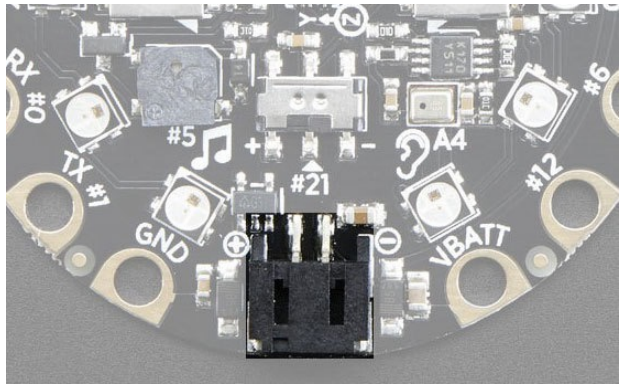


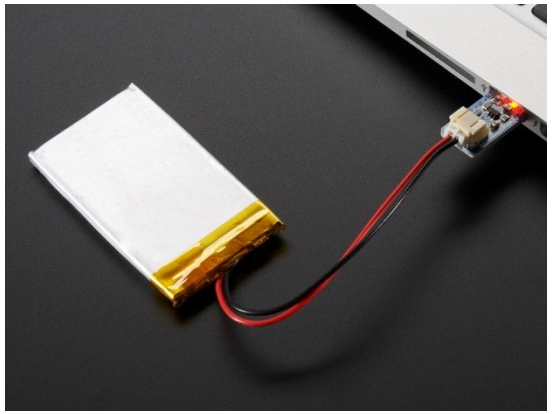


Power option 1: battery power jack



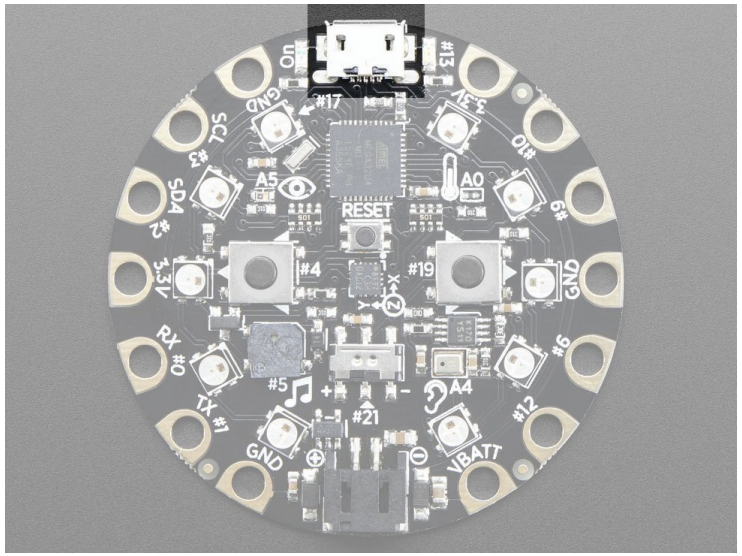
on or lithium polymer battery





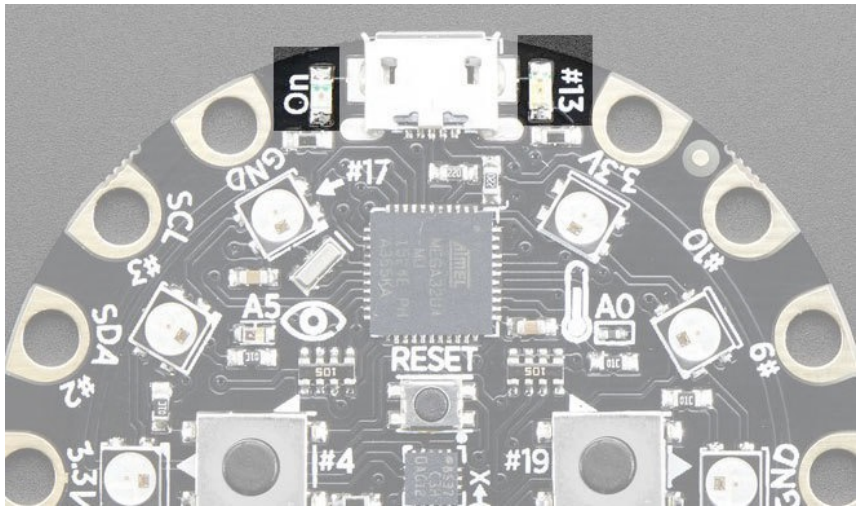
Power option 2: USB

- Can also be powered via USB
- Important: to program, you must use a USB cable that transfers data, not one that just charges
- Most Arduino boards require a USB to Serial Interface Translator chip to work with USB, but that comes with the ATmega32u4 processor



Built-in red LED

- Left LED turns green if the board is on, flickers if something is wrong
- `CircuitPlayground.redLED(true)`
  - Turns the right LED on

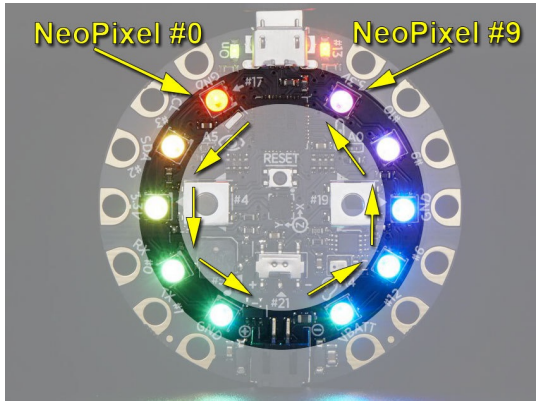


NeoPixels LEDs

- `CircuitPlayground.setPixelColor(`*`n, red, green, blue`*`)`
  - Changes the color of the n^{th} LED to RGB color
  - 0 (off) ≤ red, green, blue ≤ 255 (on)

- `CircuitPlayground.clearPixels()`
  - Turns off all the LEDs

- `CircuitPlayground.setBrightness(b)`
  - Sets LED brightness
  - 0 (no light) ≤ brightness ≤ 255 (super bright)
  - Brightness controlled using pulse-width modulation



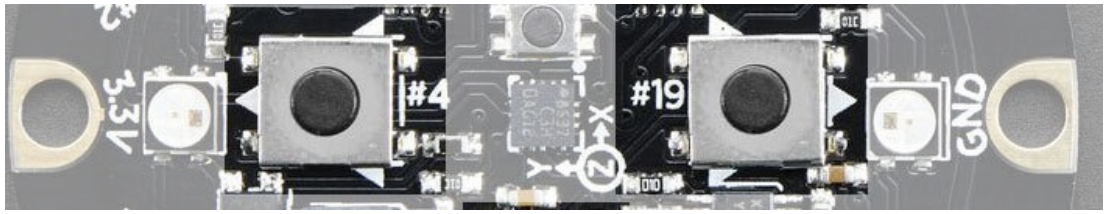
Using the Neo Pixels

```
#include <Adafruit_CircuitPlayground.h>

void setup() {
  CircuitPlayground.begin();
}

void loop() {
  CircuitPlayground.clearPixels();

  delay(500);

  CircuitPlayground.setPixelColor(0, 255,   0,   0);
  CircuitPlayground.setPixelColor(1, 128, 128,   0);
  CircuitPlayground.setPixelColor(2,   0, 255,   0);
  CircuitPlayground.setPixelColor(3,   0, 128, 128);
  CircuitPlayground.setPixelColor(4,   0,   0, 255);

  CircuitPlayground.setPixelColor(5, 0xFF0000);
  CircuitPlayground.setPixelColor(6, 0x808000);
  CircuitPlayground.setPixelColor(7, 0x00FF00);
  CircuitPlayground.setPixelColor(8, 0x008080);
  CircuitPlayground.setPixelColor(9, 0x0000FF);

  delay(5000);
}
```

Push Buttons

- `CircuitPlayground.leftButton()`, `CircuitPlayground.rightButton()`
  - Returns true if button is pressed

- Connected to 10KΩ *pull-down* resistors.
  - Value is “high” when button is pressed



Signal Debouncing

- Pushbuttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program
- Check twice in a short period of time to make sure the pushbutton is definitely pressed.
- Without debouncing, pressing the button once may cause unpredictable results. This sketch uses the millis() function to keep track of the time passed since the button was pressed.

Signal Debouncing: Formal Definition

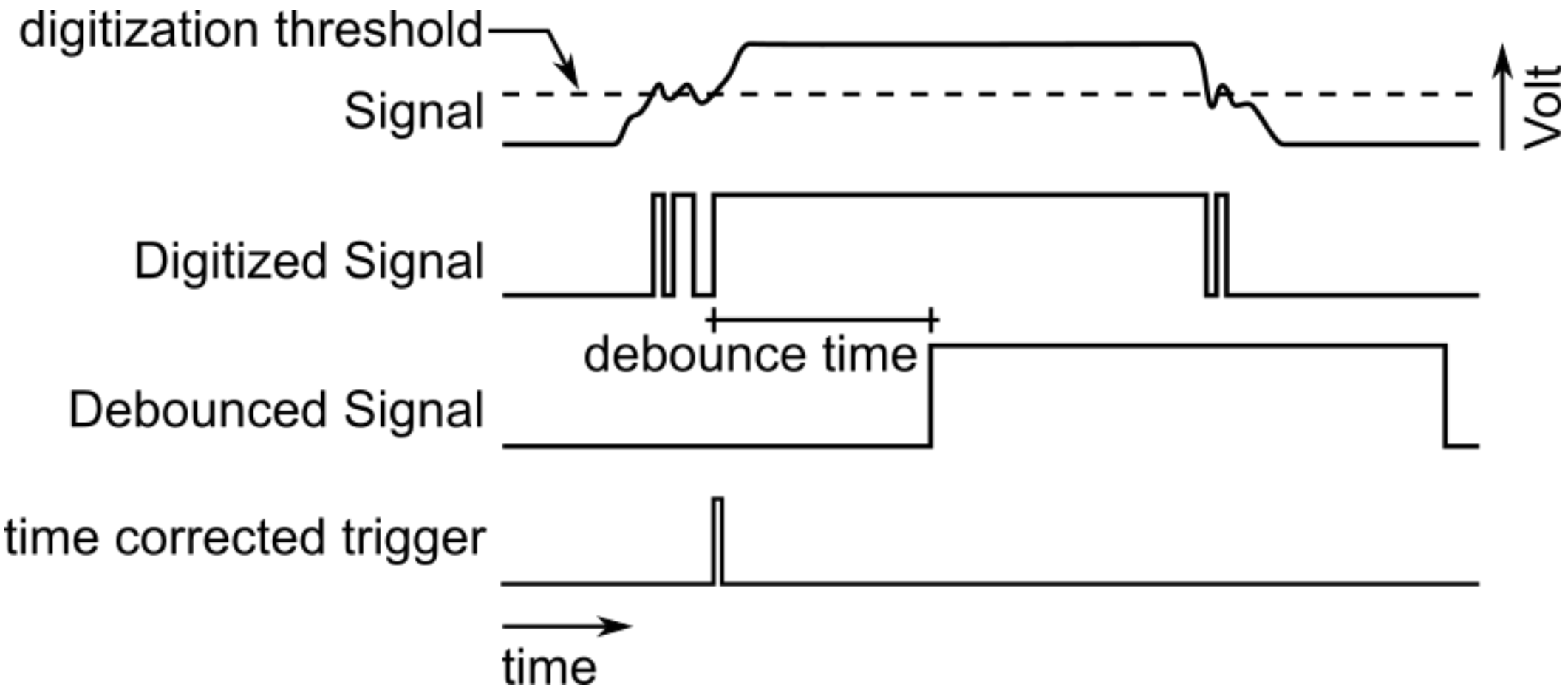
- **Bouncing**

tendency of any two metal contacts in an electronic device to generate multiple signals as the contacts close or open;

- **Debouncing**

is any kind of hardware device or software that ensures that only a single signal will be acted upon for a single opening or closing of a contact.

Signal Debouncing

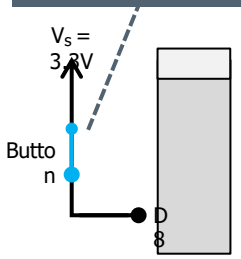


Signal Debouncing

```
  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer than the
debounce
    // delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;

      // only toggle the LED if the new button state is HIGH
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }
```

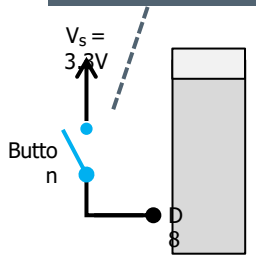
Hooking up a Button: Pull-down Resistors

When this button is closed (as it is below), what would the **microcontroller** (MCU) **read** from the **input pin**?



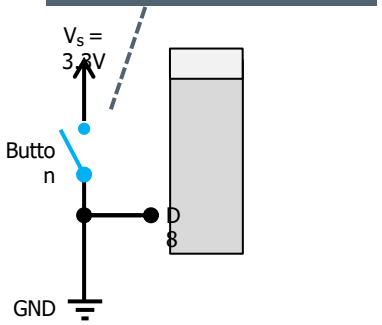
Answer: The MCU would read 3.3V (or HIGH)

Now, when the **button switches** to **open**, what would the MCU read?



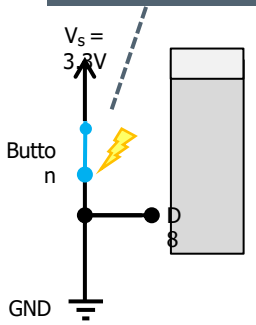
Answer: Uh, oh, the input pin is in an unknown state (commonly called "floating")—this is bad!

Well, can't we just solve that by **adding GND** here that "pulls" D8 to 0V when the switch is open?



Answer: You're on the right track but this will create a short circuit when the button is closed!

