PHYSICAL COMPUTING FOR INTERACTIVE MUSIC

MICHAEL GUREVICH,
UNIVERSITY OF MICHIGAN
SCHOOL OF MUSIC, THEATRE & DANCE

FOOD FOR THOUGHT

https://www.youtube.com/watch?v=h4V8uklXCjg

https://www.youtube.com/watch?v=U1L-mVGqug4

https://vimeo.com/43025087

QUESTIONS

- **What was similar or different about the performances?**
- **What dimensions, aspects, or qualities can we use to assess or critique the performances?**

SOME OF MY CONSIDERATIONS

Scenario

Identity & Metaphor

Intentionality & Purpose

Skill

Temporal dynamics (see Paine: Control vs Create)

Scale

Style

SCENARIO

Who?

How many?

Where?

Much of the context for traditional music performances is given. With new interactive systems, how do we create context?

IDENTITY & METAPHOR

What IS it? How does a spectator make sense of it?

Is it a toy? A game? A puzzle? A musical instrument? A drum machine?

INTENTIONALITY / PURPOSE

Purpose → Identity: What is it FOR?

Intentionality: What is the performer trying to accomplish?

Sonification of incidental or external processes?

Is the system the primary locus of attention?

SKILL

- Who can use it?
- Floors + Ceilings
  - Threshold/entry fee: what prior knowledge, experience or ability is required? How much practice?
  - Ceiling: Ability to develop virtuosity? Will an expert be distinguishable from a novice?
- Is the dream of “meaningful control” or “meaningful contribution” from a complete beginner attainable? Should it be?

TEMPORAL DYNAMICS

Control ←→ Create

Programming ←→ Performing

(Other models: turn-taking, conversation, physical modeling)

SCALE

Degrees of freedom

Dynamic range of input

Frequency range of input

Dynamic range of output

Spectral/timbral/textural range of output

STYLE

Beyond ceilings and floors — Walls —Range of activities supported

Jorda: Macrodiversity…Mid-diversity…Microdiversity

Macrodiversity: How musically specialized or generic? Can you play solo free jazz as well as symphonic music?

Mid-diversity: How different will two different compositions sound? Does it sound like you're "always playing the same piece?"

Microdiversity: How different will two different performances of the SAME composition sound?

