Arno Puder

Internet of Things

Introduction to Arduino-based Microcontrollers

Agenda

1. Platforms
2. Electronics 101
3. Arduino Sketches
4. Infrared
5. Cloud
6. Hardware Interfaces
7. Sensors & Actuators
8. Security

Companion page: https://github.com/apuder/iot-workshop

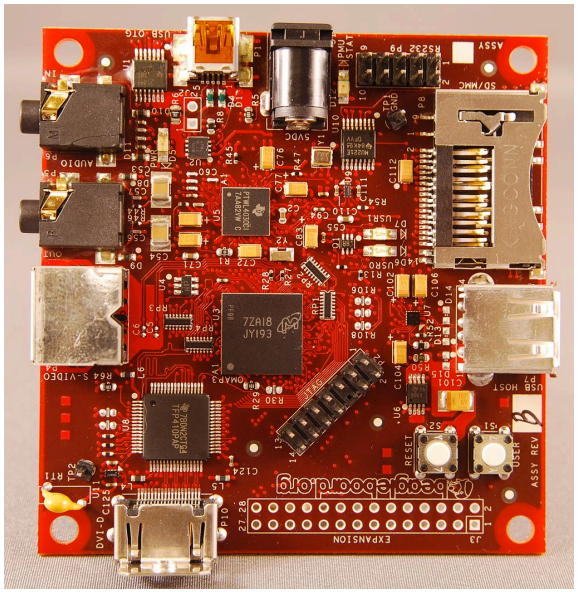
Internet of Things

- IoT:
  - Term coined in 1999.
  - Inter-networking of physical devices.
  - Allows objects to be sensed or controlled remotely across existing network infrastructure.
- First smart device: coke machine that reported its inventory (CMU, 1982)
- Conceptual foundation in Mark Weiser's vision of Ubiquitous Computing (1991)
- IoT evolved due to a convergence of technologies such as ubiquitous wireless communication, real-time analytics, machine learning, commodity sensors, and embedded systems.
- Application areas: environmental monitoring, healthcare, energy management, home automation, …
- Platforms such as Raspberry Pi and Arduino enable DIY of IoT devices.
- One of the biggest challenges is security.

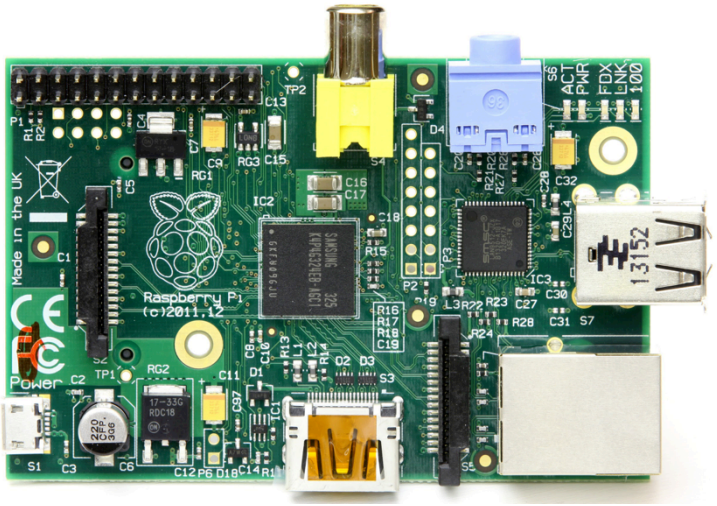
BeagleBoard

beagleboard.org

- Low-power, Open Source, single board computer produced by TI since 2008.
- Cost: $95 - $149
- Processor: ARM Cortex-A8 CPU.
- Frequency: 600 MHz − 1GHz
- Memory: 128 MB − 512 MB
- Power consumption: 2 W.
- Runs Linux, Minix, FreeBSD, OpenBSD.
- Ports: USB, RS232, I2C, SPI, …
- BeagleBone:
  - Since 2011.
  - Barebone development board.
  - microSD slot.
  - On-chip Ethernet.
  - Capes: LCD touchscreen, battery, breakout, …



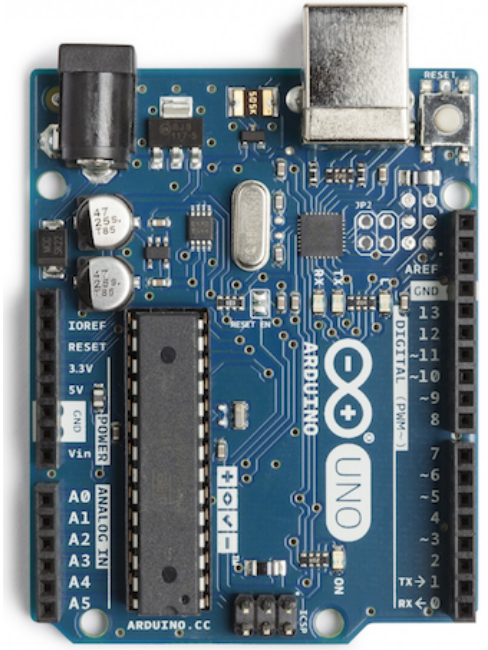
Raspberry Pi



- Credit card sized computer developed by Raspberry Pi Foundation.
- Developed to promote teaching of basic computer science.
- Its very low cost ($25) and low powered.
- It comes with Broadcom SoC (System-on-Chip) which includes an ARM CPU and a GPU.
- 256MB to 1GB RAM.
- SD card to store the OS and other data.
- Runs Linux.
- Features common interfaces: GPIO, UART, I2C, and SPI.
- Homepage: https://www.raspberrypi.org/

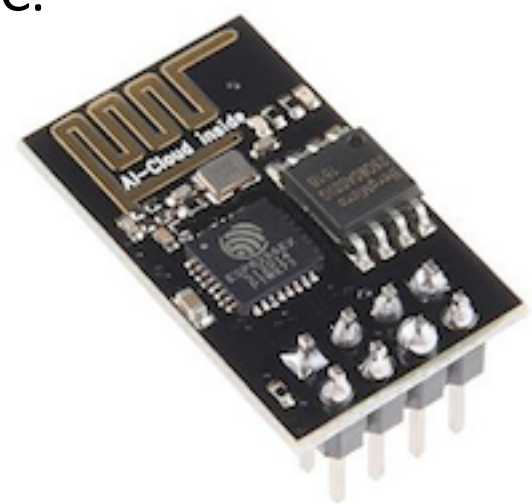
Arduino

- Open source specification for a programmable microcontroller.
- Started in 2003 in Italy.
- Main purpose is to interface with sensors and actuators via GPIO/I2C and SPI.
- CPUs: Atmel AVR, ARM Cortex, Intel Quark (x86)
- Does not run an operating systems!
- Arduino Uno R3:
  - CPU: ATmega328P @ 16 MHz
  - Flash memory: 32 kB
  - SRAM: 2 kB
  - Interface: GPIO, UART, SPI, I2C.
  - Size: 68 x 53 mm
  - Power consumption: 225 mW
  - Arduinio Pro Mini: 15 µW - 12 mW (4 years on a 9V battery)
- Homepage: https://www.arduino.cc/



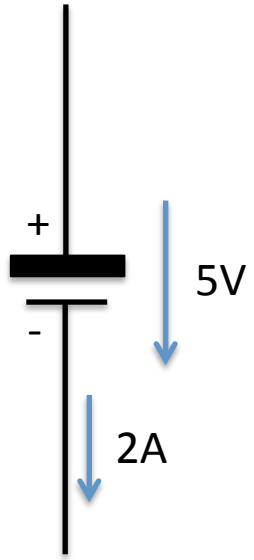
ESP8266

- Low-cost microcontroller including full TCP/IP stack.
- Produced by Shanghai-based Chinese manufacturer, Espressif Systems.
- CPU: 32-bit RISC Tensilica Xtensa L106 @ 80 MHz
- RAM: 64 kB for instructions; 96 kB of data.
- WiFi:
  - IEEE 802.11 b/g/n
  - WEP or WPA/WPA2 authentication, or open networks.
- Interface: 16 GPIO pins, SPI, I2C, UART, 10-bit ADC.
- Power consumption: 250 µW to 660 mW
- Size: 25 x 15 mm



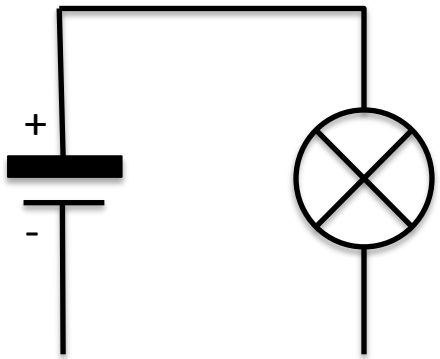
Current/Voltage

- Electronic devices depend on the movement of electrons.
- The amount of electrons moving from one molecule to another is called **Current** which is measured in **Amps.**
- The difference in potential (the number of free electrons) between two points is called **Electromotive Force** (EMF) which is measured in **Volts.**
- Plumbing analogy: voltage is equivalent to the water pressure, the current is equivalent to the flow rate.