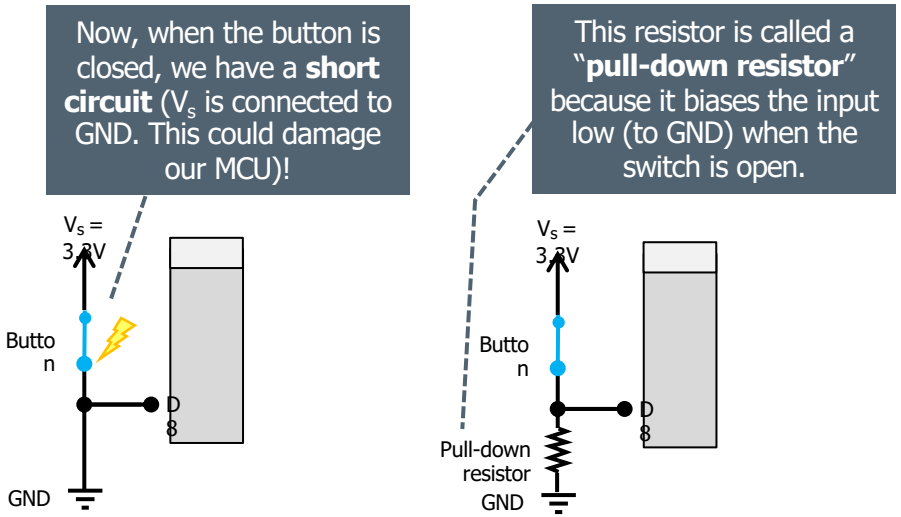
Now, when the button is closed, we have a **short circuit** (V_s is connected to GND. This could damage our MCU)!



Question: So, what do we do? We add what's called a pull-down resistor.

See: https://learn.sparkfun.com/tutorials/switch-basics/all; https://learn.sparkfun.com/tutorials/pull-up-resistors/all

Hooking up a button: Pull-down Resistors



Question: So, what do we do? We add what's called a pull-down resistor.

Now, when the switch is open, the MCU is pulled to GND. When the switch is closed, the MCU is 3.3V (HIGH).

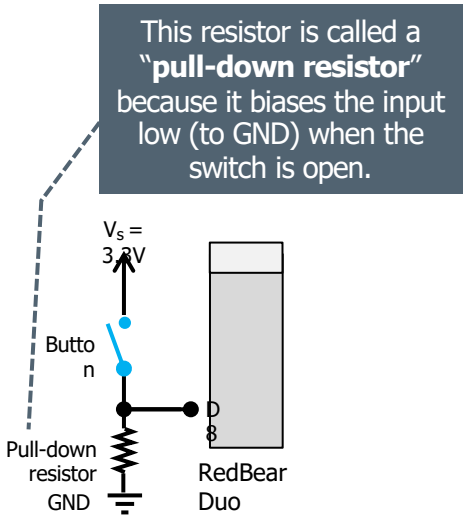
How do you choose the resistor value for the pull-down?

If you choose a low resistor value, more current will be "wasted" by flowing from V_s to GND when the button is closed. In contrast, a high resistor value (*e.g.*, 4MΩ) may not work as a pull-down (not enough current will flow).

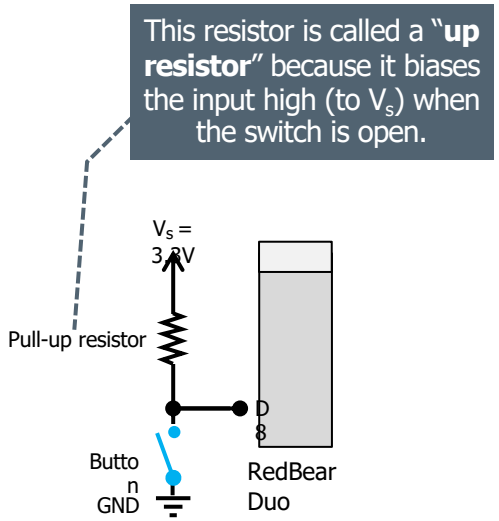
Arduino recommends using a 10K Ω resistor.

See: https://learn.sparkfun.com/tutorials/switch-basics/all; https://learn.sparkfun.com/tutorials/pull-up-resistors/all

Pull-up vs. pPull-down Resistors



Pull-Down Resistor Configuration
When the switch is open, the MCU is pulled to GND. When the switch is closed, the MCU is 3.3V (HIGH) as it becomes directly connected to V_s.



Pull-Up Resistor Configuration
When the switch is open, the MCU is pulled to V_s. When the switch is closed, the MCU is 0V (LOW) as it becomes directly connected to GND.

See: https://learn.sparkfun.com/tutorials/switch-basics/all; https://learn.sparkfun.com/tutorials/pull-up-resistors/all

Both pull-down resistor configurations and pull-up resistor configurations achieve the same goal.

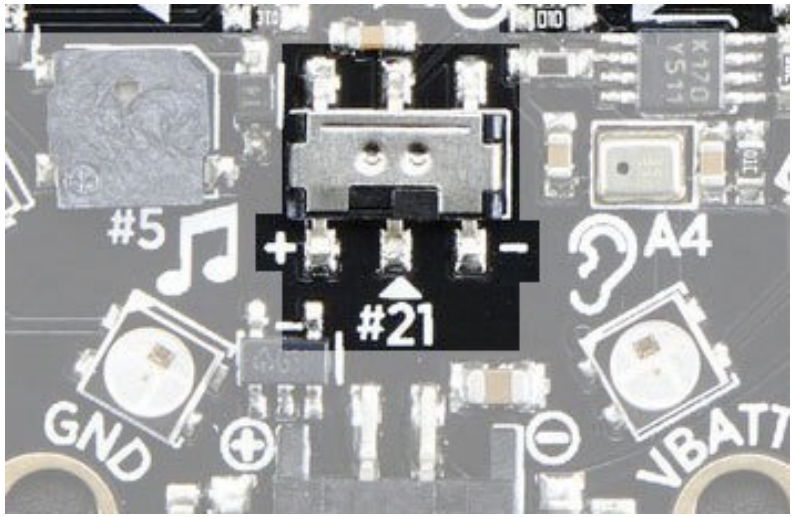
Pull-down resistors tend to make more sense because the switch is 0V (LOW) when open and 3.3 (HIGH) when closed.

Using the Push Buttons

```
#include <Adafruit_CircuitPlayground.h>

bool leftButtonPressed;
bool rightButtonPressed;

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

void loop() {
  leftButtonPressed = CircuitPlayground.leftButton();
  rightButtonPressed = CircuitPlayground.rightButton();

if (leftButtonPressed) { }
 else {}

if (rightButtonPressed) {}
 else {}

delay(1000);
}
```

Slide Switch

- `CircuitPlayground.slideSwitch()`
  - Returns true if switch is set to the left

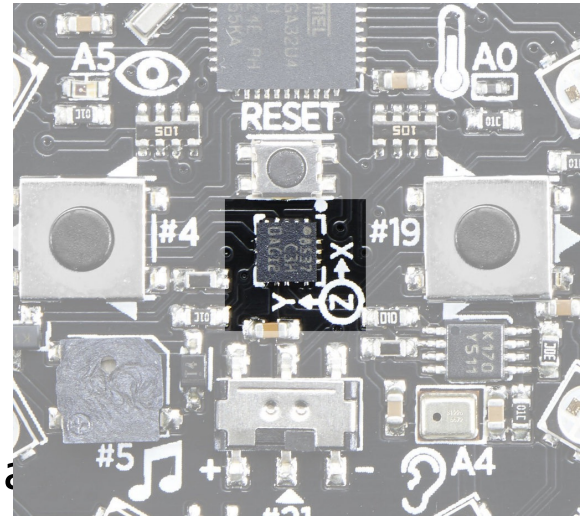


Using the Slide Switch

```
#include <Adafruit_CircuitPlayground.h>

bool slideSwitch;

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

void loop() {
  slideSwitch =
CircuitPlayground.slideSwitch();

  Serial.print("Slide Switch: ");
  if (slideSwitch) {
    Serial.print("+");
  } else {
    Serial.print("-");
  }
  Serial.println();

  delay(1000);
}
```

Accelerometer

- Connected to hardware SPI pins
- Sensitivity: ±2*g*, 4*g*, 8*g*
- Smaller range == more precision



- `CircuitPlayground.motionX()`, `CircuitPlayground.motionY()`, `CircuitPla`
  - Reads acceleration values
- `CircuitPlayground.setAccelRange(LIS3DH_RANGE_2_G)`
  - Changes sensitivity of the accelerometer

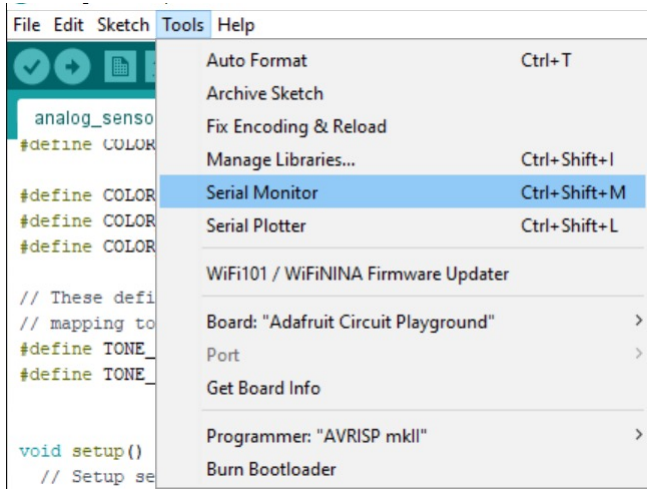
Using the Accelerometer

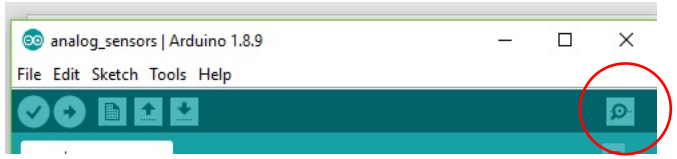
```
#include <Adafruit_CircuitPlayground.h>

float X, Y, Z;

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

void loop() {
  X = CircuitPlayground.motionX();
  Y = CircuitPlayground.motionY();
  Z = CircuitPlayground.motionZ();

  Serial.print("X: ");
  Serial.print(X);
  Serial.print("  Y: ");
  Serial.print(Y);
  Serial.print("  Z: ");
  Serial.println(Z);

  delay(1000);
}
```

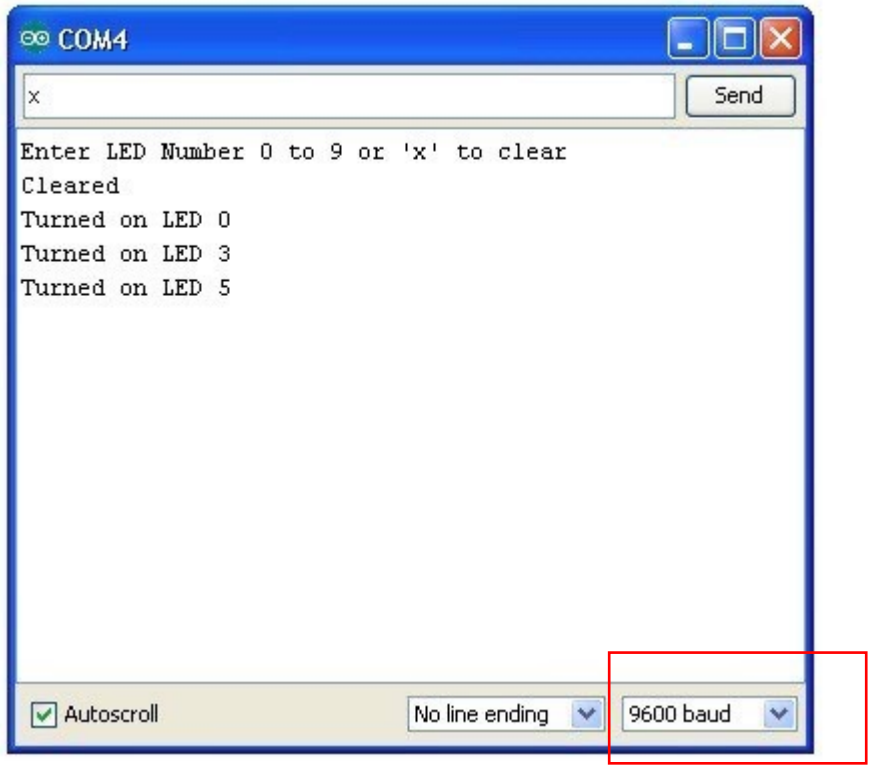
Debugging my System?

Serial Monitor:

Used to send messages from your computer to an Arduino board (over USB) and to receive messages from the Arduino.