EXAMPLES

Michel Waisvisz

https://youtu.be/pYfRORkuPX8

http://youtu.be/U1L-mVGqug4

http://youtu.be/AedrvTXBgEA

Nicolas Collins

http://youtu.be/xOBnYaqvaec

Leafcutter John

http://youtu.be/2jIILHfSEfs

T Stick

http://youtu.be/BudSGA511pg

http://youtu.be/l4u3XghyDSo

Les Gestes, Prosthetic Instruments

https://vimeo.com/69552445

Laetitia Sonami's Lady's Glove

http://vimeo.com/11316136

http://youtu.be/C8GqbS2w_Lg

Laetitia Sonami's other thing

https://youtu.be/odR-kvwkvqc

http://youtu.be/48B7tvq4mg4

Imogen Heap

http://youtu.be/ci-yB6EgVW4

https://youtu.be/h4V8uklXCjg

Mogees

https://youtu.be/30xWm0fyqsc

http://youtu.be/-dPKHLiM9AM

Reactable

http://vimeo.com/26678704

EXAMPLES

Double Slide Controller

https://youtu.be/0sX4Dleqz_0

Eigenharp

https://youtu.be/CBzVTmaGOI4

Linnstrument

https://youtu.be/px0ld-fVD9M

https://youtu.be/YMfXA8pJd0Q

ROLI Seabord

https://youtu.be/8n-bEy9lSpM

NOMIS

https://youtu.be/lQ-gHQT2l5I

Myriam Bleau, Soft Revolvers

https://vimeo.com/104996493

Nicolas D'Alessandro, Handsketch

https://youtu.be/EK1Q7X_c3Q8

2016 Guthman Musical Instrument Competition Final Concert

https://youtu.be/5EUHqy_QmHk

http://www.guthman.gatech.edu/winners

Eigenharp

https://youtu.be/U6Rzp1FQPao

http://www.eigenlabs.com/product/

Tenori On

https://youtu.be/M20ukzn_rsw

https://youtu.be/1Z3PT2fUiKg

EXAMPLES

Guthman Musical Instrument Competition

https://guthman.gatech.edu/

ARDUINO

MICROCONTROLLER

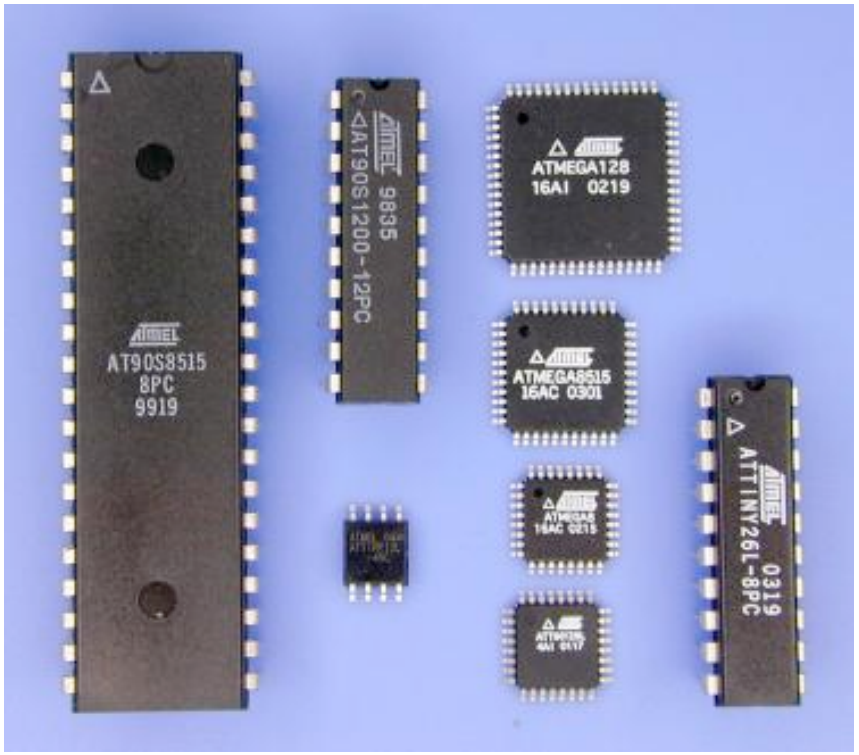
"Computer" on a chip

Different shapes and sizes

Different capabilities

Different architectures

Found in?



MICROCONTROLLER = COMPUTER

CPU

- Instruction set

Clock

- Execute "instructions" at a regular fixed rate

Input

Output

Memory

Storage

MICROCONTROLLER ≠ COMPUTER

	Microcontroller	Computer
Form	Single Integrated Circuit (IC)	Many components connected together
Clock	~16 MHz	~3 GHz
Storage	Flash memory: 32kB	Hard disk: 2 TB
Memory	RAM: 2 kB	RAM: 8GB
Bus	8 bit	64 bit
Math	Fixed point	Floating point
Program flow	Single program, runs continuously	Operating System; Multitasking
Peripherals	Low-level: Sensors, switches, LEDs, motors, LCD	High-level: Mouse, printer, camera, sound card

But...

ATARI 800 (1979)

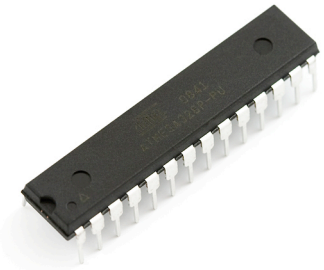
1.8 MHz

8 kB RAM

90 kB Storage (external floppy)

$999.95

ATMEGA 328 (2009)



20 MHz

2 kB RAM

32 kB storage (internal Flash)