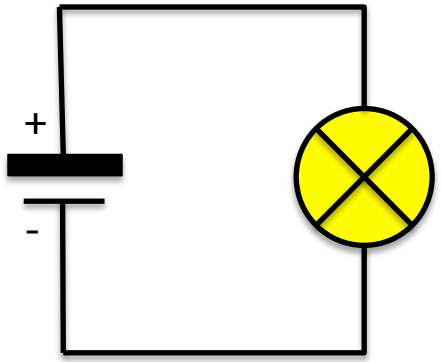


Electronic Circuit

- Electrons flow from the negative terminal of the battery through the circuit to the positive terminal.
- There needs to be a complete circuit for current to flow.



No current
Light is off



Current will flow
Light is on

Resistance

- Materials that allow easy movement of electrons are called **Conductors.**
  - E.g., copper, silver, gold, aluminum.
- Materials that do not allow easy movement of electrons are called **Insulators.**
  - E.g., glass, paper, rubber.
- Materials that are neither good conductors or good inductors provide **Resistance** to the movement of electrons.
  - E.g., carbon.
- The unit of resistance is **Ohm** (Ω).

Ohm's Law

- A (linear) resistor is an element for which:

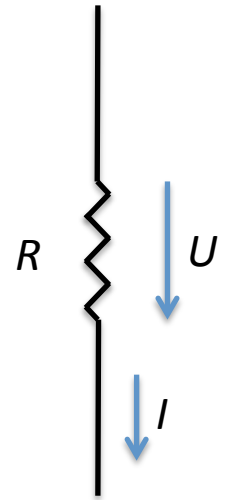
$$I = \frac{U}{R}$$

where:

I = Current in Amps

U = EMF in Volts

R = Resistance in Ohms



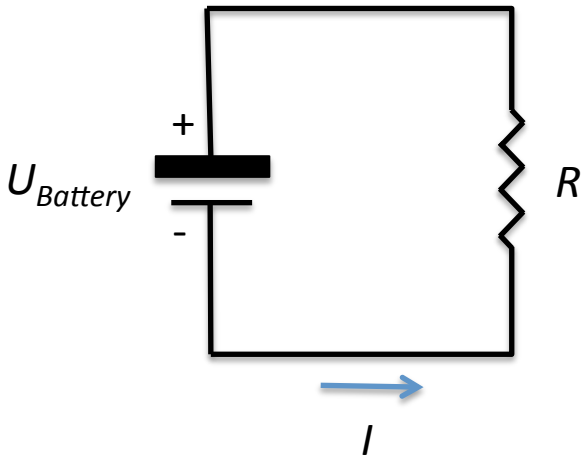
- The equation is known as **Ohm's Law**.
- Plumbing analogy: diameter of the pipe.

Ohm's Law: Example

- $U_{Battery}$ = 5V
- R = 100Ω
- What is the current?

$$I = \frac{5V}{100\Omega}$$

- I = 0.05A = 50mA



Open and Short Circuits

- An *open circuit* between A and B means $I = 0\text{A}$
- Voltage across an open circuit: any value.
- An open circuit is equivalent to $R = \infty\ \Omega$

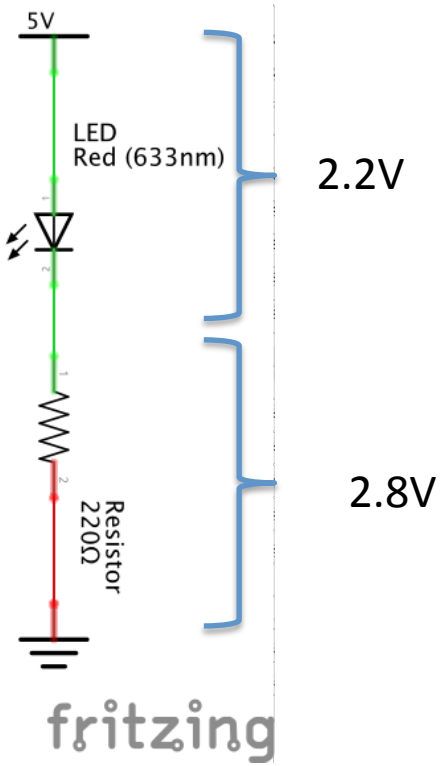
- A *short circuit* between A and B means $U = 0\text{V}$
- Current through a short circuit: any value.
- A short circuit is equivalent to $R = 0\Omega$

LED and Resistance

- Compute the resistance to protect a LED (Light Emitting Diode) from burnout:

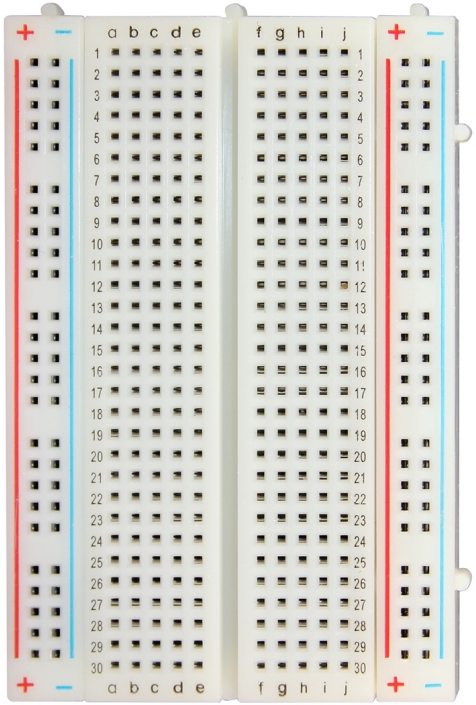
$$R = \frac{(V_S - V_{LED})}{I_{LED}}$$

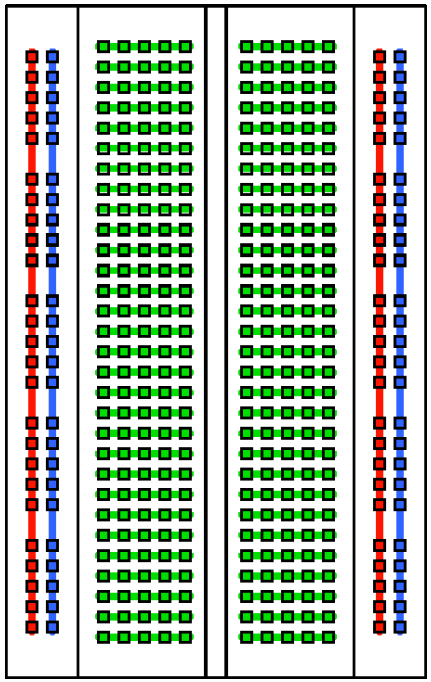
- V_S = 5V
- V_{LED} = 2.2V (forward drop)
- I_{LED} = 16 mA
- R = 175Ω
- R_{approx} = 220Ω (red-red-brown)



http://www.ohmslawcalculator.com/led-resistor-calculator

Breadboards

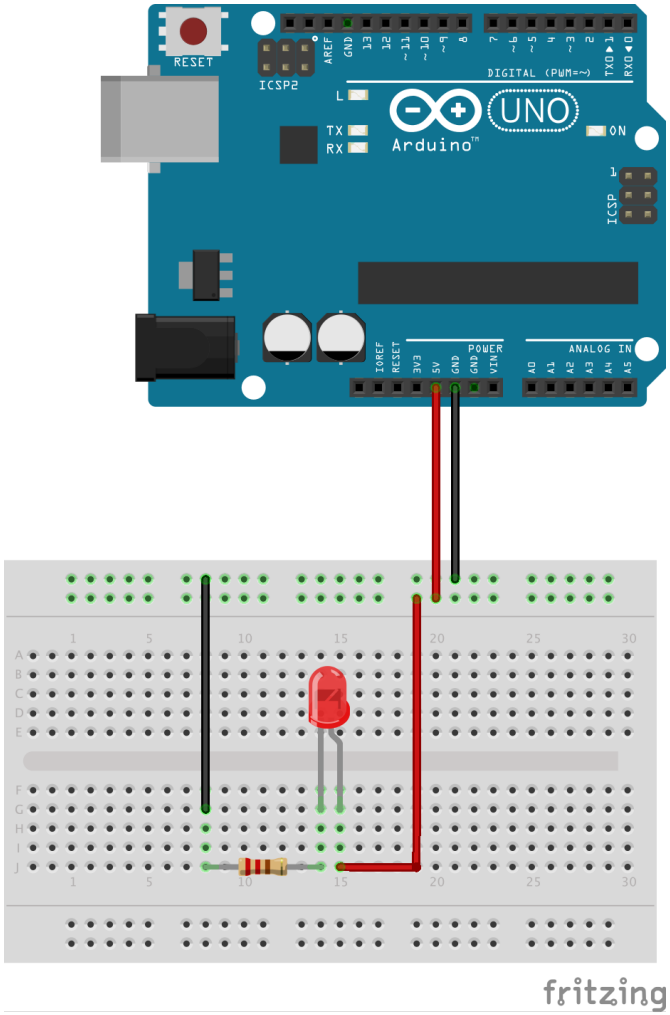




- Breadboards are used for fast prototyping.
- Allows to build an electronic circuit without soldering.
- Pins are connected as shown in the schematic on the right.