Serial Monitor

```
void setup() {
  Serial.begin(9600);//baud rate
   …
}
```

Firstly, we have the command 'Serial.begin(9600)'.
This starts serial communication, so that the Arduino can send out commands through the USB connection. The value 9600 is called the 'baud rate' of the connection. This is how fast the data is to be sent. You can change this to a higher value, but you will also have to change the Arduio Serial monitor to the same value.

```
int analogValue = 0; // variable to hold the analog value void

setup() { // open the serial port at 9600 bps:
      Serial.begin(9600);
  }

void loop() { // read the analog input on pin 0:
      analogValue = analogRead(0); // print it out in many formats:
      Serial.println(analogValue); // print as an ASCII-encoded decimal
      Serial.println(analogValue, DEC); // print as an ASCII-encoded decimal
      Serial.println(analogValue, HEX); // print as an ASCII-encoded hexadecimal
      Serial.println(analogValue, OCT); // print as an ASCII-encoded octal
      Serial.println(analogValue, BIN); // print as an ASCII-encoded binary
       // delay 10 milliseconds before the next reading:
      delay(10);
}
```

Using the Light Sensor

```
#include <Adafruit_CircuitPlayground.h>

int value;

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

void loop() {
  value = CircuitPlayground.lightSensor();

  Serial.print("Light Sensor: ");
  Serial.println(value);

  delay(1000);
}
```

Using the Sound Sensor

```
#include <Adafruit_CircuitPlayground.h>

float value;

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

void loop() {
  // Take 10 milliseconds of sound data to calculate
  value = CircuitPlayground.mic.soundPressureLevel(10);

  Serial.print("Sound Sensor SPL: ");
  Serial.println(value);

  delay(90);
}
```

Using the Speaker

```
#include <Adafruit_CircuitPlayground.h>

void setup() {
  CircuitPlayground.begin();
}

void loop() {
  CircuitPlayground.playTone(500, 100);
  delay(1000);
}
```

Using the Temperature Sensor

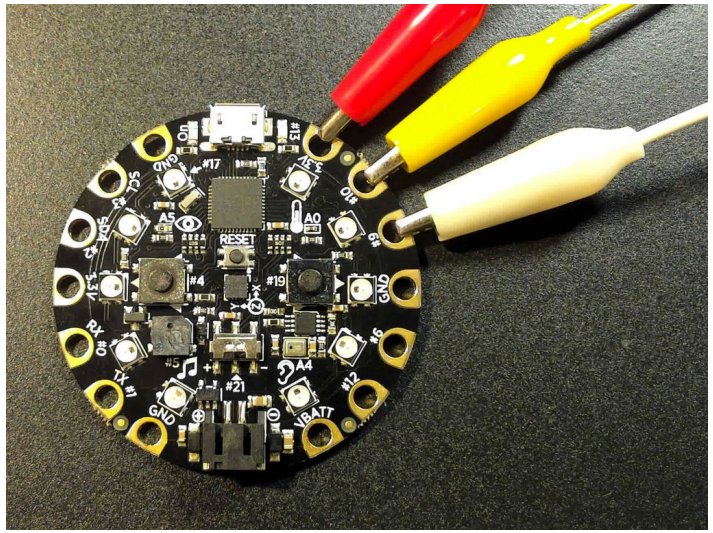
```
#include <Adafruit_CircuitPlayground.h>

float tempC, tempF;

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

void loop() {
  tempC = CircuitPlayground.temperature();
  tempF = CircuitPlayground.temperatureF();

  Serial.print("tempC: ");
  Serial.print(tempC);
  Serial.print("  tempF: ");
  Serial.println(tempF);

  delay(1000);
}
```

Using the Temperature Sensor

```
#include <Adafruit_CircuitPlayground.h>

float tempC, tempF;

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

void loop() {
  tempC = CircuitPlayground.temperature();
  tempF = CircuitPlayground.temperatureF();

  Serial.print("tempC: ");
  Serial.print(tempC);
  Serial.print("  tempF: ");
  Serial.println(tempF);

  delay(1000);
}
```

Aligator Pads & Pinout

Pin Name	Purpose
3.3V	Output 3.3V from onboard power supply
#10	Analog input (A10), PWM output
#9	Analog input (A9), PWM output
GND	Ground
#6	Analog input (A7), PWM output
#12	Analog input (A11), PWM output
VBATT	Output voltage from battery or USB
SCL #3	PWM output and interrupt input (INT0), used for I2C as clock
SDA #2	PWM output and interrupt input (INT1), used for I2C as data
RX #0	Interrupt input (INT2), UART receive
TX #1	Interrupt input (INT3), UART transmit



Pinouts

- **D10 / A10** - This pin can be digital I/O, or Analog Input. This pin has PWM output
- **D9 / A9** - This pin can be digital I/O, or Analog Input. This pin has PWM output.
- **D6 / A7** - This pin can be digital I/O, or Analog Input. This pin has PWM output.
- **D12 / A11** - This pin can be digital I/O, or Analog Input.
- **D1** - This pin can be digital I/O, it is also used for **Hardware Serial Transmit,** and can be an interrupt input.
- **D0** - This pin can be digital I/O, it is also used for **Hardware Serial Receive,** and can be an interrupt input.
- **D2** - This pin can be digital I/O, it is also the **I2C SDA** pin, and can be an interrupt input
- **D3** - This pin can be digital I/O or PWM output, it is also the **I2C SCL** pin, and can be an interrupt input

Internally Used Pins

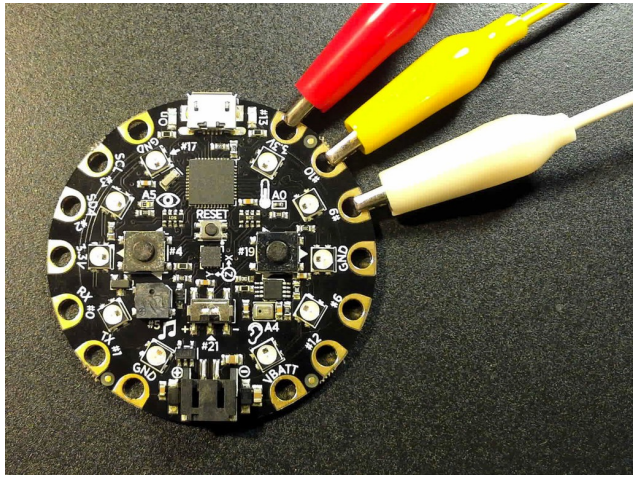
These are the names of the pins that are used for built in sensors and such!

- **D4** - Left Button A
- **D5** - Speaker PWM output
- **D7** - Accelerometer interrupt
- **D13** - Red LED
- **D17** - Built-in 10 NeoPixels
- **D19** - Right Button B
- **D21** - Slide Switch
- **A0** - Temperature Sensor
- **A4 -** Microphone sound sensor
- **A5** - Light Sensor

Aligator Pads & Pinout

- All non-power pads (i.e., not GND, 3.3V, VBATT) act as **capacitive touch pads**

- `CircuitPlayground.readCap(pin)`
  - Pins = 0, 1, 2, 3, 6, 9, 10, 12
  - Reads capacitance, usually 0-1000, reading ≥ 50 considered touch



Quick Tips

- `Serial.begin(9600)`
  - Should appear in the `setup()` part of the code to establish bitrate of serial communication

- `delay(value)`
  - Used in `loop()` to set delay between successive readings (i.e., sampling rate) and also ensure stability

- `CircuitPlayground.begin()`
  - Used in `setup()` to load up the library functions specific to the board

- `#define arr_len(x) (sizeof(x)/sizeof(*x))`
  - Handy macro for getting the length of an array

Android Pulse Sensor

EE P 523, Lecture 6

Pulse Sensor

- When blood is pumped through your body, it gets squeezed into the capillary tissues, and the volume of those tissues increases very slightly.

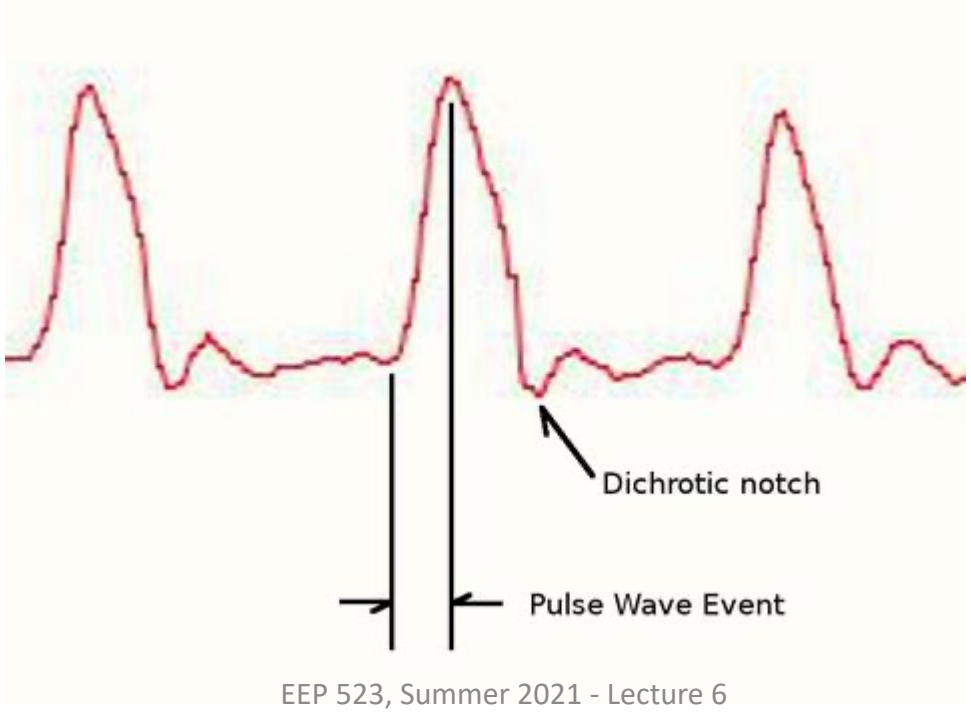
- Then, between heart beats, the volume decreases.

- **The change in volume effects the amount of light that will transmit through. This fluctuation is very small, but we can sense it with electronics**

Arduino Pulse Sensor

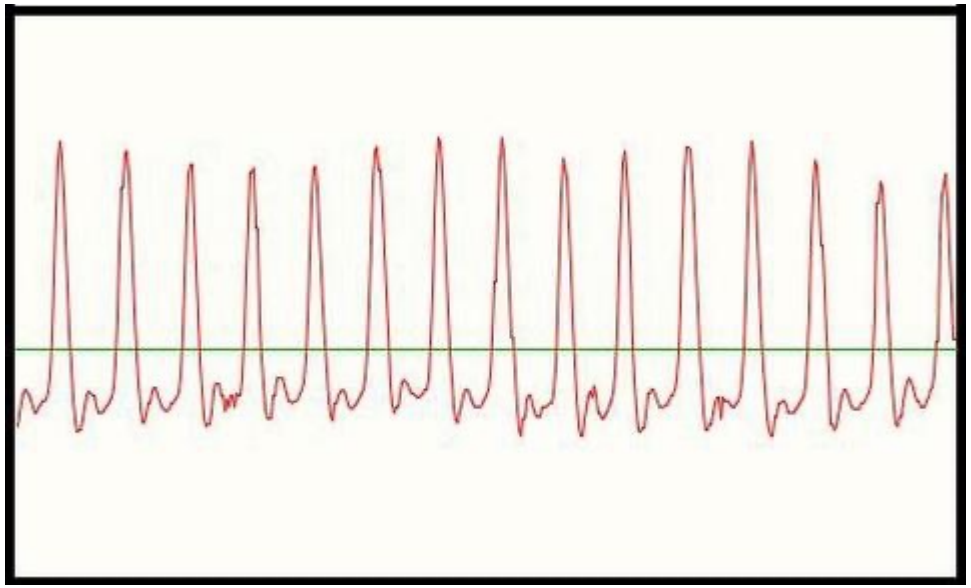
The Pulse Sensor that we make is essentially a photoplethysmograph (PPG), which is a well-known medical device used for **non-invasive heart rate monitoring**.

The heart pulse signal that comes out of a PPG is an **analog fluctuation in voltage**, and it has a predictable wave shape

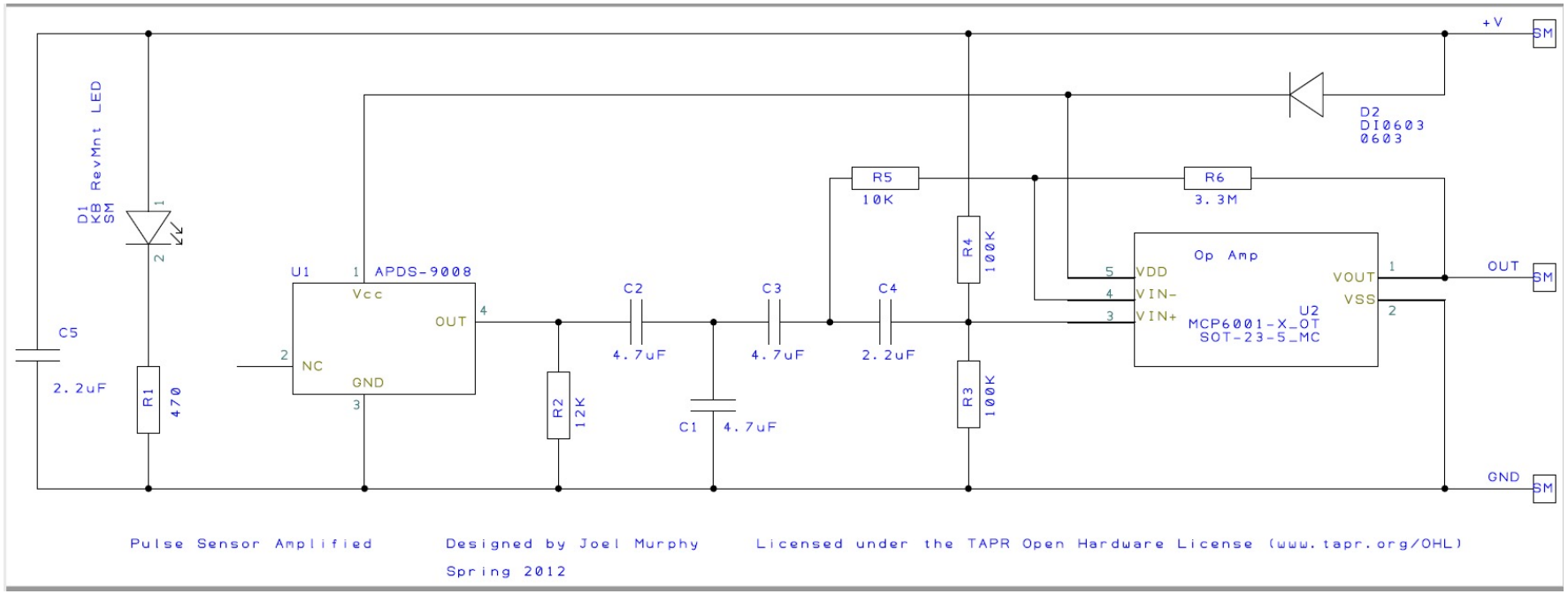


Arduino Pulse Sensor

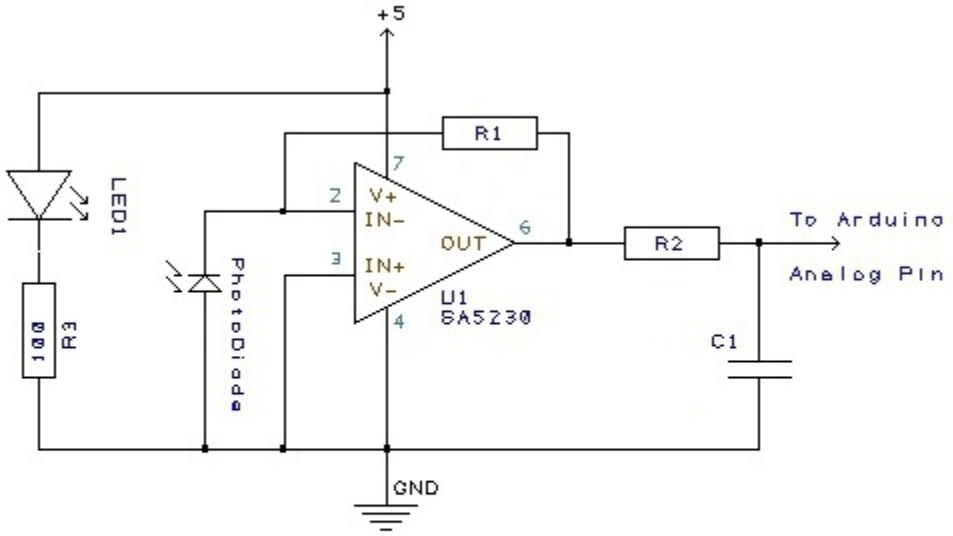
- When the Pulse Sensor Amped is not in contact with any fingers, the analog signal hovers around the mid-point of the voltage, or V/2.
- When Pulse Sensor Amped is in close contact with your fingertip **the change in reflected light when blood pumps through your tissues makes the signal fluctuate around that reference point.**
- Arduino watches the analog signal from Pulse Sensor, and decides a pulse is found when the signal rises above the mid-point. That's the moment when your capillary tissues gets slammed with a surge of fresh blood.
- Then, when the signal drops below the mid-point, Arduino sees this and gets ready to find the next pulse.