$4.00

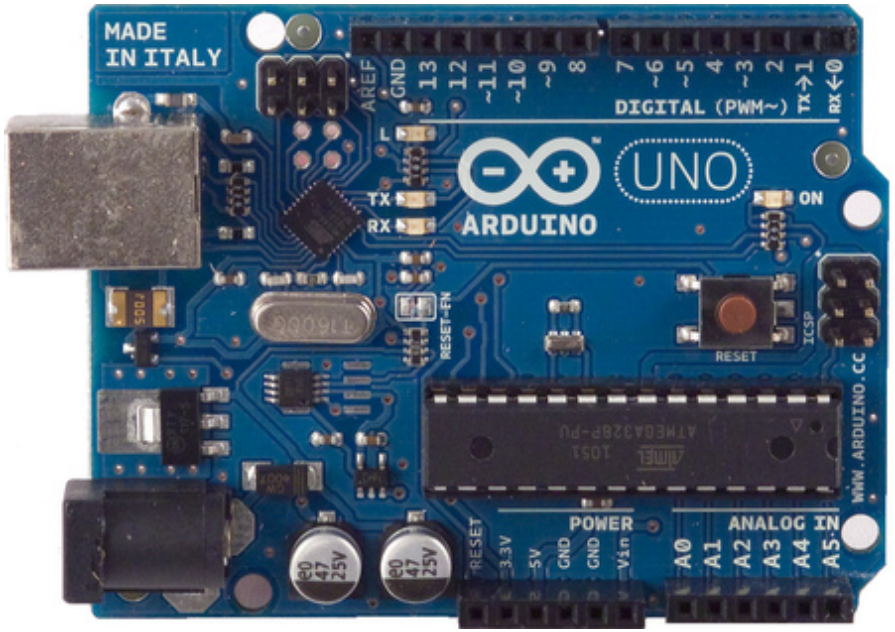
ARDUINO

Microcontroller development kit

- convenient hardware access to ATMega328 I/O and programming

"Open hardware" project

- Schematics are developed by a community
- Anyone can build (and sell) one



PROGRAMMING AN ARDUINO

Write code in C on your computer

Efficient compiler

- Designed to translate C-language code into AVR instructions
- Open source

Compiler generates AVR instructions

- Actually a map of the AVR's Flash program memory
- .hex file

Copy .hex file into the AVR's Flash program memory

- We do this via a USB cable

After this, the AVR runs on its own, no need to connect it to a computer

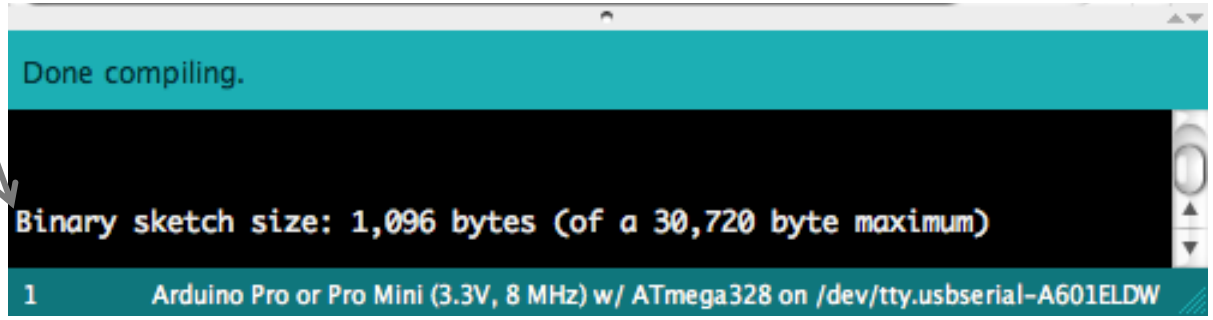
ARDUINO PROGRAMMING ENVIRONMENT

PROGRAMMING ENVIRONMENT

"Verify"
* Compiles Code

"Upload"
* Sends compiled code to Arduino board

How you know your code was compiled successfully. If not, errors are displayed here.



ARDUINO PROGRAM STRUCTURE

```
void setup() {
  // code executed once, at beginning
}


void loop() {
  // code executed continuously, as fast as
possible
}
```

ELECTRICAL REPRESENTATION OF DIGITAL LOGIC

The principle of digital systems is to represent bits as discrete voltage levels.

The "logic level" of the system dictates what the high voltage is (we use 5V).

The low level is ground (0V).

With the microcontroller, we can interact with the outside world in 2 very basic ways:

- Read the voltage of an input
- Set the voltage of an output

We have 2 voltage possibilities: +5V and 0V

DIGITAL I/O

Arduino has 14 pins for doing digital I/O

Pins can be configured as input or output with the `pinmode` function

```
pinMode(pin,MODE);
      // Set pin (integer) to INPUT or OUTPUT
```

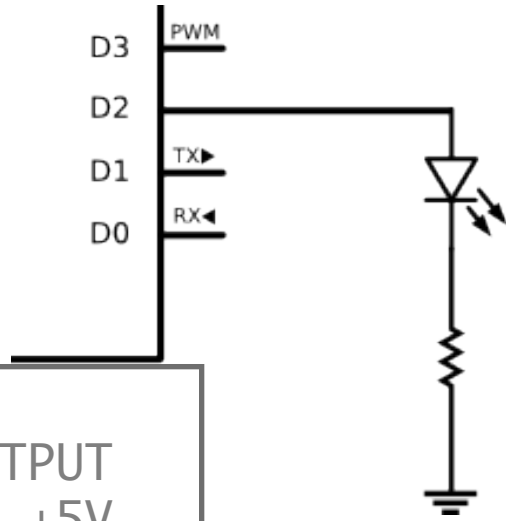
Initially, we may want to use input pins to read the state of switches or buttons