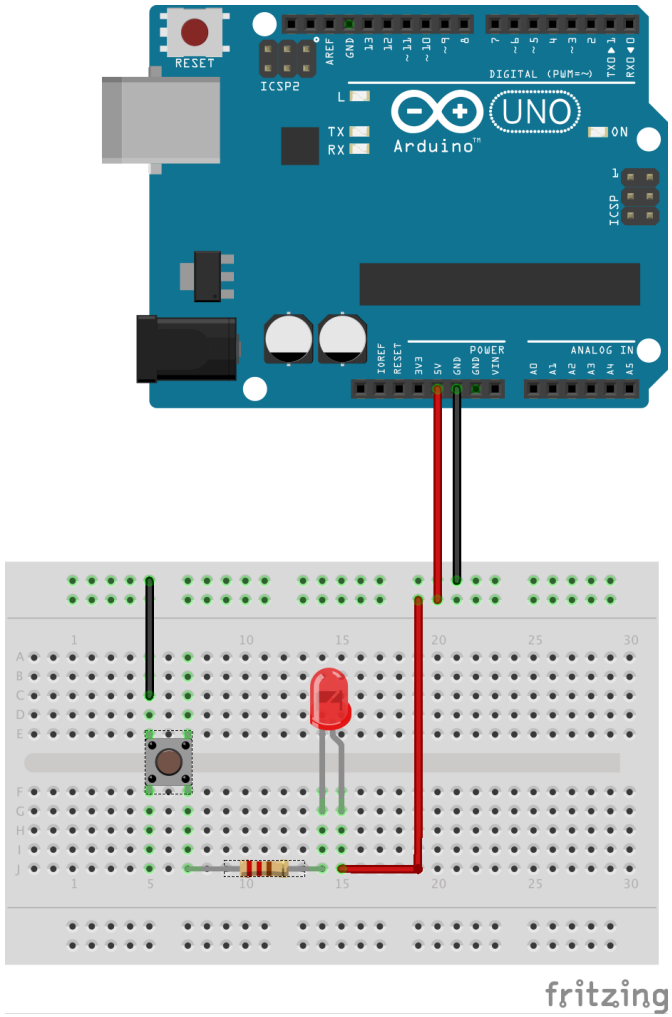
“Hello World” Electronics

- Simple electronic circuit that powers a LED.
- Arduino is only used to provide power to the circuit:
  - Ground: black wire.
  - 5V: red wire.
- 220Ω resistor is used to avoid burnout of the LED.
- Note: polarity of LED is important!



Push Button

- Simple electronic circuit using a LED and a push button.
- Arduino is only used to provide power to the circuit.
- LED will only turn on when the push button is pressed.

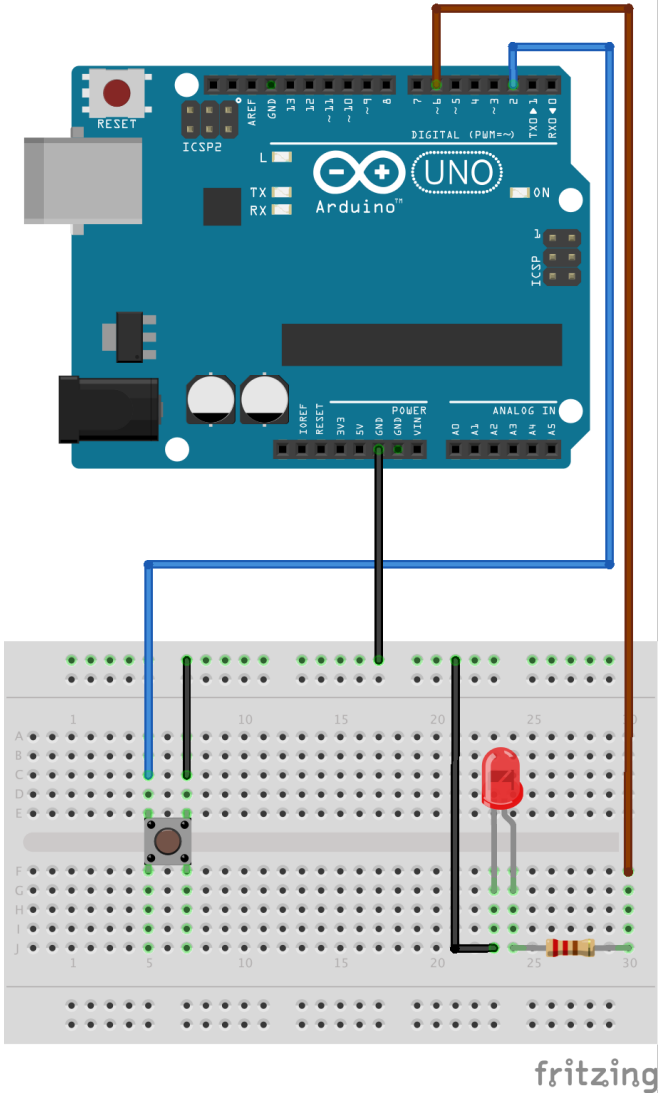


General Purpose Input/Output (GPIO)

- Generic pin with no pre-defined purpose.
- Can be configured either as input or output.
- Programmable interface:
  - Read state of a binary device (e.g., push button).
  - Control on/off state of a binary output device (e.g., LED)
- Arduino Uno:
  - 14 digital GPIO pins.

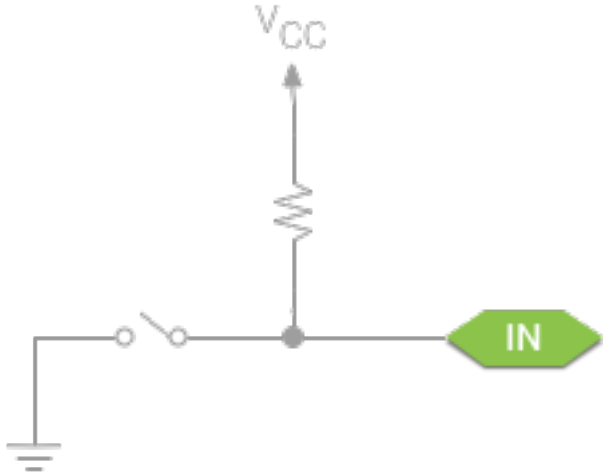
“Hello World” Arduino

- First electronic that makes use of the Arduino.
- Push button is connected to GPIO pin 2.
- LED is connected to GPIO pin 6.
- Both LED and the push button are also connected to ground to form a complete circuit with their respective GPIO pins.
- Pushing the button will not turn on the LED.
- Arduino needs to be programmed to implement desired behavior.



Pull-Up Resistor

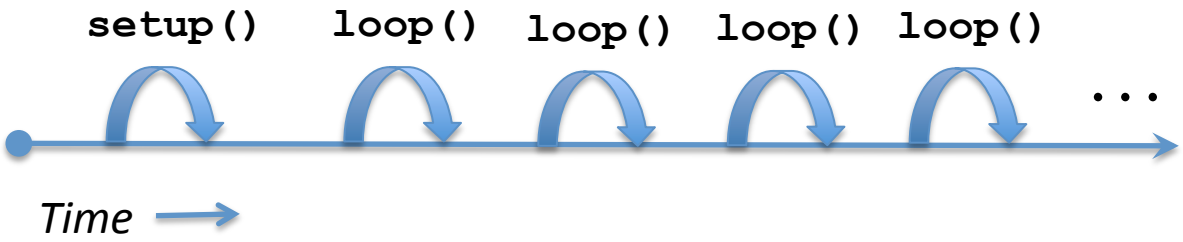
- Floating input:
  - Digital input not actively connected to any signal.
  - Are susceptible to electromagnetic interference.
  - Can cause unpredictable readings.
- Pull-up resistor:
  - Ensure that the line is driven to a stable value, even when nothing else is connected.
  - Guarantee that each signal has a stable default state that the rest of the system can rely on, without significantly affecting the input or output signal directly.
  - Pull-up resistor values are typically between 1kΩ and 10kΩ.
  - Arduino has built-in pull-up resistors on its GPIO pins.