Arduino Pulse Sensor

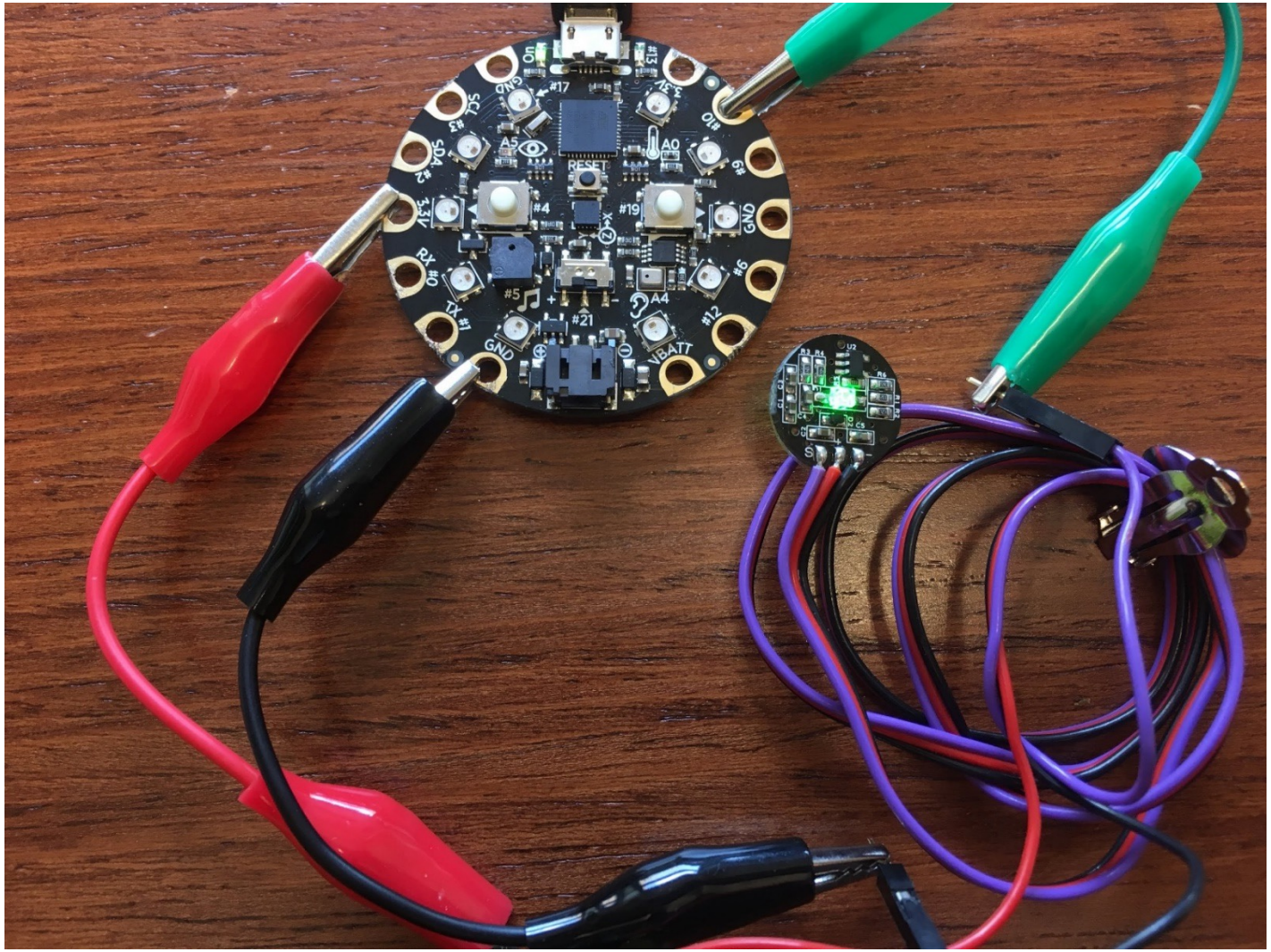


Arduino Pulse Sensor



Heartbeat Monitor Circuit
Feedback R1=1M
Low Pass Filter R2=100 C1=4.7uF

Connecting to Arduino



Connecting to Arduino

To use our Arduino board and the pulse sensor

```
#include <PulseSensorPlayground.h>
```

To use initialize the Arduino board and clear all the LEDs

```
void setup() {
            Serial.begin(9600);
     CircuitPlayground.begin();
     CircuitPlayground.clearPixels();
    …

    }
```

Visualizing the Signal

```
//  Variables
int PulseSensorPurplePin = 10;        // Pulse Sensor PURPLE WIRE connected to ANALOG PIN 10
int LED13 = 13;   //  The on-board Arduion LED

int Signal;                // holds the incoming raw data. Signal value can range from 0-1024
int Threshold = 550;            // Determine which Signal to "count as a beat", and which to ingore.

void setup() {
  pinMode(LED13,OUTPUT);       // pin that will blink to your heartbeat!
   Serial.begin(9600);         // Set's up Serial Communication at certain speed.}

void loop() {

  Signal = analogRead(PulseSensorPurplePin);  // Read the PulseSensor's value.
                                              // Assign this value to the "Signal" variable.
   Serial.println(Signal);                    // Send the Signal value to Serial Plotter.


   if(Signal > Threshold){  // If the signal is above "550", then "turn-on" Arduino's on-Board LED.
     digitalWrite(LED13,HIGH);
   } else {
     digitalWrite(LED13,LOW); //  Else, the sigal must be below "550", so "turn-off" this LED.
   }


delay(10);}
```

Android BLE Connectivity

EE P 523, Lecture 6

Android Connectivity

- **Bluetooth**
- **Bluetooth Low Energy (BLE)**
- **Wi-Fi**
- **Near Field Communication (NFC)**

Bluetooth

- Used to transmit data that can be used for interactive services such as audio, messaging, and telephony.

Bluetooth + Low Energy

BLE: Sensors, remote control

BT: audio streaming

Bluetooth Low Energy (BLE)

- Provides significantly lower power consumption.
- This allows Android apps to communicate with BLE devices that have stricter power requirements, such as **proximity sensors, heart rate monitors, and fitness devices**.

Bluetooth + Low Energy

- Bluetooth Low Energy (BLE), available in Android 4.3 and later, creates short connections between devices to transfer bursts of data.
- BLE remains in **sleep mode when not connected**.
- Lower bandwidth and reduced power consumption compared to Classic Bluetooth.
- It is ideal for applications such as a heart-rate monitor or a wireless keyboard.
- To use BLE, devices need to have a **chipset that supports BLE.**

BLE

- Modulation rate: 1Mbps
  (In practice, you can expect between 5-10 KB per second, depending on the limitations of the devices used.

- Typical operating range is closer **to 2 to 5 meters**.

- The higher the range the more battery consumption

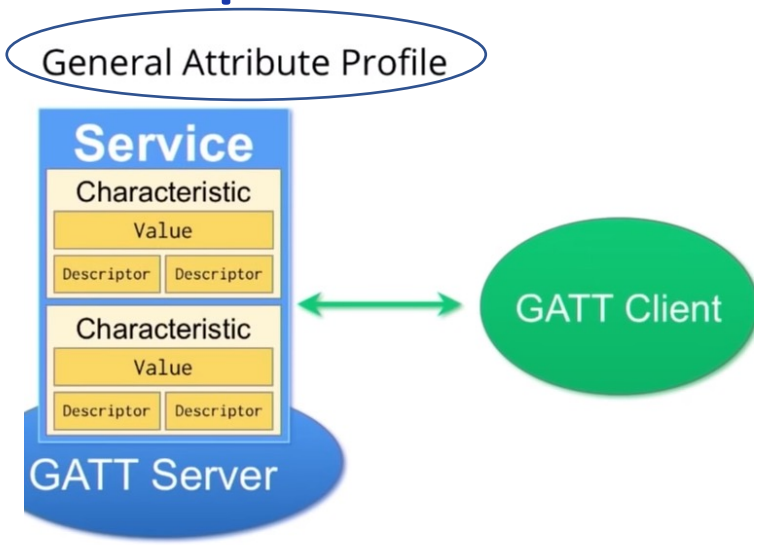
- **Bluetooth Device Address - a 48-bit number which uniquely identifies a device among peers.**

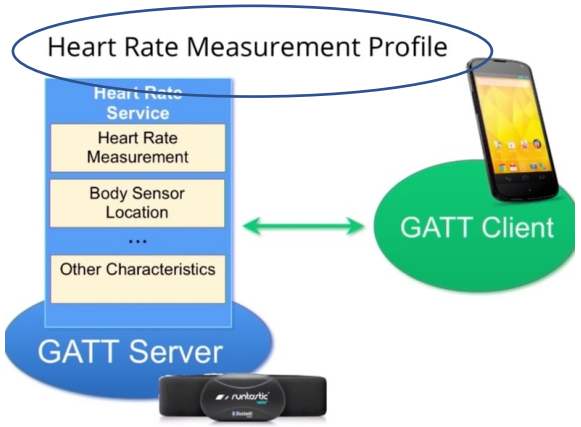
  You can think of a BDA as something similar to the MAC address in IP.

BLE Packets

A BLE device can talk to nearby devices in one of two ways: **Broadcasting** and **Connections**.

BLE Concepts for Android

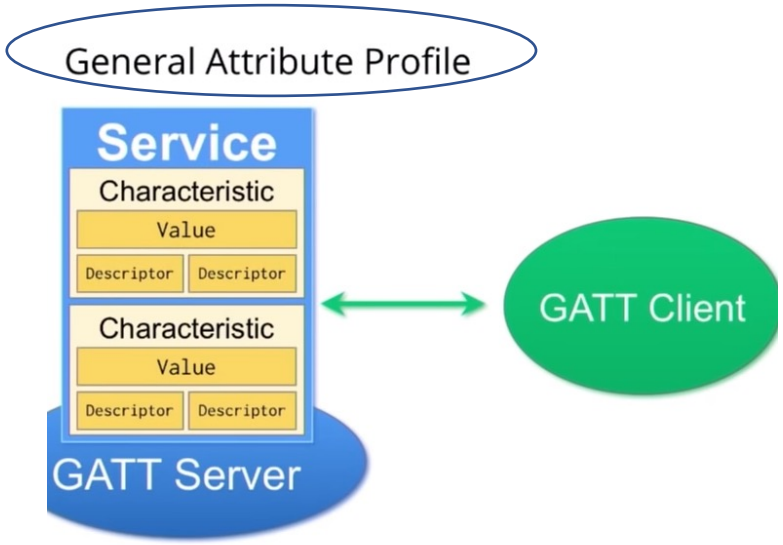


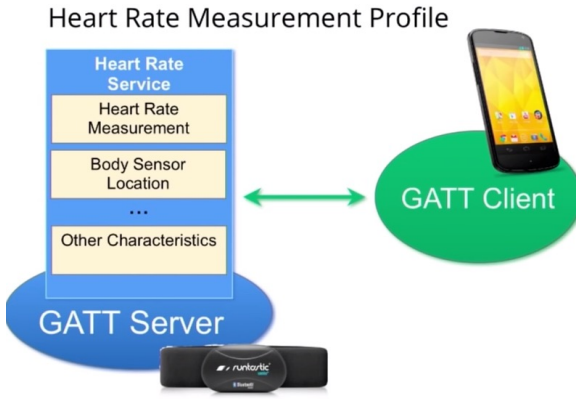


Generic Attribute Profile (GATT)

- General specification for sending and receiving short pieces of data known as "attributes" over a BLE link.
- All current Low Energy application profiles are based on GATT.
- A profile is a specification for how a device works in a particular application.
- A device can implement more than one profile. For example, a device could contain a heart rate monitor and a battery level detector.

Bluetooth Low Energy

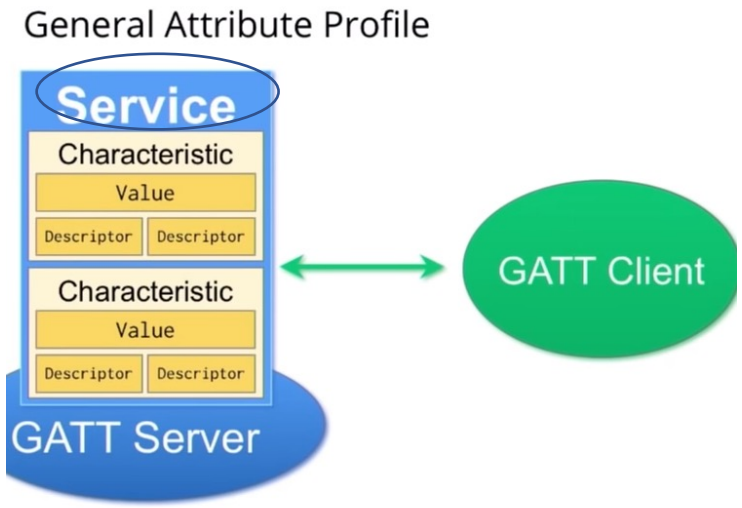




Attribute Protocol (ATT)—GATT is built on top of the Attribute Protocol (ATT).