We may want to use output pins to turn on or off LEDs

WRITING AN OUTPUT PIN

When we set an Output pin HIGH, we can think of it as a 5V battery

- It can source a small amount of current (40 mA)
- We can use this to light up an LED:

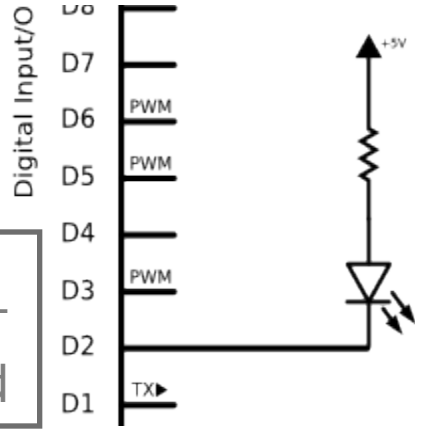



```
int ledPin = 2;
pinMode(ledPin,OUTPUT); // Set pin 2 to OUTPUT
digitalWrite(ledPin,HIGH); // Set pin 2 to +5V
```

WRITING AN OUTPUT PIN

When we set an Output pin LOW, we can think of it as ground

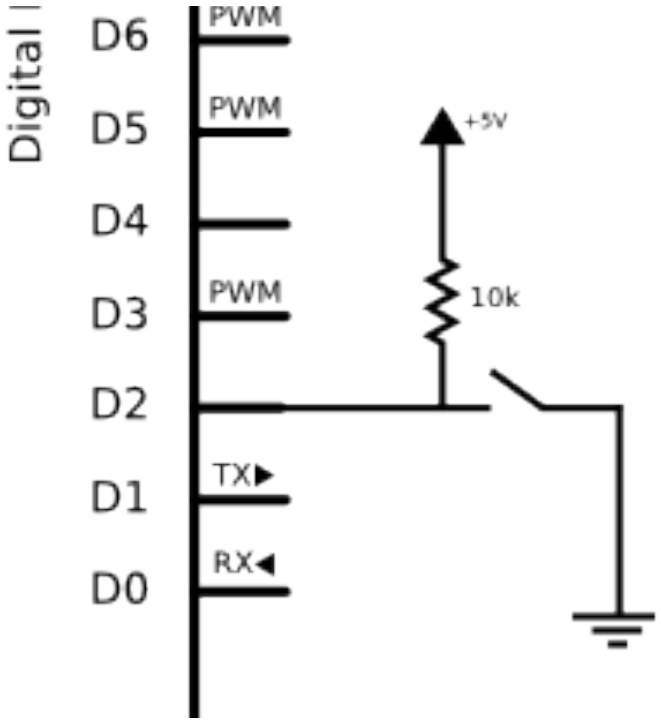
- It can sink a small amount of current (40 mA)
- We can also use this to light up an LED, with a different circuit:




```
int ledPin = 2;
pinMode(ledPin,OUTPUT);   // Set pin 2 to OUTPUT
digitalWrite(ledPin,LOW); // Set pin 2 to ground
```

READING AN INPUT PIN

```
int digitalRead(pin)
        // returns the value applied to an input pin
        // either HIGH or LOW
```