Arduino Sketches

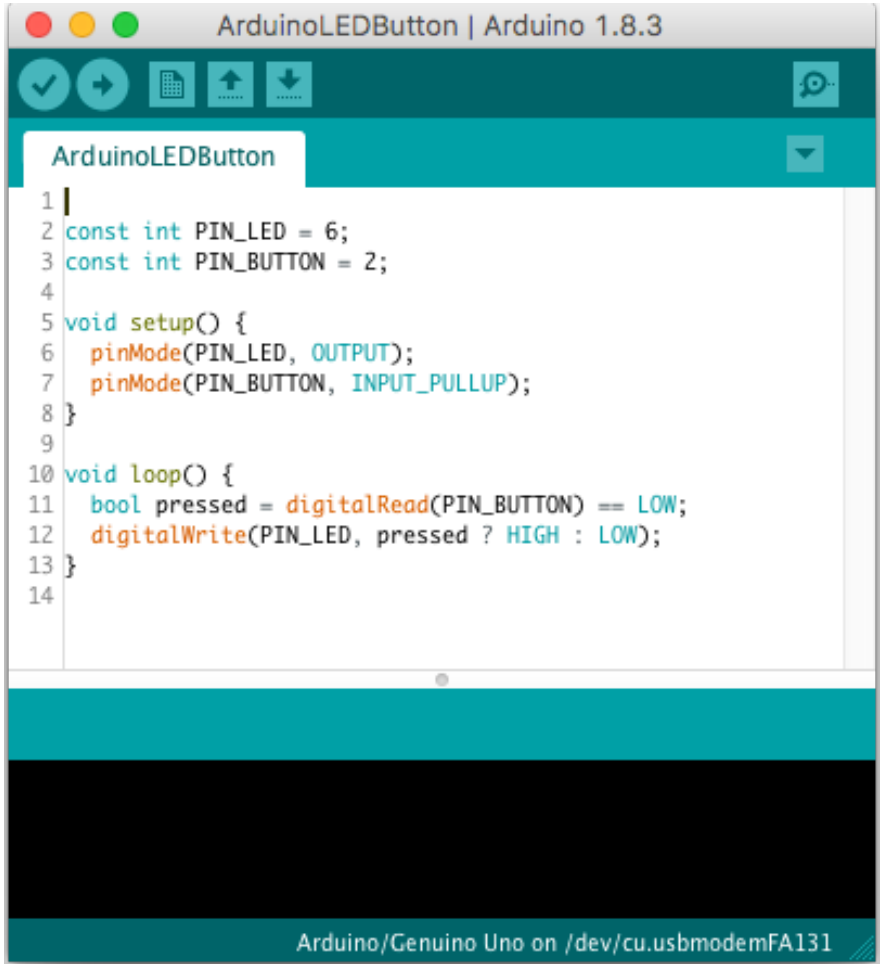
- Programs for the Arduino are called **Sketches**.
- Sketches can be written in C/C++.
- No operating system!
- Only one thread of execution.
- Support for common POSIX functions (e.g., **`malloc()`**, **`strlen()`**, etc)
- IDE automatically includes common header files.
- Popular IDEs:
  - Arduino IDE.
  - Atom with PlatformIO.

```
// Basic template for
// a sketch

void setup() {
  // ...
}

void loop() {
 // ...
}
```



Arduino IDE

- Download free Arduino IDE: https://www.arduino.cc/en/Main/Software
- Configuration:
  - *Tools > Board > Arduino/Genuino Uno*
  - *Tools > Programmer > AVR ISP*
  - *Tools > Port > /dev/XXX*
- Note: the Arduino needs to be connected to the laptop in order for it to show up under *Tools > Port*.



Sketch for Button/LED Circuit

```
const int LED_PIN = 6;
const int BUTTON_PIN = 2;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  bool pressed = digitalRead(BUTTON_PIN) == LOW;
  digitalWrite(LED_PIN, pressed ? HIGH : LOW);
}
```

- This sketch assumes the wiring as shown on the "Hello World" for Arduino slide.
- GPIO pins can be configured either as input (**`INPUT_PULLUP`**) or output (**`OUTPUT`**) via **`pinMode()`**
- **`digitalRead()`** can return either **`HIGH`** or **`LOW`**. For a pin configured as **`INPUT_PULLUP`** it will return **`LOW`** when the button is pressed. It will also activate an internal pull-up resistor on that pin.
- The value of **`HIGH`** or **`LOW`** can be set for an output pin via **`digitalWrite()`**
- This sketch will constantly update the LED with each call to **`loop()`**

Blinking LED

```
const int LED_PIN = 6;

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(500);
  digitalWrite(LED_PIN, LOW);
  delay(250);
}
```

- Function **`delay()`** will sleep for the number of milliseconds provided as a parameter.
- This sketch will repeatedly first turn the LED on for half a second (500ms) and then off for a quarter of a second (250ms).

Concurrent Jobs

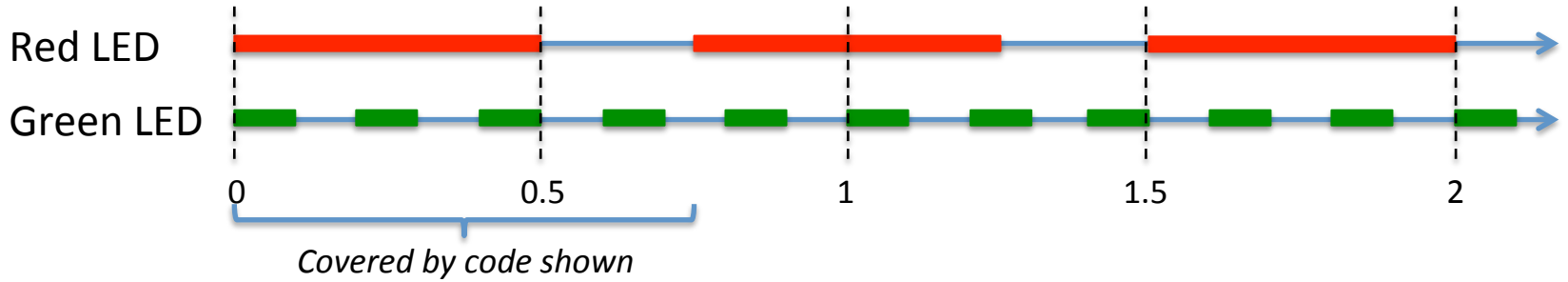
- Problem statement:
  - Two LEDs, one green and one red, should blink in different intervals.
  - Red LED should be on for 0.5s and off for 0.25s.
  - Green LED should blink in 0.1s interval.
- Ordinarily one would create two processes/threads that each control one LED.
- Since there is only one thread of execution, it is not easy to do two different jobs at the same time.

- Period of red LED is 750ms.
- Period of green LED is 200ms.
- Period of both LEDs is the least common multiple of 750 and 200:

$$LCM(200,750) = \frac{200 * 750}{GCD(200,750)} = \frac{200 * 750}{50} = 3000$$

  The pattern of the two blinking LEDs repeats itself after 3 seconds.
- One iteration of the `loop()` function takes 3 seconds and covers one complete on/off cycle for both LEDs.
- Spaghetti code!

```
void loop() {
  digitalWrite(LED_RED_PIN, HIGH);
  digitalWrite(LED_GREEN_PIN, HIGH);
  delay(100);
  digitalWrite(LED_GREEN_PIN, LOW);
  delay(100);
  digitalWrite(LED_GREEN_PIN, HIGH);
  delay(100);
  digitalWrite(LED_GREEN_PIN, LOW);
  delay(100);
  digitalWrite(LED_GREEN_PIN, HIGH);
  delay(100);
  digitalWrite(LED_GREEN_PIN, LOW);
  digitalWrite(LED_RED_PIN, LOW);
  delay(100);
  digitalWrite(LED_GREEN_PIN, HIGH);
  delay(100);
  digitalWrite(LED_GREEN_PIN, LOW);
  delay(50);
  digitalWrite(LED_GREEN_PIN, HIGH);
  // Continue like this for full
  // period (3 seconds)
}
```



- Code shown here makes one LED blink without the use of **`delay()`**
- Blink pattern:
  - On-time: 500ms
  - Off-time: 250ms
- Sketch makes use of function **`millis()`** that returns the number of milliseconds since the Arduino booted.
- **`millis()`** will wrap-around to 0 after 50 days.
- **`tsLastStateChange`** keeps track of the timestamp when the state of the LED was changed last.
- **`ledState`** keeps track of the current state of the LED (off or on)
- Since there is no **`delay()`**, one iteration of **`loop()`** will only take a fraction of a second.

```
const int LED_PIN = 6;
const int LED_ON_TIME = 500;
const int LED_OFF_TIME = 250;

int ledState = LOW;
unsigned long tsLastStateChange = 0;

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  unsigned long delta = millis() -
                        tsLastStateChange;
  switch(ledState) {
  case LOW:
    if (delta > LED_OFF_TIME) {
      ledState = HIGH;
      tsLastStateChange = millis();
    }
    break;
  case HIGH:
    if (delta > LED_ON_TIME) {
      ledState = LOW;
      tsLastStateChange = millis();
    }
    break;
  }
  digitalWrite(LED_PIN, ledState);
}
```