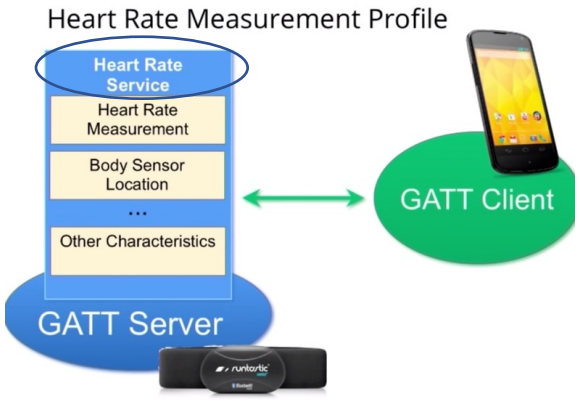
- Each attribute is uniquely identified by a Universally Unique Identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information.

Bluetooth Low Energy

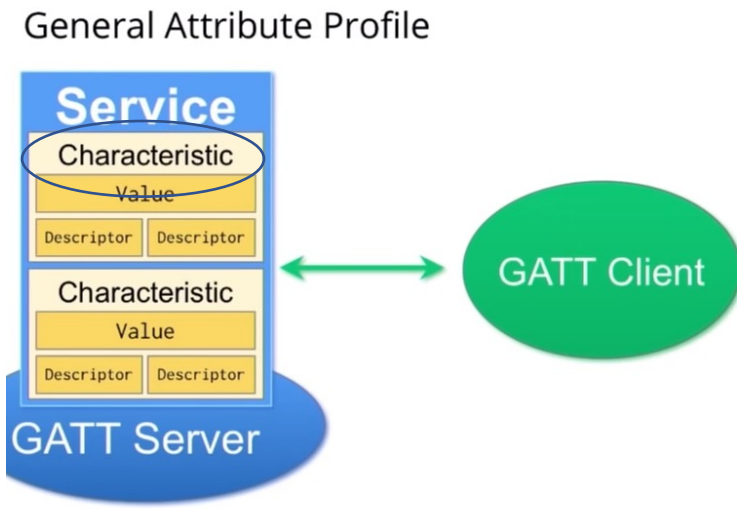


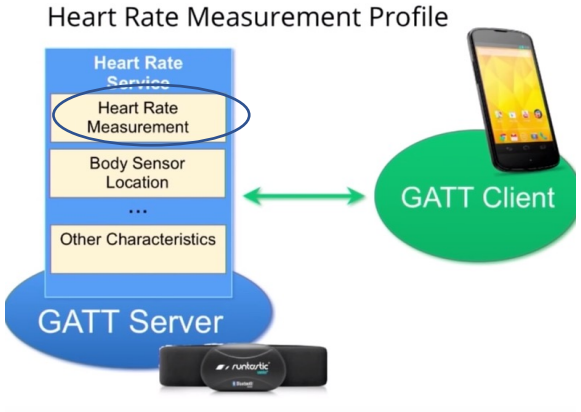
Service

- **C**ollection of characteristics.
- For example, you could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement."



Bluetooth Low Energy





- **Characteristic**
  A characteristic can be thought of as a type, analogous to a class.

- **Descriptor**
  Attributes that describe a characteristic value.

For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure that is specific to a characteristic's value.

BLE in Android

1. Request Permission
2. Set Up BLE
3. Find BLE devices
4. Connect to a GATT Server
5. Read BLE attributes
6. Receive GATT notifications
7. Close the Client App

1. Request BLE Permission

```
<uses-permission android:name="android.permission.BLUETOOTH"/>

<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

<uses-feature
android:name="android.hardware.bluetooth_le"android:required="true"/>
```

```
// Check permissions in the onCreate of your main Activity
ActivityCompat.requestPermissions(this,
    arrayOf( Manifest.permission.BLUETOOTH, Manifest.permission.BLUETOOTH_ADMIN,
Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.ACCESS_COARSE_LOCATION), 1)
```

2. Set up BLE

- Get the **Bluetooth Adapter**
  - **T**here's one Bluetooth adapter for the entire system, and your application can interact with it using this object.

```
private val adapter: BluetoothAdapter
```

- **Enable Bluetooth**

```
val adapter: BluetoothAdapter?
adapter = BluetoothAdapter.getDefaultAdapter()
if (adapter != null) {
   if (!adapter.isEnabled) {
      val enableBtIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
      startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT)

   }     }
```

3. Find BLE devices

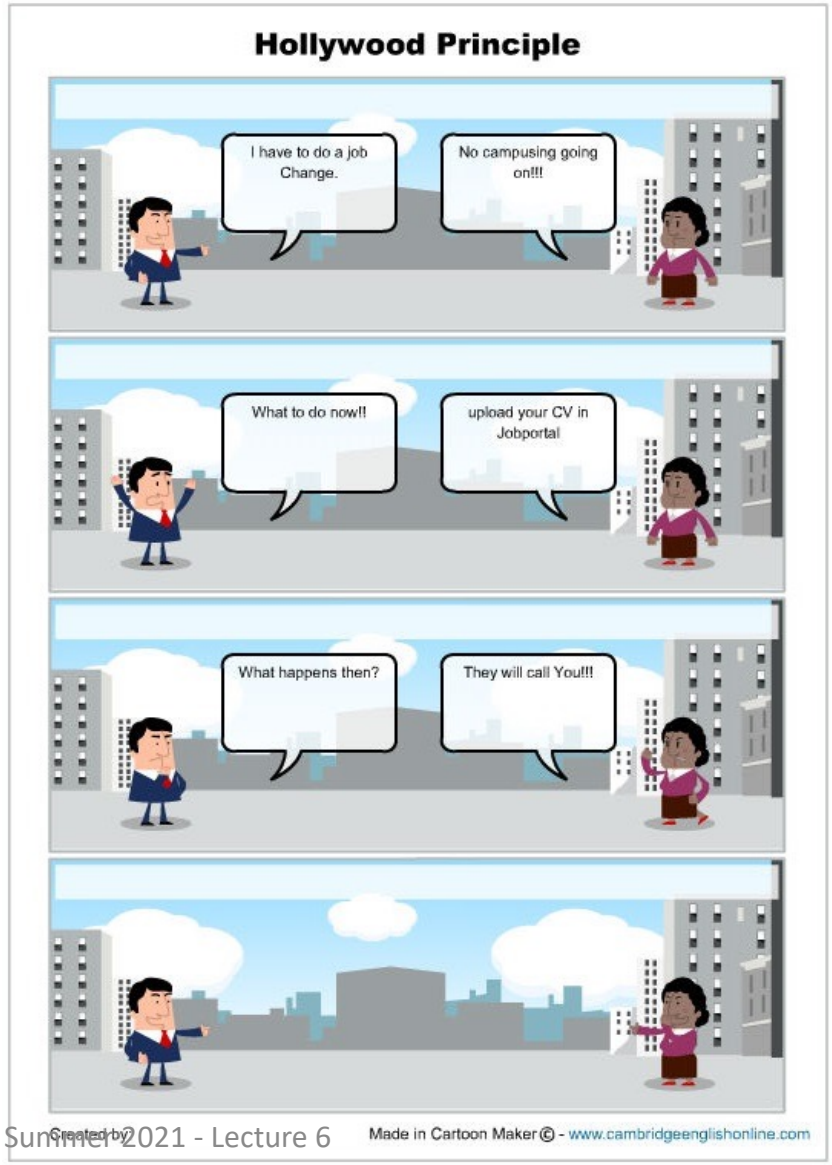
- To find BLE devices, you use the startLeScan() method.
- This method takes a BluetoothAdapter.LeScanCallbackas a parameter.
- You must implement this callback, because that is how scan results are returned.

Because **scanning is battery-intensive**, you should observe the following guidelines:

- As soon as you find the desired device, stop scanning.
- Never scan on a loop, and set a time limit on your scan. A device that was previously available may have moved out of range, **and continuing to scan drains the battery**.

Callback functions

- The concept of callbacks is to *inform a class synchronous / asynchronous if some work in another class is done*.



3. Find BLE devices

```
fun connect(v: View) {
    startScan()
}
```

```
fun connectFirstAvailable() {
    // Disconnect to any connected device.
    disconnect()
    // Stop any in progress device scan.
    stopScan()
    // Start scan and connect to first available device.
    connectFirst = true
    startScan()
}
```

```
fun startScan() {
    if (adapter != null) {
        if (!adapter.isEnabled) {
            val enableBtIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)
            context.startActivity(enableBtIntent)
        }
        adapter.startLeScan(this)
    }
}
```

4. Conect to a GATT server

- The first step in interacting with a BLE device is connecting to it— more specifically, connecting to the GATT server on the device.

- To connect to a GATT server on a BLE device, you use the connectGatt() method.

This method takes three parameters:

- a Context object,
- autoConnect (boolean indicating whether to automatically connect to the BLE device as soon as it becomes available),
- a reference to a BluetoothGattCallback:

5. Read BLE atrributes

- Once your Android app has connected to a GATT server and discovered services, it can read and write attributes, where supported.

1. Loops through available GATT Services.

2. Loops through available Characteristics

6. Receive GATT notifications

It's common for BLE apps to ask to be notified when a particular characteristic changes on the device.

```
// Setup notifications on RX characteristic changes (i.e. data received).
// First call setCharacteristicNotification to enable notification.
if (!gatt.setCharacteristicNotification(rx, true)) {
    // Stop if the characteristic notification setup failed.
    Log.e("BLE", "characteristic notification setup failed")
    connectFailure()
    return
}
```

Once notifications are enabled for a characteristic, an onCharacteristicChanged() callback is triggered if the characteristic changes on the remote device:

```
override fun onCharacteristicChanged(gatt: BluetoothGatt, characteristic:
BluetoothGattCharacteristic) {
    super.onCharacteristicChanged(gatt, characteristic)
    notifyOnReceive(this, characteristic)
}
```

7. Close the Client App

Once your app has finished using a BLE device, it should call close() so the system can release resources appropriately:

```
fun close() {
    bluetoothGatt?.close()
    bluetoothGatt = null
}
```

```
else if (newState == BluetoothGatt.STATE_DISCONNECTED) {
    // Disconnected, notify callbacks of disconnection.
    rx = null
    tx = null
    notifyOnDisconnected(this)
}
```

Callback functions

- Class A calls Class B to get some work done in a Thread.
- If the Thread finished the work, it will inform Class A over the callback and provide the results. So there is no need for polling or something. You will get the results as soon as they are available.

In the BLE adapter Class

```
private val callbacks: WeakHashMap<Callback, Any>
```

```
notifyOnConnected(this)
```

```
private fun notifyOnConnected(uart: BLE) {
    for (cb in callbacks.keys) {
        cb?.onConnected(uart)
    }
}
```

In the Main Activity

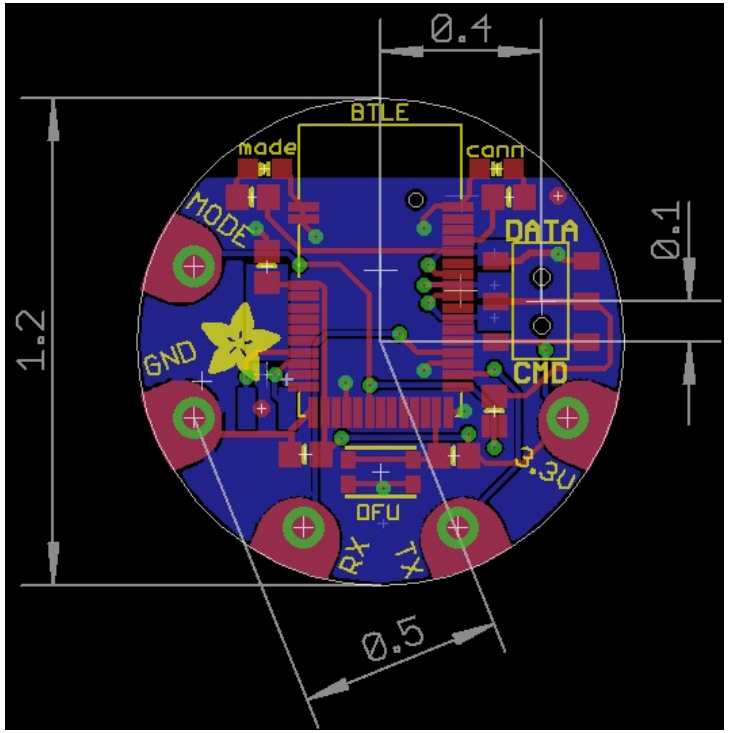
```
override fun onConnected(ble: BLE)
{
    writeLine("Connected!")

}
```

Android-Arduino Interaction

EE P 523, Lecture 6

Connecting Android with Arduino

Flora Wearable Bluefruit LE Module



- ARM Cortex M0 core running at 16MHz
- 256KB flash memory
- 32KB SRAM
- Transport: **UART @ 9600 baud**
- Bootloader with support for safe OTA (Over The Air) firmware updates
- Easy AT command set to get up and running quickly
- Diameter: 30.5mm / 1.2"
- Height: 4mm / 0.16"
- Weight: 2.5g

Universal Asynchronous Receiver Transmitter (UART) (link)

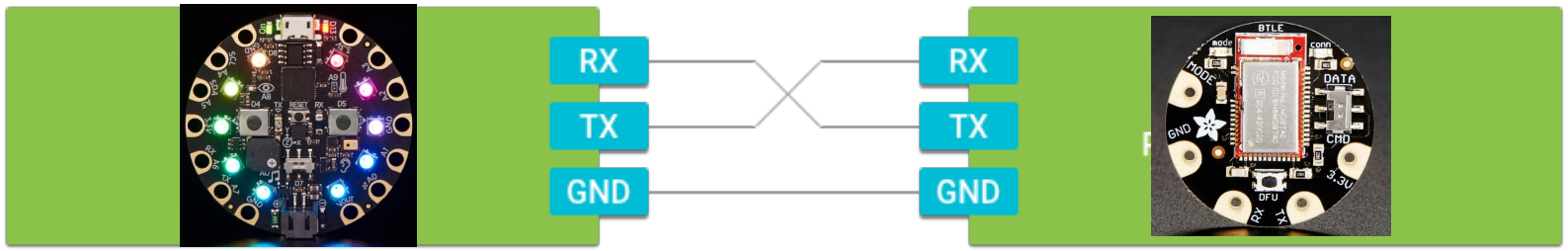
Generic interface for exchanging raw data with a peripheral device.