PUTTING IT TOGETHER

```
const int buttonPin = 2;      // the number of the button pin
const int ledPin =  13;       // the number of the LED pin
int buttonState = 0;          // variable for storing button state

void setup() {
   pinMode(ledPin, OUTPUT);   // initialize LED pin as an output
   pinMode(buttonPin, INPUT); // initialize button pin as an input
   digitalWrite(ledPin, LOW); // begin with LED OFF
}

void loop(){
   buttonState = digitalRead(buttonPin); // read state of the button

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

DIGITAL SYSTEMS

Digital here refers to discrete systems

- Can represent a fixed set of values

Most basic quantity is a bit

- Bit: Binary Digit

Binary numbers are simply base-2 numbers

- We normally use base-10 or decimal numbers
  - digits can represent 10 different values (0-9)
  - Where n is the base, there are n different digits from 0 to (n-1)
  - base 10 numbers are constructed as follows:

$$5*10^3 + 7*10^2 + 2*10^1 + 9*10^0$$

$$5729$$

BITS

Binary numbers use base 2

$$1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = ??$$

1011

Q: What is the maximum base-2 number we can represent with 4 digits?

Q: What is the maximum base-10 number we can represent with 4 digits?

Q: What is a general formula for the maximum number we can represent with n digits?

BITS & BYTES

With n digits, the maximum base-x number you can represent is:

$$x^n - 1$$

We call an 8-digit binary number a byte

We can represent $2^8 = 256$ different values from a range of 0000 0000 to 1111 1111

- 0 to $2^8 - 1$
- 0 to 255

BYTES IN A MICROCONTROLLER

Microcontroller represents data in bytes

- Memory is a series of 8-bit wide containers called registers