- Class **BlinkingLED** is an object-oriented version to blink a LED for a given on/ off pattern.
- Uses the same technique as outlined on the previous slide.
- Constructor takes the pin number as well as the desired on- and off-time pattern.
- Pin is configured as **OUTPUT** in the constructor.
- Method **update()** will turn the LED off or on using **millis()**
- Method **update()** is essentially the implementation of **loop()** on the previous slide.
- Global variables such as **LED_PIN** on the previous slide have been turned to private members of class **BlinkingLED**.

```
class BlinkingLED {
  const int LED_PIN;
  const int LED_ON_TIME;
  const int LED_OFF_TIME;
  int ledState = LOW;
  unsigned long tsLastStateChange = 0;
public:
  BlinkingLED(int pin, int onTime, int offTime) :
    LED_PIN(pin), LED_ON_TIME(onTime),
    LED_OFF_TIME(offTime) { pinMode(LED_PIN, OUTPUT);}
  void update() {
    unsigned long delta = millis() - tsLastStateChange;
    switch(ledState) {
    case LOW:
      if (delta > LED_OFF_TIME) {
        ledState = HIGH;
        tsLastStateChange = millis();
      }
      break;
    case HIGH:
      if (delta > LED_ON_TIME) {
        ledState = LOW;
        tsLastStateChange = millis();
      }
      break;
    }
    digitalWrite(LED_PIN, ledState);
  }
};
```

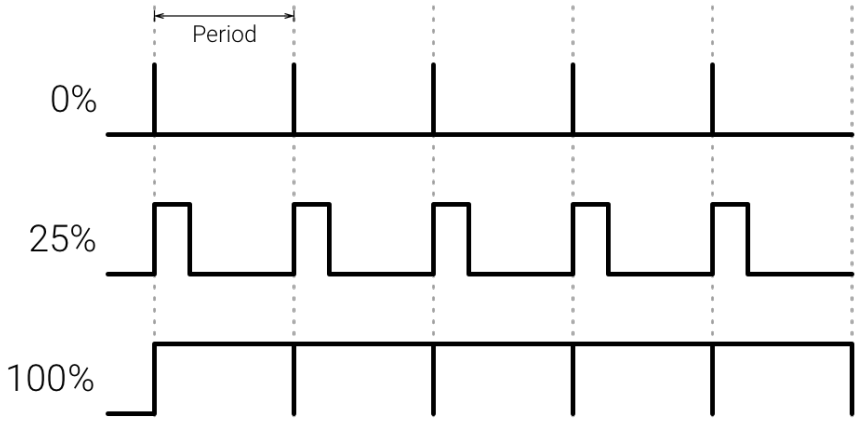
Two Blinking LEDs

```
BlinkingLED ledRed(6, 500, 250);
BlinkingLED ledGreen(7, 100, 100);

void setup() {}

void loop() {
  ledRed.update();
  ledGreen.update();
}
```

- With the help of C++ class **`BlinkingLED`**, two instances **`ledRed`** and **`ledGreen`** can be declared, each with individual on/off patterns.
- **`loop()`** function just calls the respective **`update()`** methods.
- This is a common programming pattern for environments with only one execution thread: each sub-module contributes an **`update()`** method.

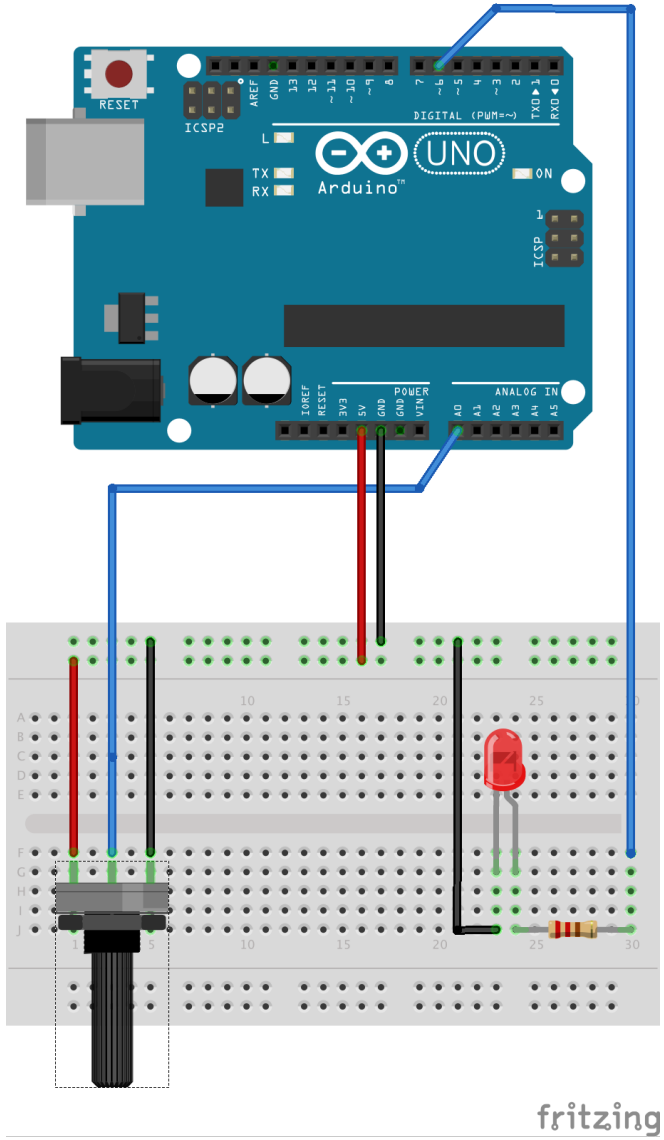
Pulse Width Modulation (PWM)

- Technique to achieve analog results with digital means.
- On/off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off.
- E.g., by asserting an output pin to HIGH for 25% of the time at high frequency, the brightness of a connected LED is reduced proportionally. Typical period time is 2ms (500Hz).
- Period that the signal is high is also referred to as *duty cycle*.
- GPIO pins that support PWM are prefixed with "~".



Analog Data

```
const int LED_PIN = 6;
const int POTENTIOMETER_PIN = A0;

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // val will be between 0 and 1023
  int val = analogRead(POTENTIOMETER_PIN);
  // (val / 4) will be between 0 and 255
  analogWrite(LED_PIN, val / 4);
}
```



- GPIO pins marked with "~" support PWM. For those pins **`analogWrite()`** can be used.
- Special pins can be used for analog input. Voltage levels from 0V to 5V are mapped to numbers 0 to 1023.
- **`analogRead()`** can only be used for analog pins prefixed with "A" (not GPIO pins)
- Potentiometer used in the circuit is a variable resistor to change the input voltage on the analog pin A0.

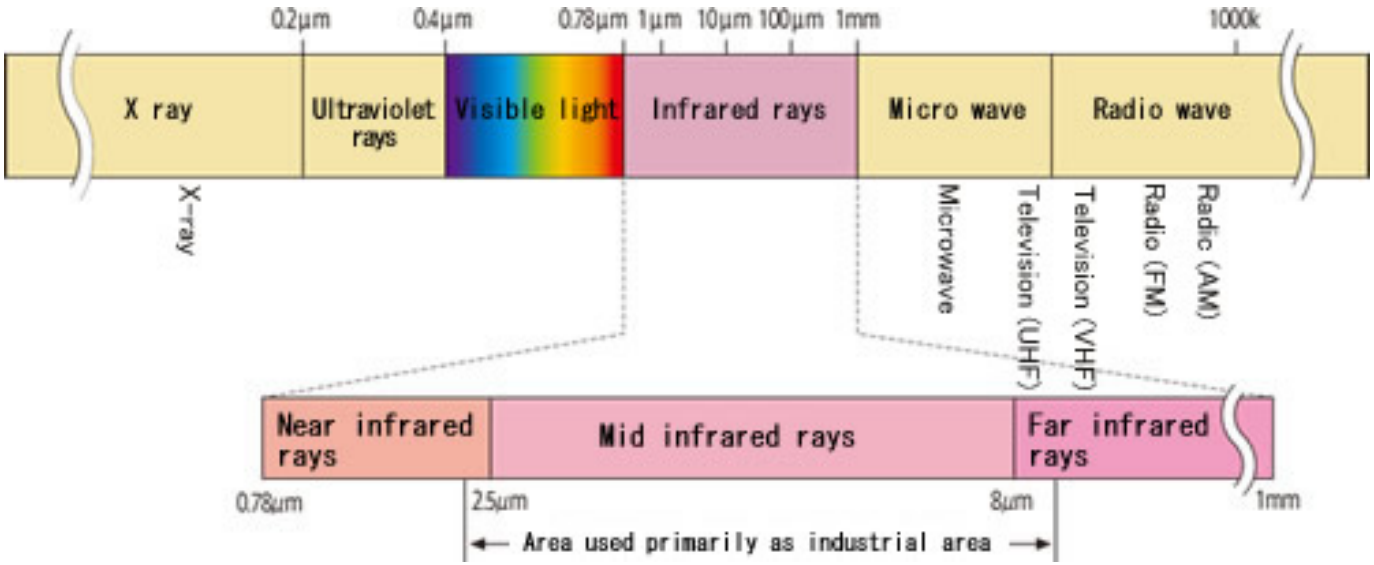
Button/LED Sketch using Interrupts

```
const int LED_PIN = 6;
const int BUTTON_PIN = 2;

void isr_button() {
  bool pressed = digitalRead(BUTTON_PIN) == LOW;
  digitalWrite(LED_PIN, pressed ? HIGH : LOW);
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  int irq = digitalPinToInterrupt(BUTTON_PIN);
  attachInterrupt(irq, isr_button, CHANGE);
}

void loop() { /* Do nothing */ }
```

- Pins 2 and 3 on the Arduino Uno can generate interrupts.
- This sketch attaches an ISR specified by function pointer **`isr_button`** to be called whenever there is a state change on **`BUTTON_PIN`**.
- Notes:
  - ISRs should execute very fast. Do not use **`delay()`** while inside the ISR.
  - Interrupts are turned off within an ISR.
  - Return value of **`millis()`** will not change while inside the ISR.