- ***Universal***: both the data transfer speed and data byte format are configurable.
- ***Asynchronous:*** there are no clock signals present to synchronize the data transfer between the two devices.

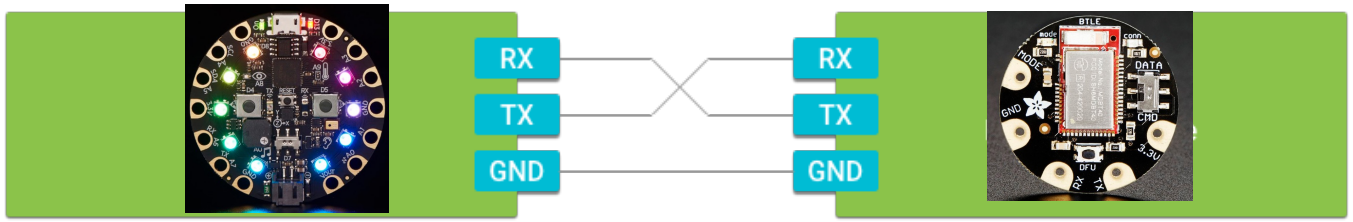
- The device hardware collects all incoming data in a first-in first-out (FIFO) buffer until read by your app.

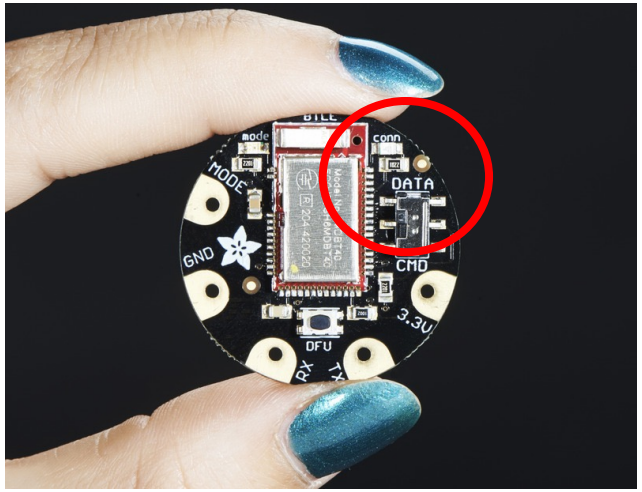
- UART data transfer is ***full-duplex***, meaning data can be sent and received at the same time.

- Typically faster than I^2C, but the lack of a shared clock means that both devices must agree on a common data transfer rate that each device can adhere to independently with minimal timing error



Universal Asynchronous Receiver Transmitter UART

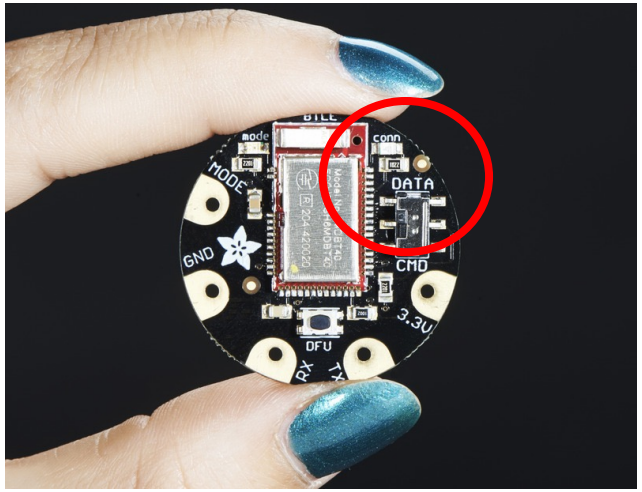




Don't forget, set the Bluefruit LE module switch to the DATA position!

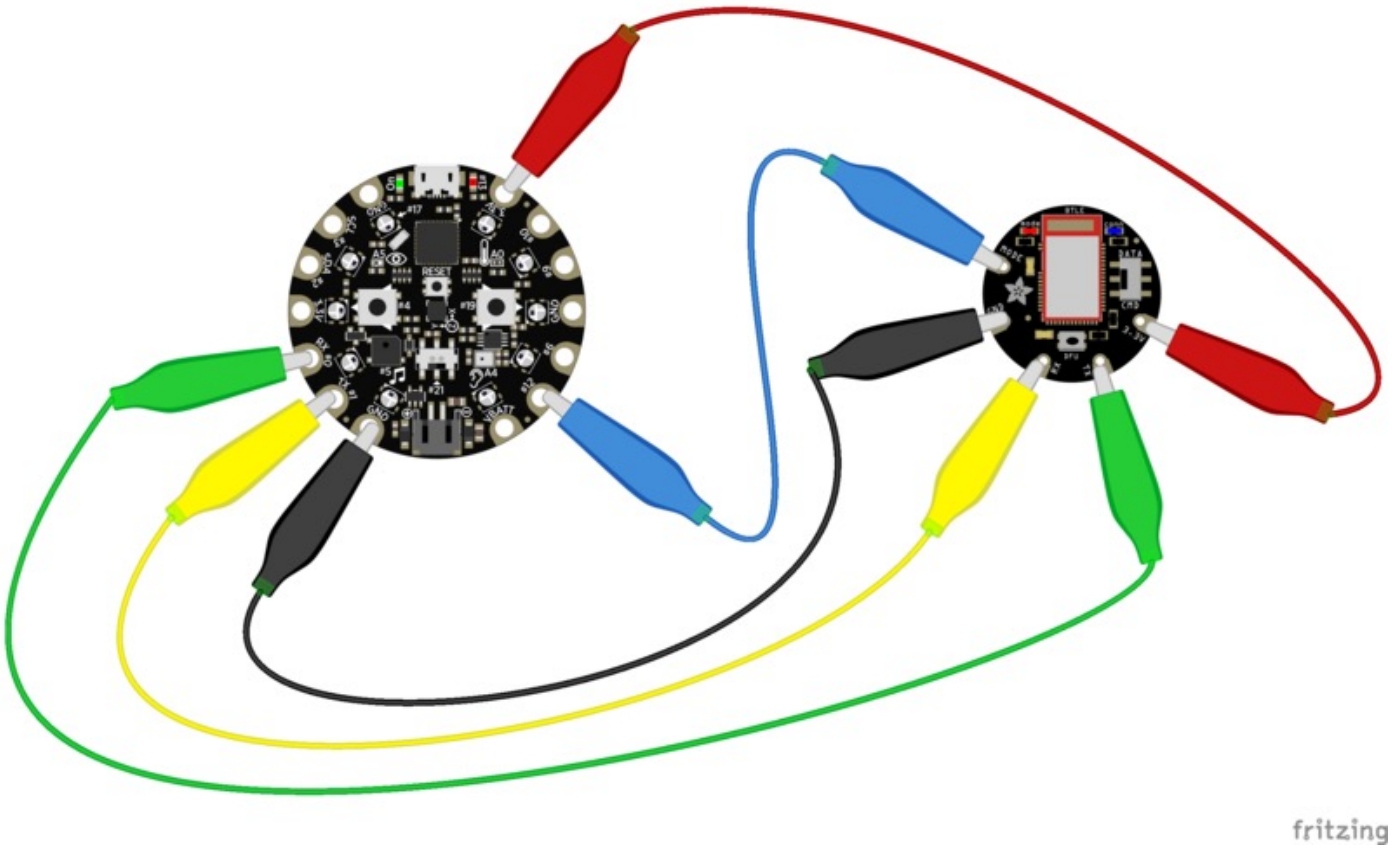
Command Mode

- Enter a variety of Hayes AT style commands to configure the device or retrieve basic information about the module of BLE connection(set the mode selection switch to **CMD** or setting the MODE pin to a high voltage)



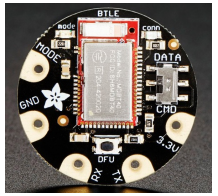
Don't forget, set the Bluefruit LE module switch to the DATA position!

Flora Wearable Bluefruit LE Module



UART UUID for BLE shield

```
companion object {

    // UUIDs for UART service and associated characteristics.
    var UART_UUID = UUID.fromString("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
    var TX_UUID = UUID.fromString("6E400002-B5A3-F393-E0A9-E50E24DCCA9E")
    var RX_UUID = UUID.fromString("6E400003-B5A3-F393-E0A9-E50E24DCCA9E")

    // UUID for the UART BTLE client characteristic which is necessary for
notifications.
    var CLIENT_UUID = UUID.fromString("00002902-0000-1000-8000-00805f9b34fb")

    // UUIDs for the Device Information service and associated characteristics.
    var DIS_UUID = UUID.fromString("0000180a-0000-1000-8000-00805f9b34fb")
    var DIS_MANUF_UUID = UUID.fromString("00002a29-0000-1000-8000-00805f9b34fb")
    var DIS_MODEL_UUID = UUID.fromString("00002a24-0000-1000-8000-00805f9b34fb")
    var DIS_HWREV_UUID = UUID.fromString("00002a26-0000-1000-8000-00805f9b34fb")
    var DIS_SWREV_UUID = UUID.fromString("00002a28-0000-1000-8000-00805f9b34fb")

}
```



https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v15.0.0%2Fble_sdk_app_nus_eval.html

Nordic Semiconductors

Android -> Arduino

In the Main Activity

```
ble!!.send("red")
```

```
// Send data to connected UART device.
fun send(data: ByteArray?) {
    if (tx == null || data == null || data.size == 0)
{
        // Do nothing if there is no connection or
message to send.
        return
    }
    // Update TX characteristic value.  Note the
setValue overload that takes a byte array must be
used.
    tx!!.value = data
    //        writeInProgress = true; // Set the
write in progress flag
    Log.d("BLE", "writing")
    gatt!!.writeCharacteristic(tx)
    try {
        writeInProgress.acquire()
    } catch (e: InterruptedException) {
        e.printStackTrace()
    }

}
```

```
int c = ble.read();
Serial.print((char)c); // Show
in the serial monitor

received += (char)c;

if(red == received){

      //Turn the LED red light

}
```

Android < - Arduino

```
sensorTemp = CircuitPlayground.temperature();

//Send data to Android Device
   char output[8];
   String data = "";
   data += sensorTemp;
   data.toCharArray(output,8);
   ble.print(data);
```

```
override fun onReceive(ble: BLE, rx: BluetoothGattCharacteristic) {
    writeLine("Received value: " + rx.getStringValue(0))

}
```

Change BLE shield name: Android Side

In a scenario with multiple BLE active with the same name:

```
// Get Bluetooth
ble = BLEControl(applicationContext, DEVICE_NAME)
```

```
companion object {
    private val DEVICE_NAME = "YodaBest"
    private val REQUEST_ENABLE_BT = 0
}
```

Change BLE shield name: Arduino Side

In a scenario with multiple BLE active with the same name:

```
#define NEW_DEVICE_NAME "PMP590 is awesome"
```

1. Factory reset to make sure everything is in a known state Apply the new name

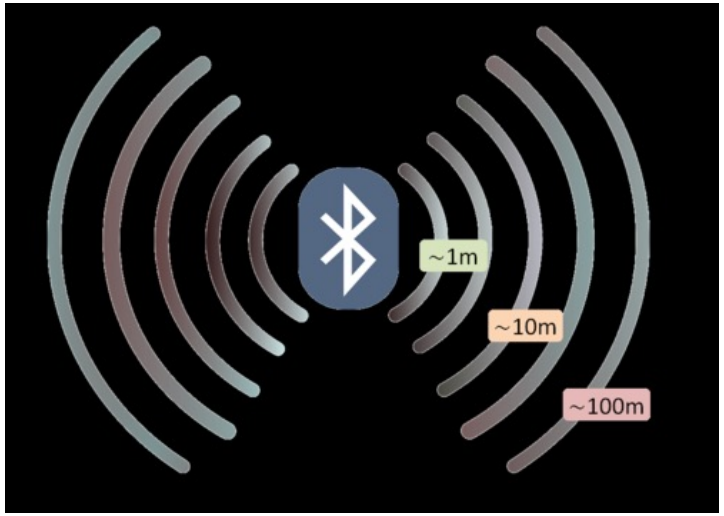
```
if ( ! ble.factoryReset() ){
      error(F("Couldn't factory reset"));
    }
```

2. Apply new name and wait for ok (see code on Arduino IDE)

Received Signal Strenght Indication (RSSI)

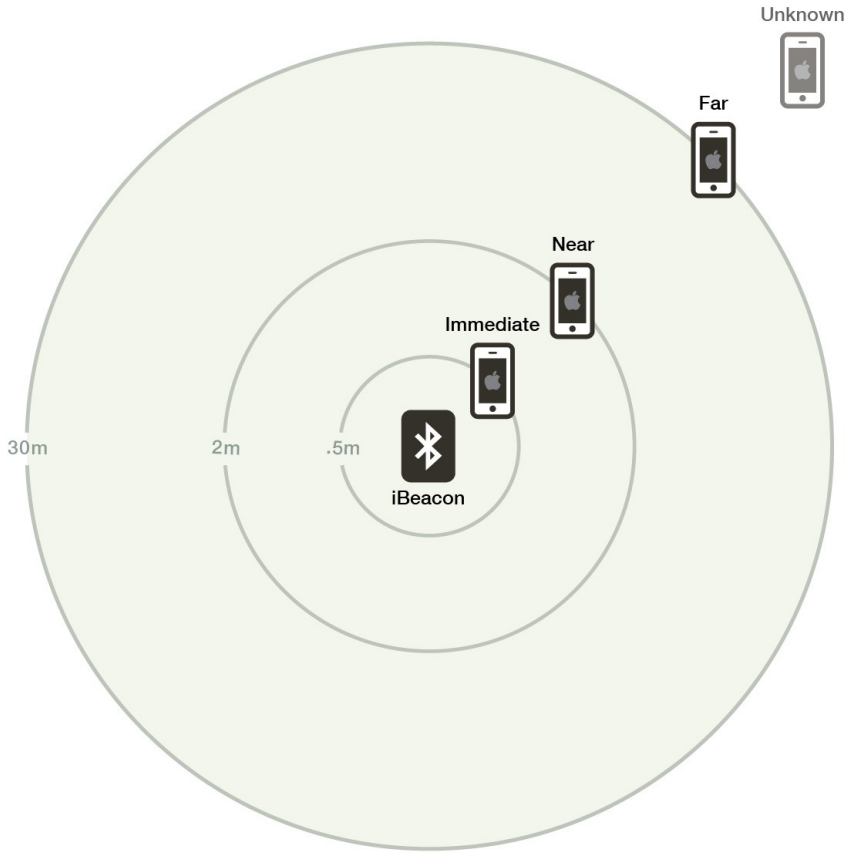
- RSSI stands for Received Signal Strength Indicator. It is the strength of the beacon's signal as seen on the receiving device, e.g., a smartphone. **The signal strength depends on distance and Broadcasting Power value**. At maximum Broadcasting Power (+4 dBm) the RSSI ranges from -26 (a few inch

- Measured Power: 1 Meter RSSI
- es) to -100 (40-50 m distance).