- Think of a shelf with 8 slots

When a bit is a 1 (there is a book in the slot on the shelf), we say it is set or high

When a bit is a 0 (the slot is empty), we say it is cleared or low

Digital logic:

- 1 or 0
- high or low
- set or cleared
- on or off

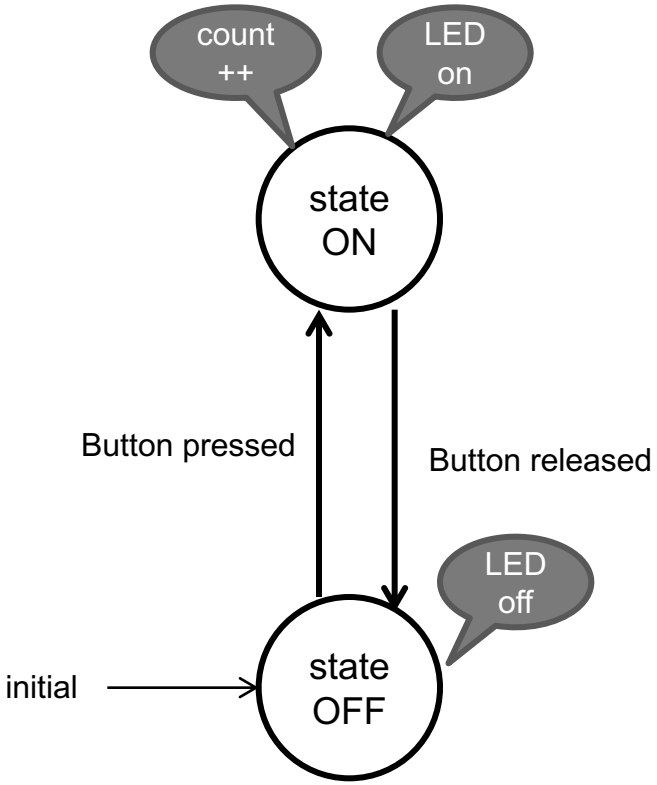
“Multitasking” Approach to Arduino

Feijs, L. 2013. “Multi-tasking and Arduino: Why and How?” in *Proceedings of the Conference on Design and Semantics of Form and Movement*, pp. 119-127.

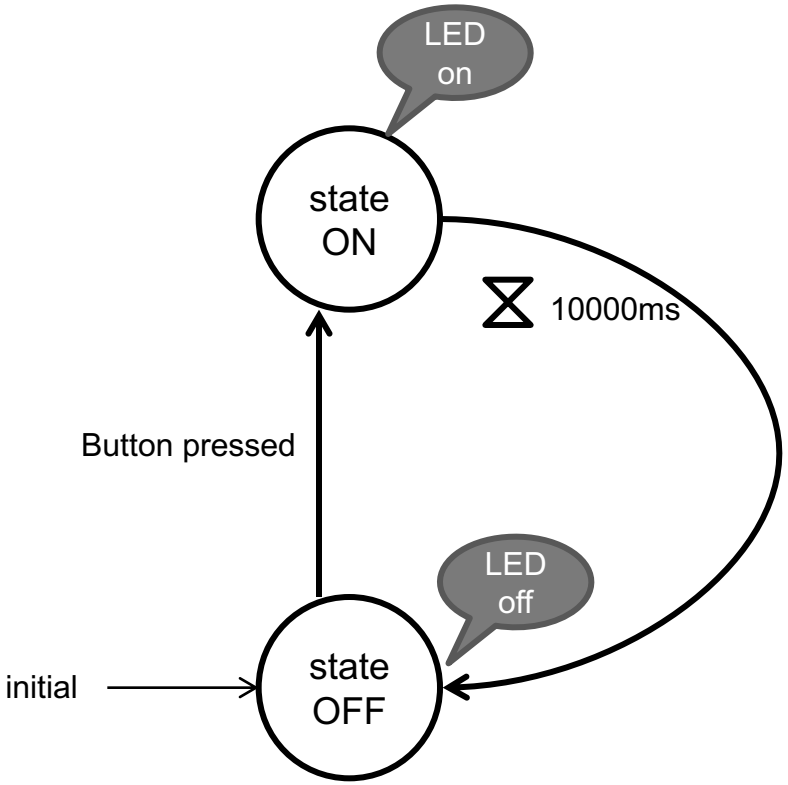
Main idea:

- **Create a function for each “task” the Arduino performs**
- **Call these functions in rapid succession in main loop**
- **Use State Transition Diagrams**

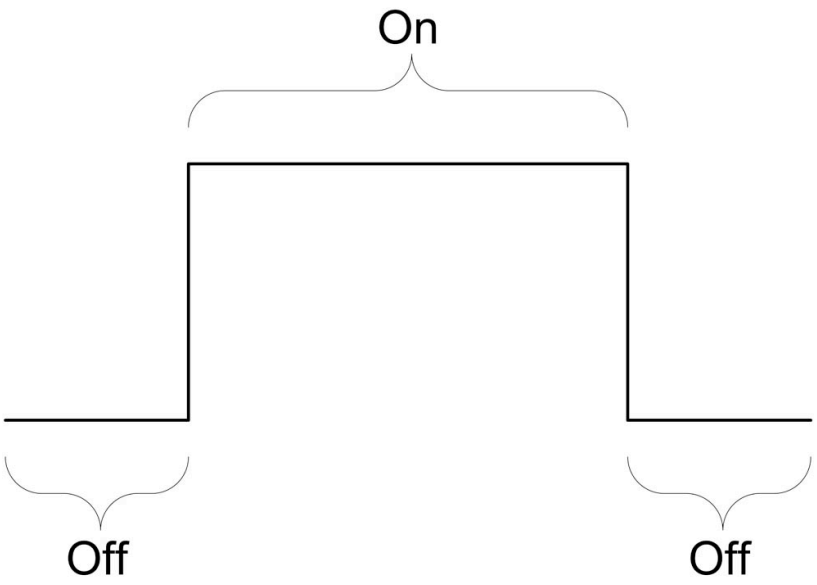
STATE TRANSITION DIAGRAM



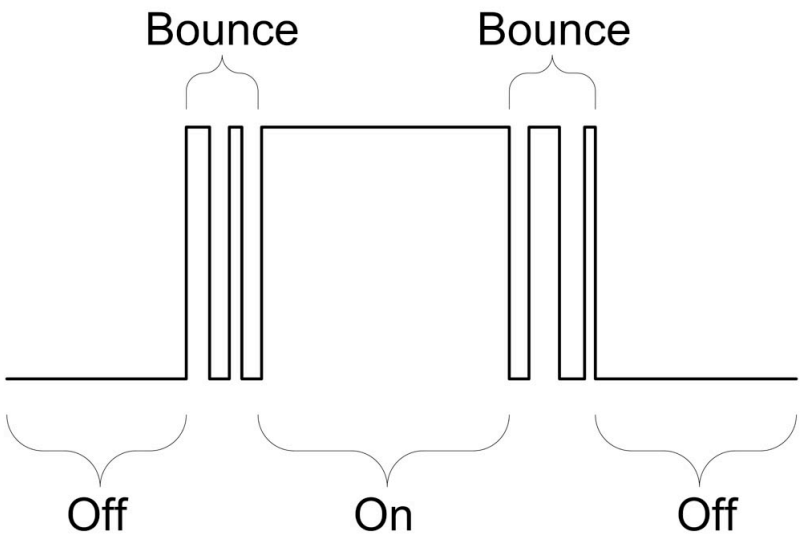
TIMED LIGHT SWITCH



BUTTON DEBOUNCING

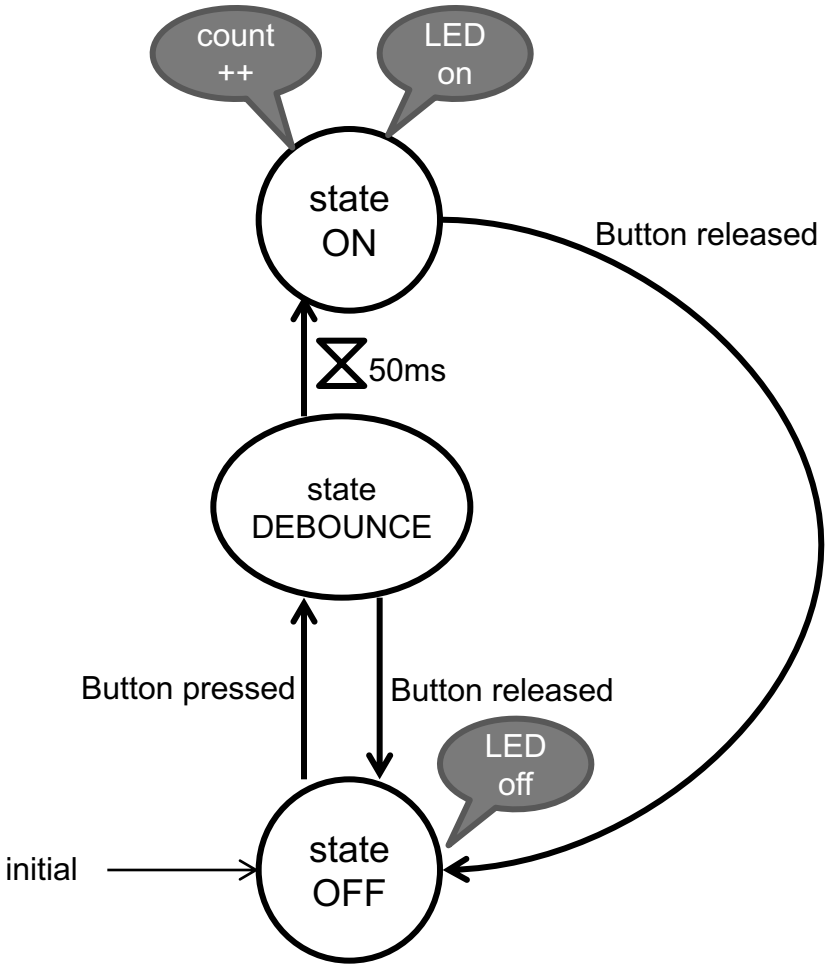