Infrared Radiation

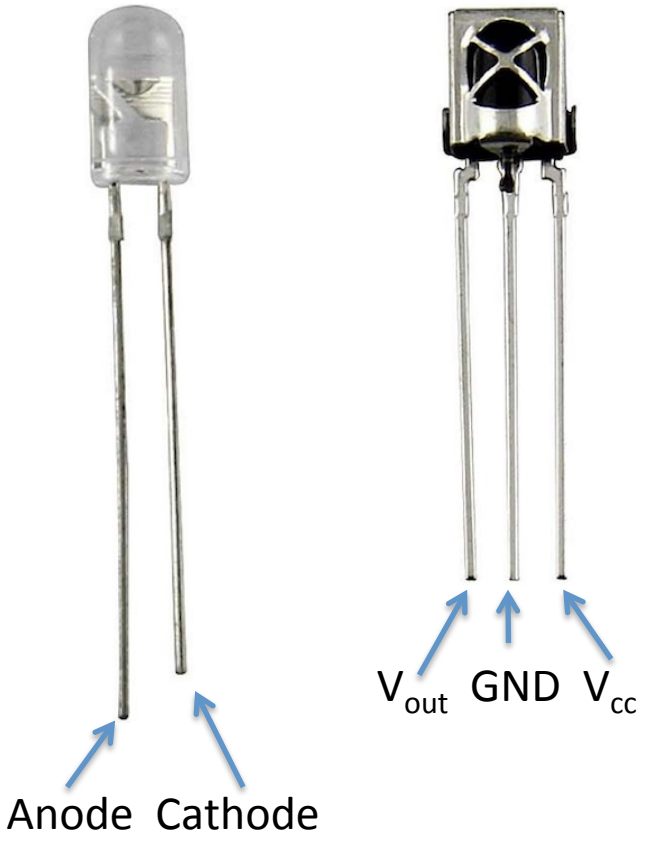
- Electromagnetic radiation (EMR) with longer wavelengths than those of visible light.
- Invisible to the human eye.
- Applications: night vision, remote controls, motion detection, …



http://www.apiste-global.com/fsv/technology_fsv/detail/id=1188

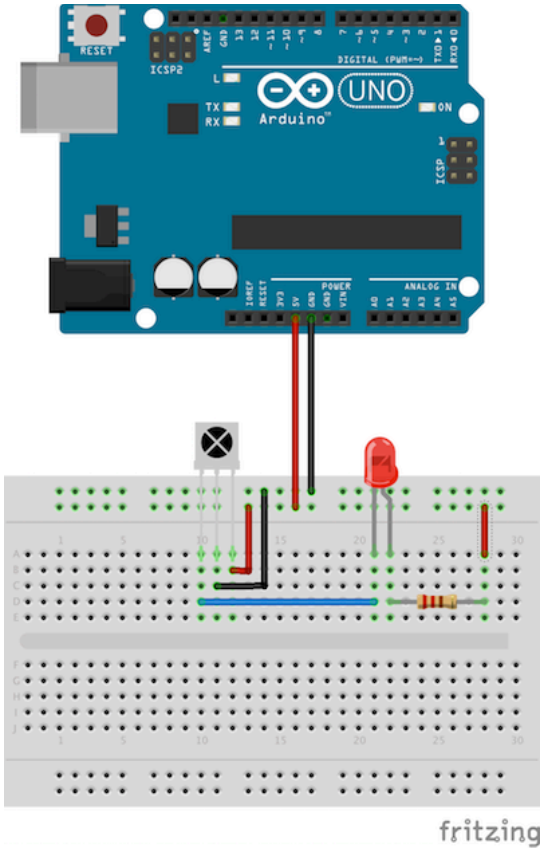
IR Diode & Receiver

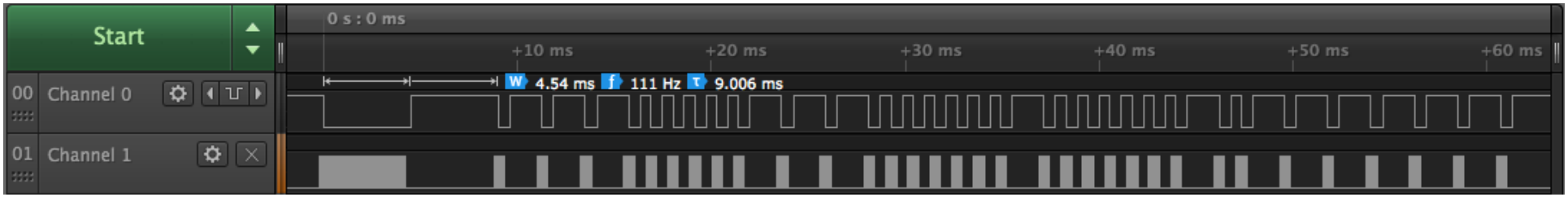
- IR Diode:
  - Emits infrared radiation.
  - Polarity is important.
  - Needs current-limiting resistor.
- IR Receiver:
  - V_{cc}: 3-5V
  - V_{out} pin is asserted to low when IR signal is detected.
  - IR signal needs to be modulated at 38 kHz.

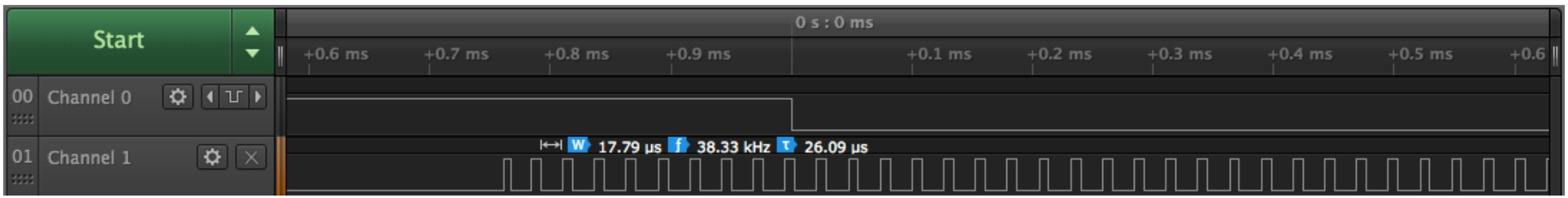


IR Signals

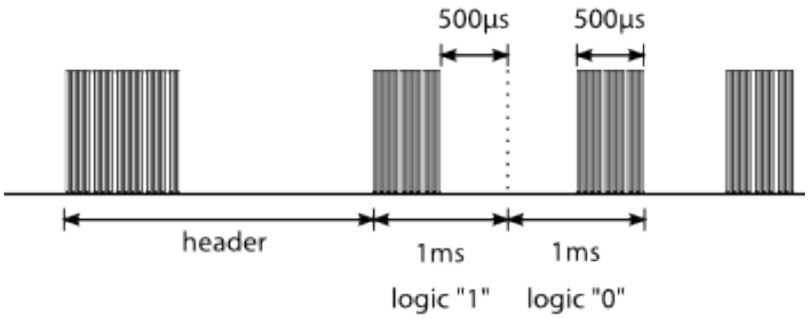
- Pointing a TV remote to the IR receiver and pushing a button will light up the LED.
- Each button of a TV remote generates a unique high/low pattern.
- Channel 0 shows the high/low pattern of the OFF button of a Samsung TV remote.
- Channel 1 shows the modulated version of the signal.
- Bottom screenshot is a zoomed-in version of the top screenshot: the first 4.54ms shown in channel 0 (top) is a sequence of 17.79 µs high/low patterns shown in channel 1 (bottom)