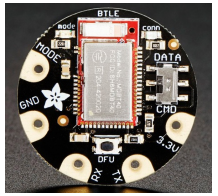


RSSI: Distance phone-arduino

$$Distance = 10^{\left(\frac{Reference\ power - RSSI}{10 * N}\right)}$$



BLE shield



Datasheet

https://cdn-shop.adafruit.com/product-files/2267/MDBT40-P256R.pdf

1.2 Features

. 2.4GHZ transceiver

. -93dbm sensitivity in Bluetooth low energy mode

. TX Power -20 to +4dbm

. RSSI (1db resolution)

. ARM Cortex – M0 32 bit processor

. Serial Wire Debug (SWD)

. S100 series SoftDevice ready

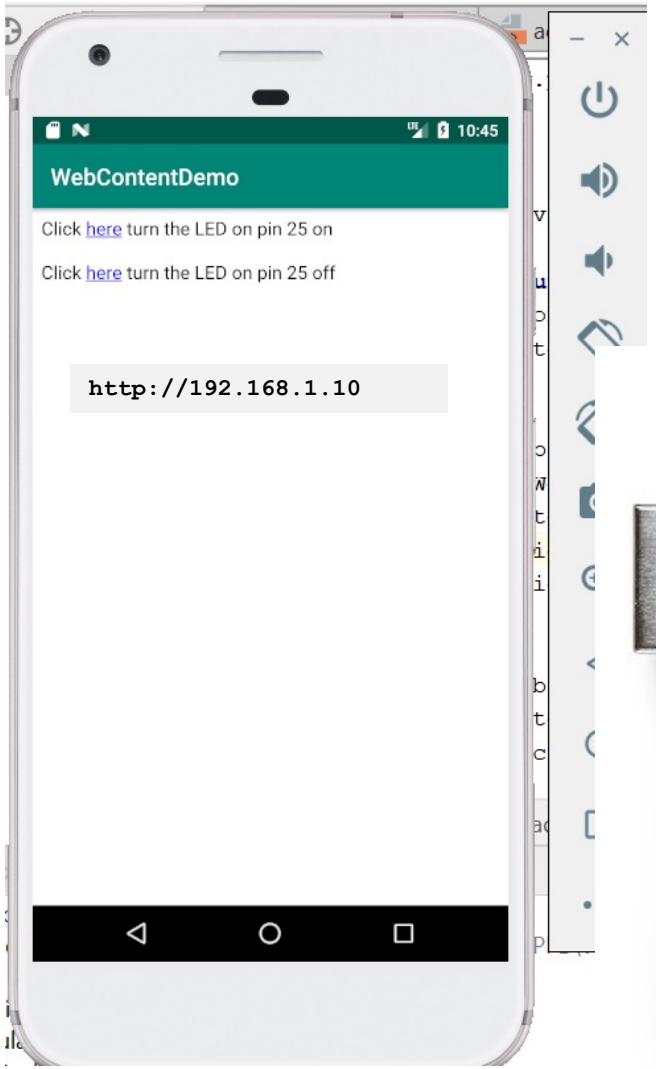
RSSI: Distance phone-arduino

BLEControl.kt class

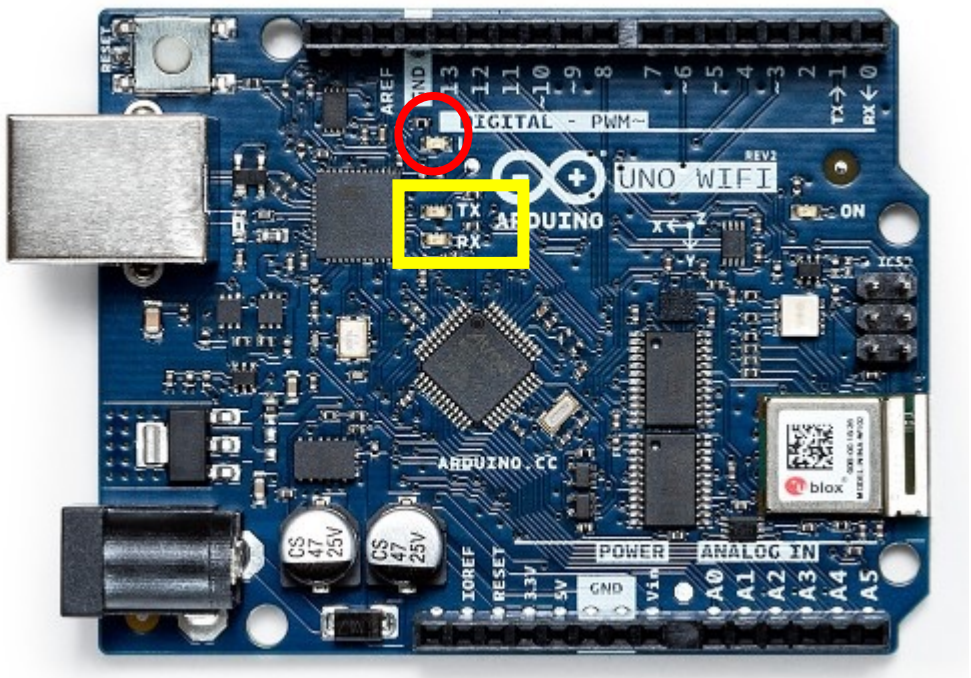
```
fun getRSSI(){
    gatt!!.readRemoteRssi()
}
override fun onReadRemoteRssi(gatt: BluetoothGatt, rssi: Int, status:Int) {
    super.onReadRemoteRssi(gatt, rssi, status)
    if (status == BluetoothGatt.GATT_SUCCESS) {
        mRSSI = rssi
        notifyRSSIread(this, mRSSI)
    } else {

    }
}
```

```
private fun notifyRSSIread(uart:BLEControl,rssi:Int){
    for (cb in callbacks.keys){
        cb?.onRSSIread(uart,rssi)
    }
}
```

Connecting Arduino to WiFi



See example on Canvas
WiFiBlinkDemo

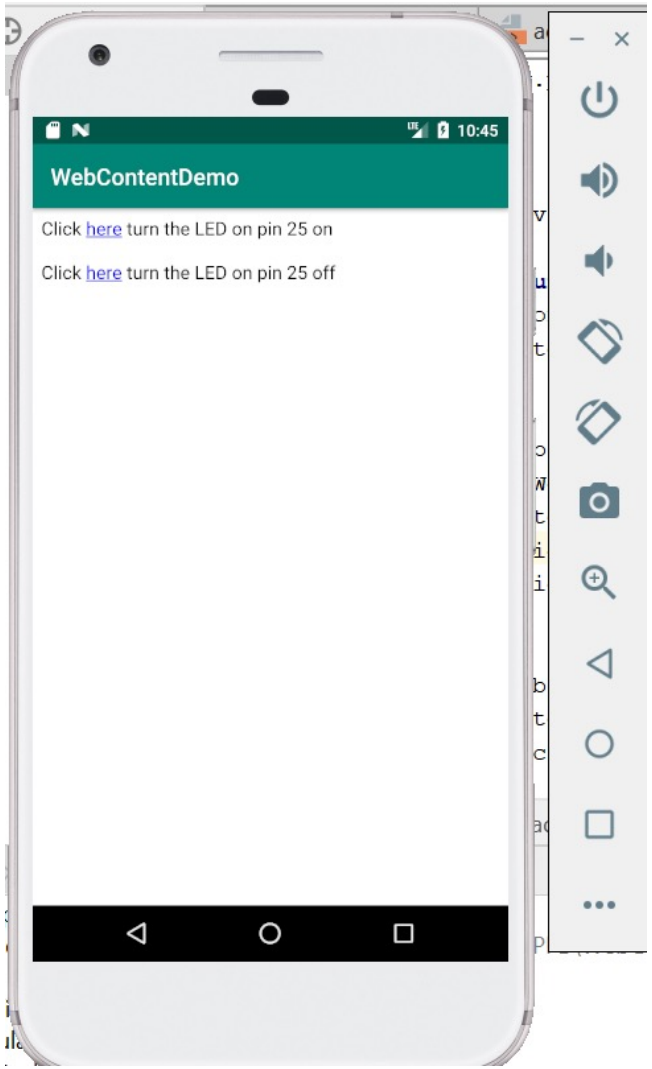


Connecting Arduino to WiFi

See example on Canvas
WiFiBlinkDemo

```
// if the current line is blank, you got two newline characters in a row.
      // that's the end of the client HTTP request, so send a response:
      if (currentLine.length() == 0) {
        // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
        // and a content-type so the client knows what's coming, then a blank line:
        client.println("HTTP/1.1 200 OK");
        client.println("Content-type:text/html");
        client.println();

        // the content of the HTTP response follows the header:
        client.println();
        client.println("<br>");
        client.print("Click <a href=\"/H\">here</a> turn the LED on pin 25 on<br>");
        client.println("<br>");
        client.print("Click <a href=\"/L\">here</a> turn the LED on pin 25 off<br>");

        // The HTTP response ends with another blank line:
        client.println();
        // break out of the while loop:
        break;
      }
```

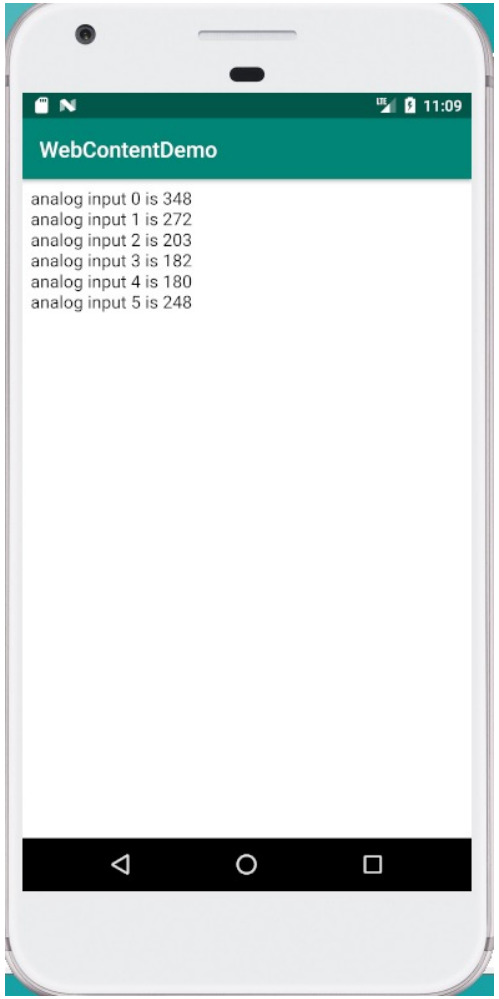
Connecting Arduino to WiFi



See example on Canvas
WiFiBlinkDemo

```
// Check to see if the client request was "GET /H" or "GET /L":
      if (currentLine.endsWith("GET /H")) {
        digitalWrite(25, HIGH); // GET /H turns the LED on

      }
      if (currentLine.endsWith("GET /L")) {
        digitalWrite(25, LOW); // GET /L turns the LED off
      }
```

Connecting Arduino to WiFi



See example
WiFiAnlogDemo

```
for (int analogChannel = 0; analogChannel < 6;
analogChannel++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
            client.print(sensorReading);
            client.println("<br />");
          }
```

Connecting Arduino to WiFi

client disonnected
new client
GET /favicon.ico HTTP/1.1
Host: 192.168.1.10
Connection: keep-alive
User-Agent: Mozilla/5.0 (Linux; Android 7.0; Android SDK built for x86 Build/NYC; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/51.0.2704.90 Mobile Safari/537.36
Accept: */*
Referer: http://192.168.1.10/H
Accept-Encoding: gzip, deflate
Accept-Language: en-US
X-Requested-With: edu.uw.pmpee590.webcontentdemo

client disonnected

NFC protocol

- NFC is a branch of High-Frequency (HF) RFID, and both operate at the 13.56 MHz frequency.
- NFC is designed to be a secure form of data exchange, and an NFC device is capable of being both an NFC reader and an NFC tag.
- This unique feature allows NFC devices to communicate peer-to-peer

NFC protocol

- As a finely honed version of HF RFID, near-field communication devices have taken advantage of the short read range limitations of its radio frequency.

- Because NFC devices must be in close proximity to each other, usually no more than a few centimeters, it has become a popular choice for secure communication between consumer devices such as smartphones.

NFC in Android

Android framework APIs are based around a NFC Forum standard called NDEF (NFC Data Exchange Format).