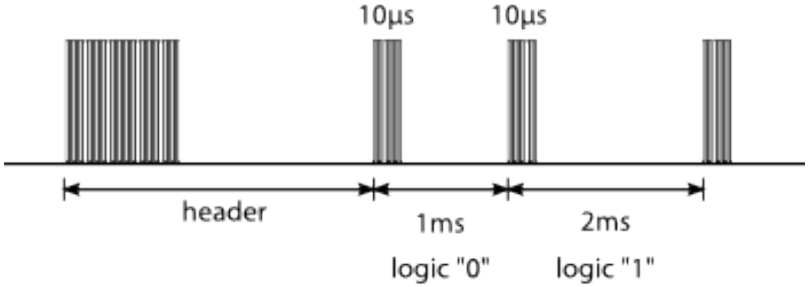




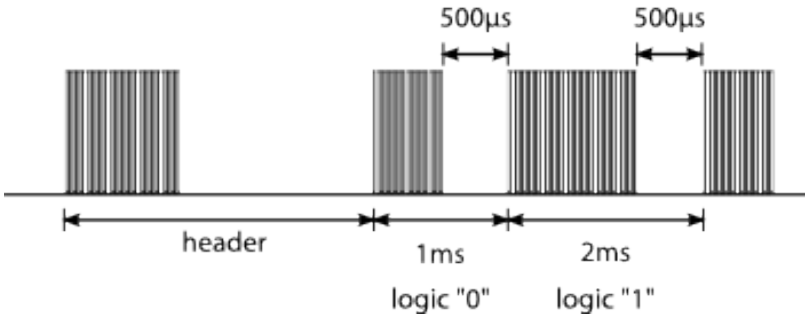
Manchester encoding



Pulse distance coding



Pulse length coding

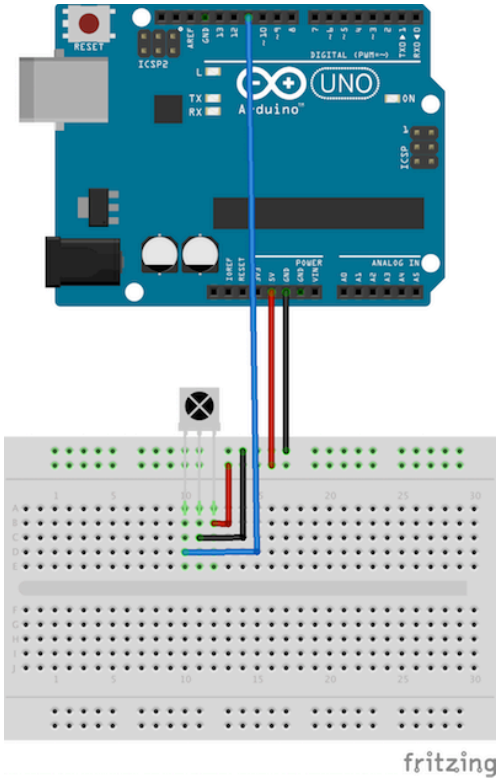


- Samsung TV remote uses Pulse Distance Coding.
- OFF button on the previous slide encodes the binary value: 11100000111000000100000010111111
- Hex: 0xE0E040BF

http://irq5.io/2012/07/27/infrared-remote-control-protocols-part-1/
http://www.vishay.com/docs/80071/dataform.pdf

Receiving an IR Signal

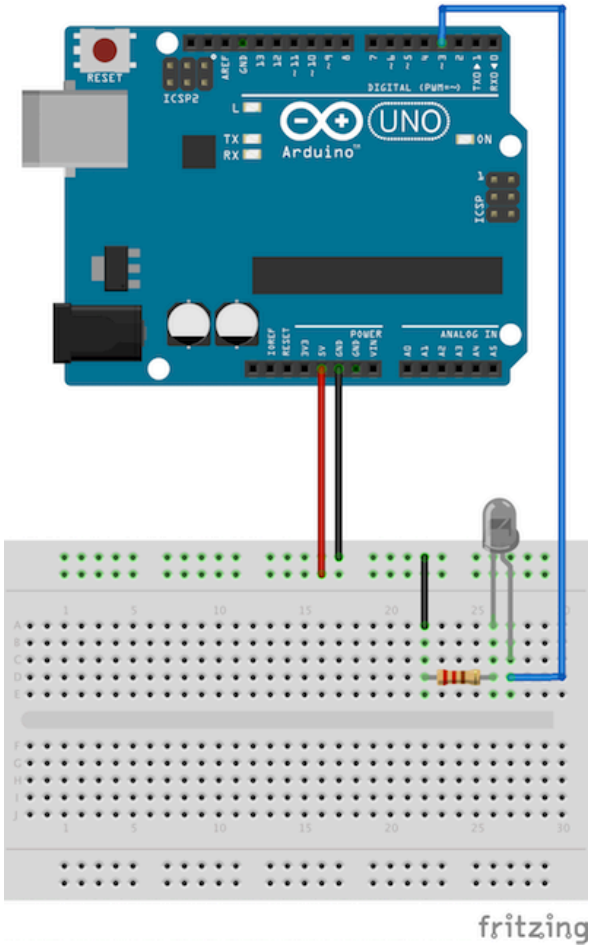
- Library IRremote allows to capture the high/low pattern of an IR sender (e.g., TV remote)
- Output pin of the IR receiver is connected to pin 11 of the Arduino.
- Check companion web page for the source code of the sketch.
- The sketch will print out the high/low pattern as an array of numbers to the Serial Monitor.
- Each number is in micro-seconds.
- Array below reflects the pattern from the previous slide.



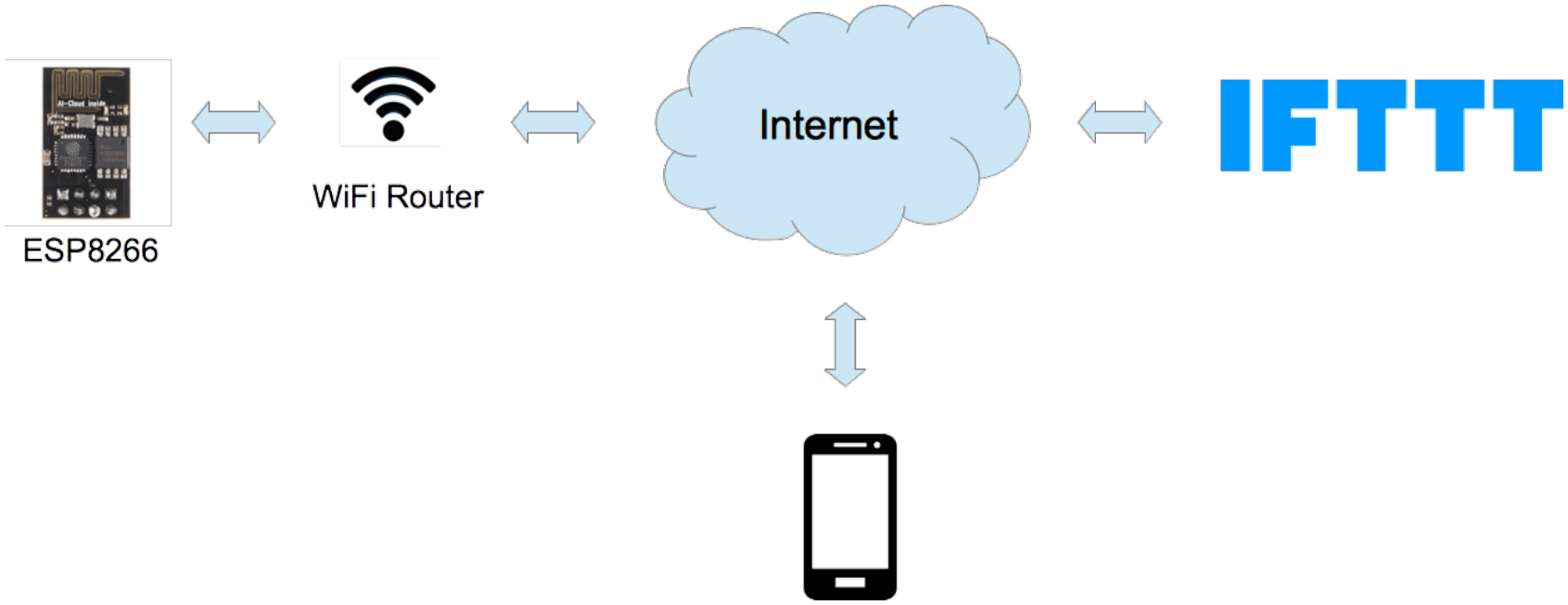

```
unsigned int rawData[] = {4500, 4450, 600, 1600, 600, … , 1650, 600, 1600, 600};
```

Sending an IR Signal

- With the help of the IRremote library and an IR diode, an IR signal can be sent.
- Anode of the IR diode is connected to pin 3 of the Arduino.
- Note polarity of the IR diode as well as current-limiting resistor.
- Check companion web page for source code of the sketch.
- The sample sketch will send the sample pattern as used on the previous slide.
- Pointing the IR diode to a Samsung TV will turn on the TV!



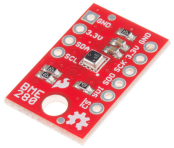
Cloud Demo



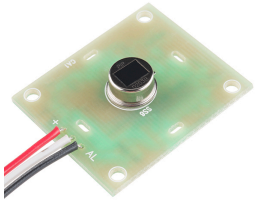
- The companion web page shows how to connect an ESP8266 with the cloud.
- ESP8266 connects via WiFi to the Internet.
- ESP8266 performs an HTTP-POST request to IFTTT.com.
- HTTP-POST request will trigger a push notification to a mobile phone.
- Follow instructions on the companion web page.

Sensors

Humidity Sensor
HIH-4030

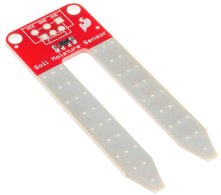


Atmospheric Sensor
BME280

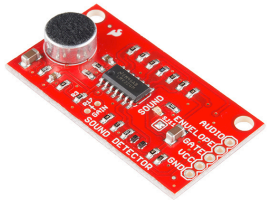


Passive Infrared
Motion Sensor

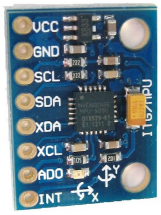
Ultrasonic Sensor
HC-SR04



Soil Moisture



Sound Detector



Accelerometer, Gyroscope
Temperature – MPU6050

Photocell

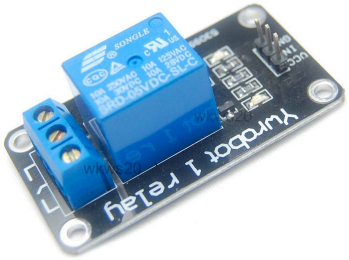
Reed Switch
TMP36

Actuators/Displays

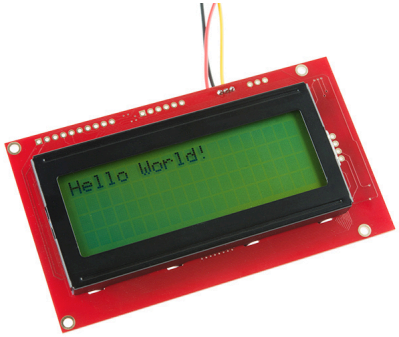
Gear motor
with Wheel Set

Solenoid

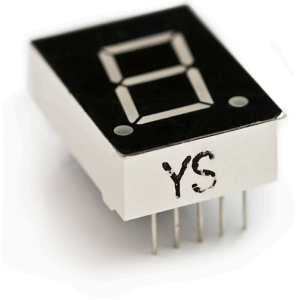
Water Valve



Relay



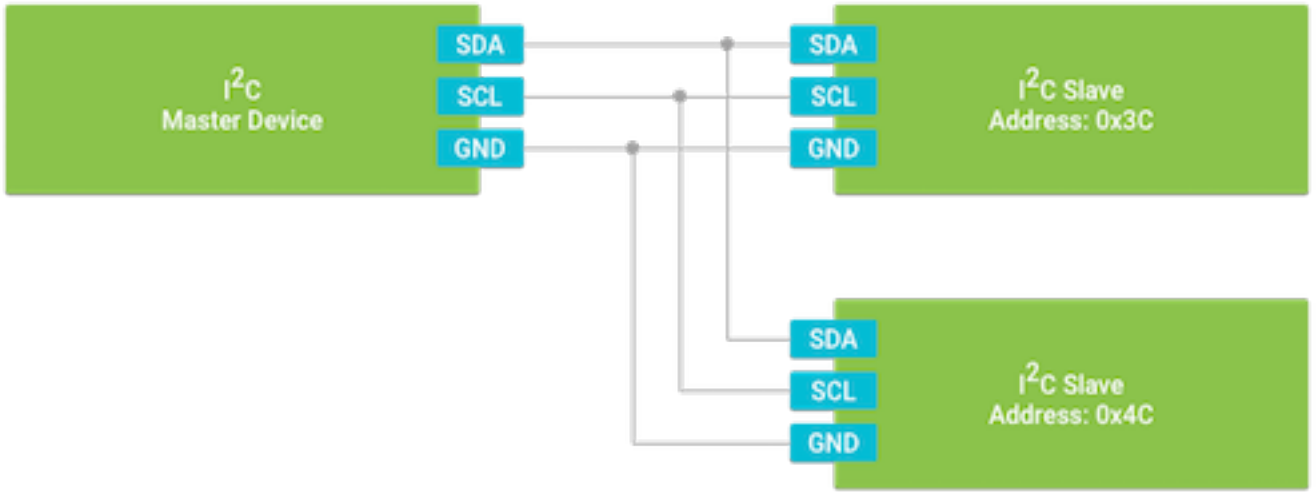
Liquid Crystal Display (LCD)



7 Segment Display

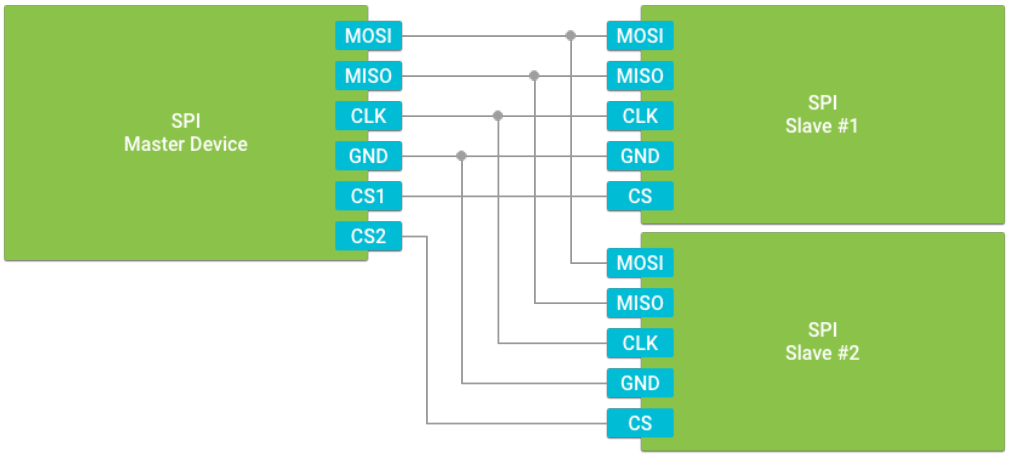
Inter-Integrated Circuit (I2C)

- Master/slave, packet switched, half-duplex, serial bus.
- 3-wire interface:
  - Shared Clock Signal (SCL)
  - Shared Data Line (SDA)
  - Ground (GND)
- Master synchronizes slaves by triggering SCL.
- Every slave has a unique address.
- Master: node that generates the clock and initiates communication with slaves
- Slave: node that receives the clock and responds when addressed by the master.
- Transmission speed 100 kb/s.



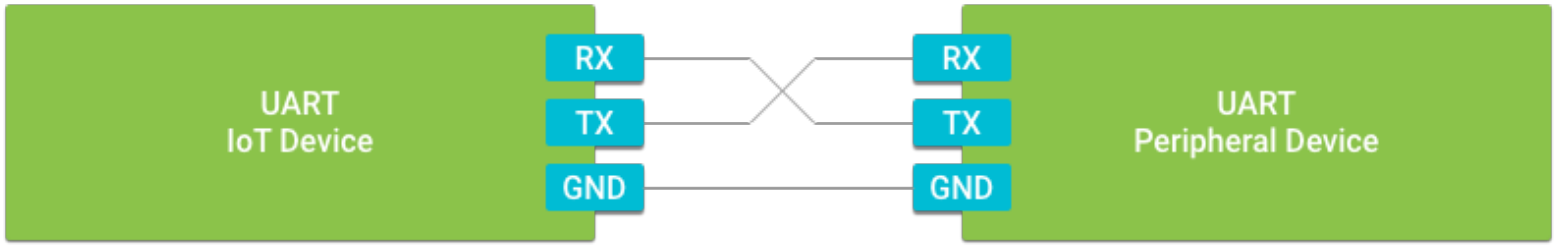
Serial Peripheral Interface (SPI)

- Master/Slave, high-bandwidth, full-duplex, serial connection.
- 4-wire interface:
  - Serial Clock (CLK): Output from master.
  - Master Out Slave In (MOSI)
  - Master In Slave Out (MISO)
  - Ground (GND)
- Slaves are selected via a dedicated Chip Select (CS) wire.
- Clock speed: 16MHz – 25MHz



Universal Asynchronous Receiver Transmitter (UART)

- Point-to-point, full-duplex connection between two devices.
- Universal: both data transfer speed and data byte format are configurable.
- Asynchronous: no clock signal present to synchronize data transfer.
- 3-wire interface:
  - Receive (RX)
  - Transmit (TX)
  - Ground (GND)
- Two additional wires for hardware flow control:
  - Request To Send (RTS)
  - Clear To Send (CTS)
- Faster than I2C.



Comparison Peripheral I/O

Protocol	Transfer Type	# of Wires	# of Peripherals	Transfer Speed
I2C	Synchronous	2	Up to 127	Low
SPI	Synchronous	4+	Unlimited	High
UART	Asynchronous	2 or 4	1	Medium

Security

- IoT's greatest challenge: SECURITY!!!
- Problem: IoT devices are full-fledged computers that rarely get updated.
- Popular Foscam web cam:
  - "The sheer number of vulnerabilities offers an attacker multiple alternatives in compromising the device."
  - "The researchers went on to say that they notified Foscam representatives of the vulnerabilities several months ago and that, to date, the manufacturer hasn't fixed any of them."
- Compromised IoT devices are turned into a Botnet and used for DDoS attacks (Distributed Denial of Service).
- Vulnerable is anything that is connected to the Internet: fridges, lights, cars, medical devices, …

https://arstechnica.com/information-technology/2017/06/internet-cameras-expose-private-video-feeds-and-remote-controls/
https://en.wikipedia.org/wiki/Mirai_(malware)