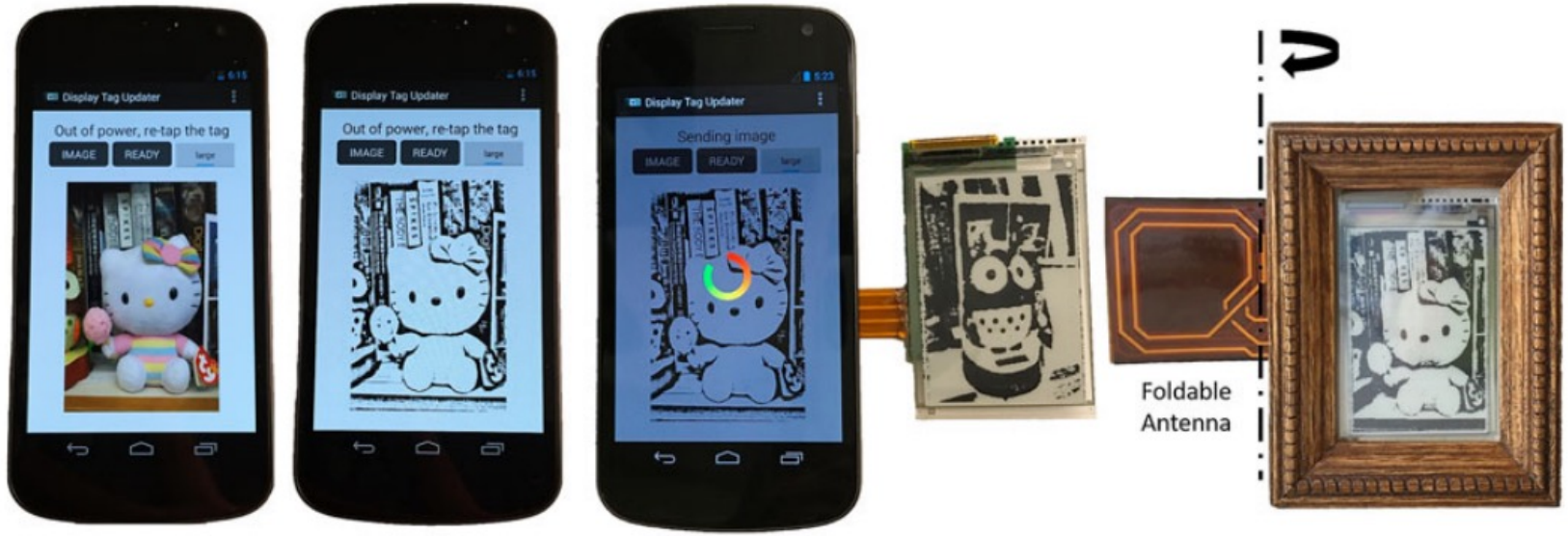
Android-powered devices with NFC simultaneously support three main modes of operation:

1.Reader/writer mode
NFC device read and/or write passive NFC tags and stickers.

2. P2P mode
NFC device exchange data with other NFC peers;

3.Card emulation mode,
NFC device itself to act as an NFC card.
The emulated NFC card can then be accessed by an external NFC reader, such as an NFC point-of-sale terminal.

NFC WISP



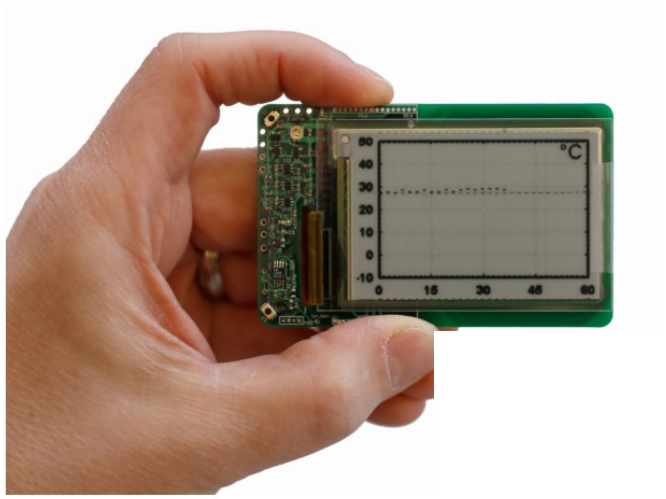
Step 1:
Choose photo and E-ink type

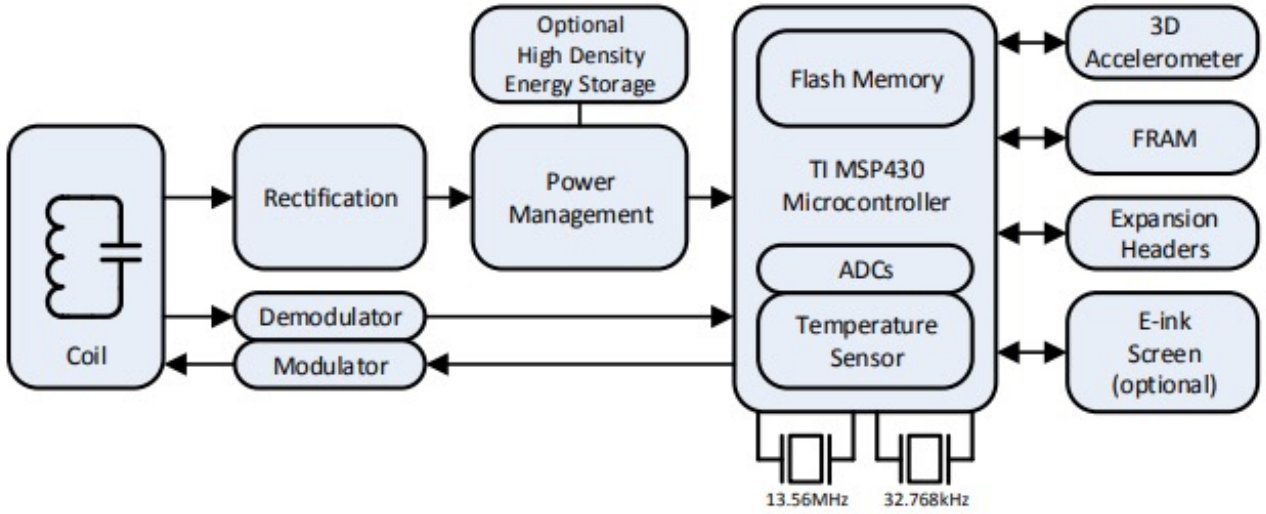
Step 2:
Pinch & crop photo and convert

Step 3:
Tap to NFC-WISP's antenna and wait for image updating within 3s

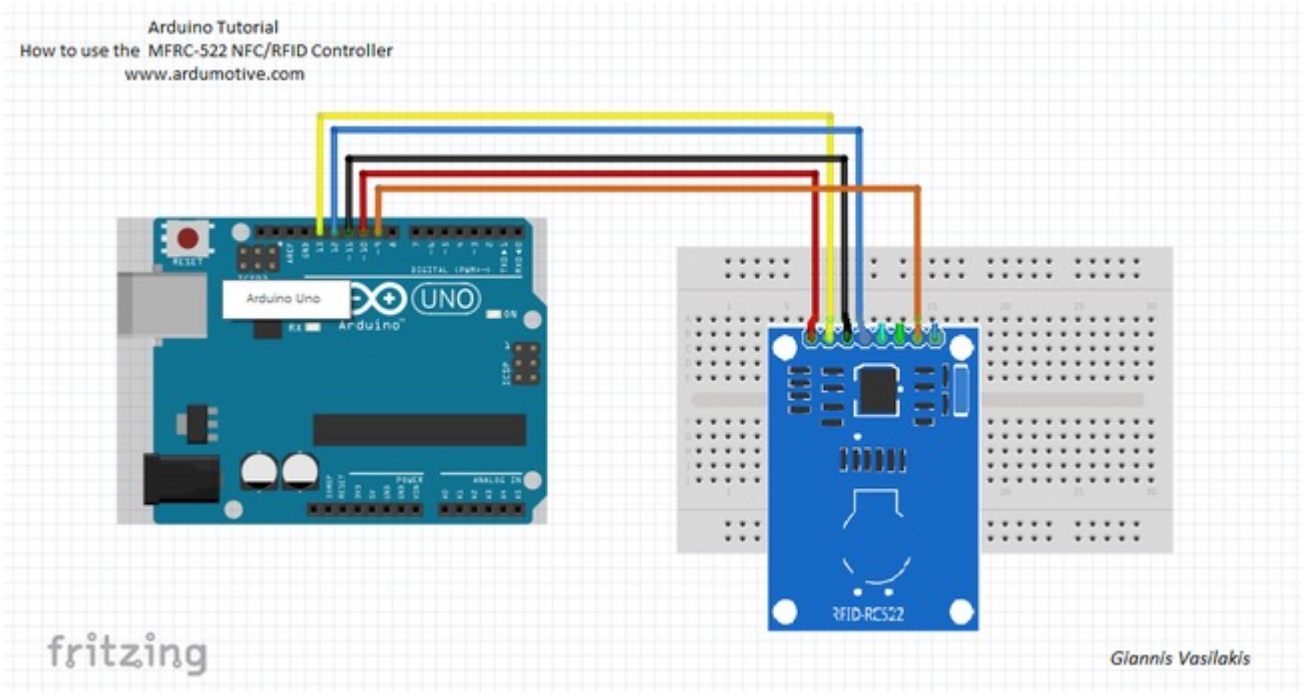
Application example:
Battery-free digital frame

NFC WISP




Optional
High Density
Energy Storage
Flash Memory
3D
Accelerometer
Rectification
Power
Management
TI MSP430
Microcontroller
FRAM
Expansion
Headers
ADCs
Coil
Demodulator
Temperature
Sensor
E-ink
Screen
(optional)
Modulator
13.56MHz
32.768kHz

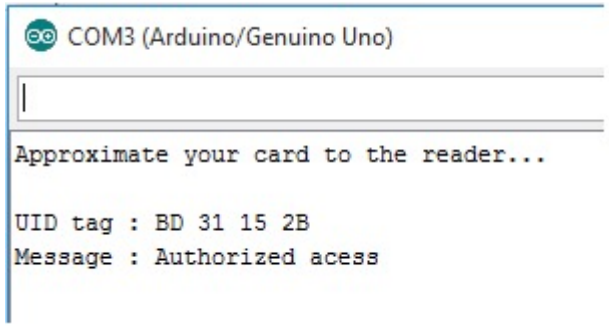
NFC in Arduino



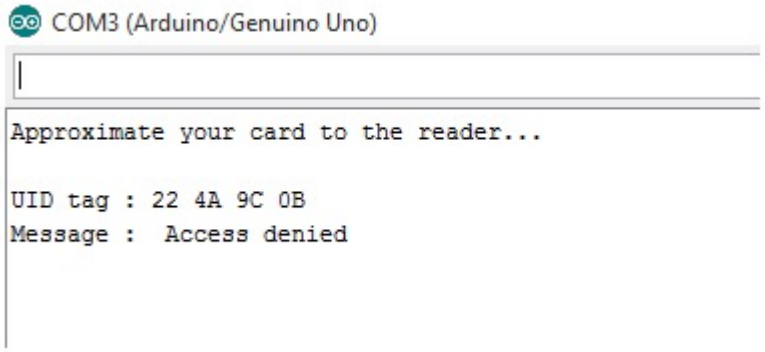
- The microcontroller and card reader uses SPI for communication
- The card reader and the tags communicate using a 13.56MHz electromagnetic field. (ISO 14443A standart tags)

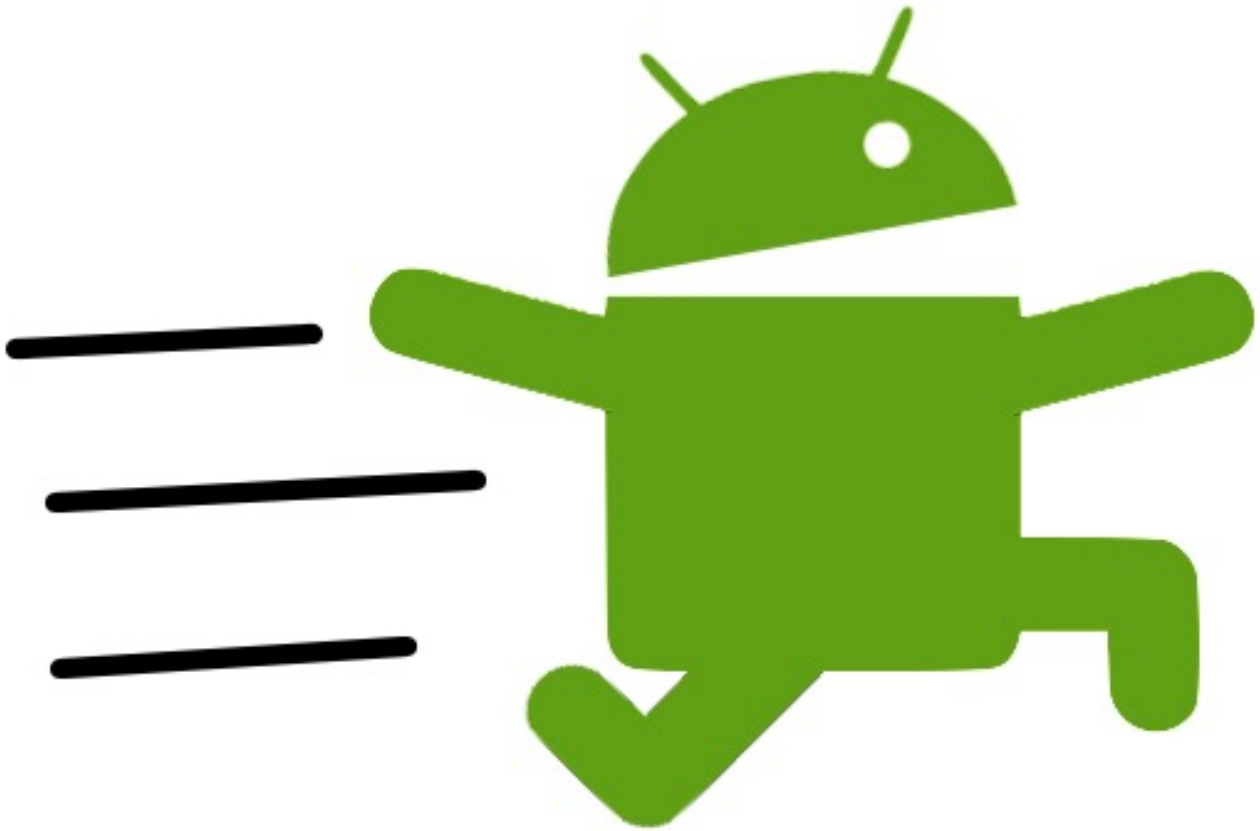
NFC in Arduino

Approximate the card you've chosen to give access and you'll see:



If you approximate another tag with another UID, the denial message will show up:





Your Questions

?