IDEAL



REALITY

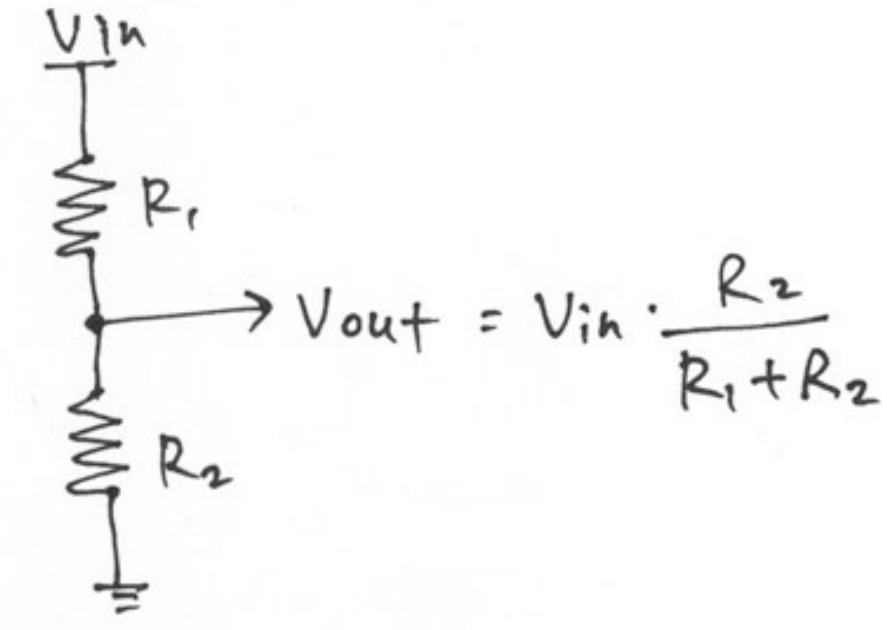
DEBOUNCING WITH STATES



SENSORS

RESISTIVE SENSORS

VOLTAGE DIVIDER



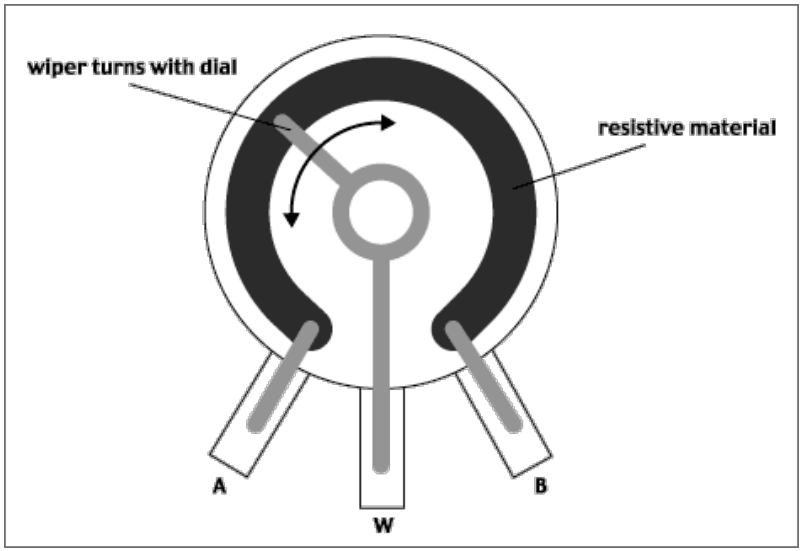
V is proportional to R, so the voltage drop across R_2 decreases as R_2 gets smaller.

By varying R_1 and/or R_2, we can vary V_{out} while V_{in} remains constant.

ROTATION SENSING

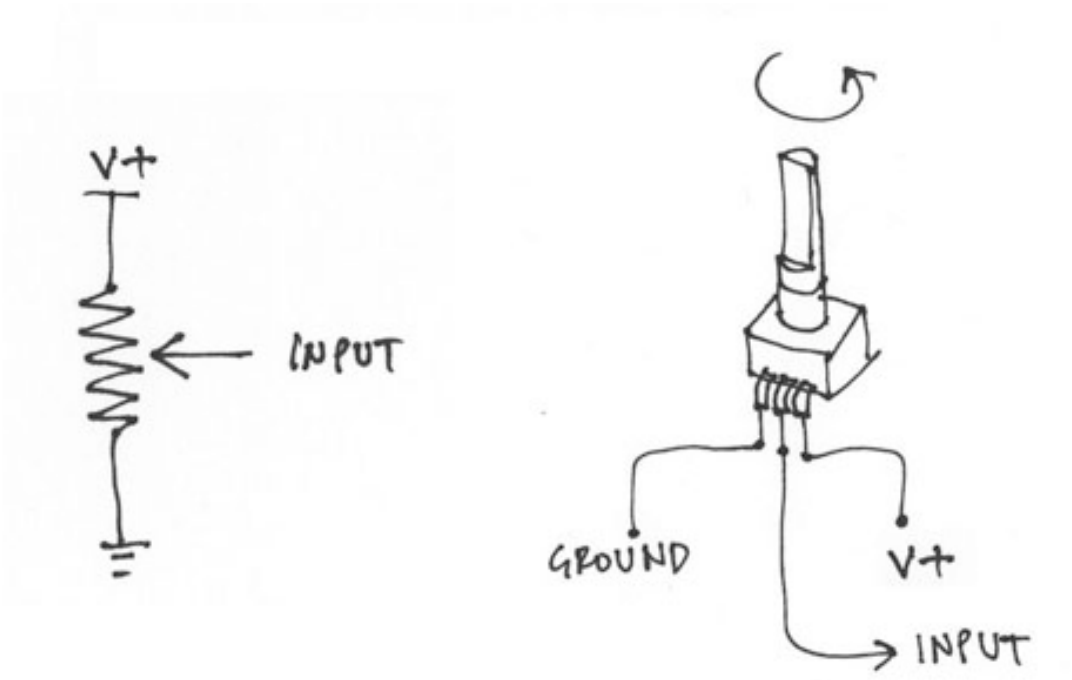
Potentiometer

- Resistive material
- Wiper moves along material
- Resistance between terminal and wiper varies with distance



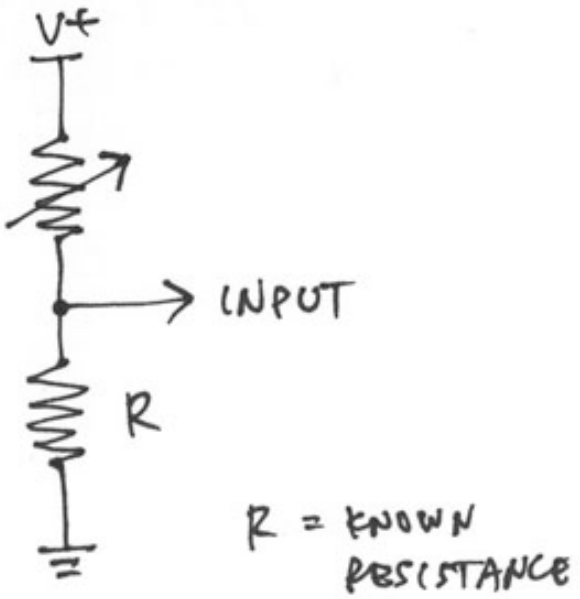
ROTATION SENSING

- The wiper “divides” the resistance into 2 parts. If we connect the outer legs of the pot to V+ and Gnd, we get a voltage divider.
- The “Input” voltage here changes as you rotate the pot.
- We can connect this to an analog input of the Ardunio to measure the voltage

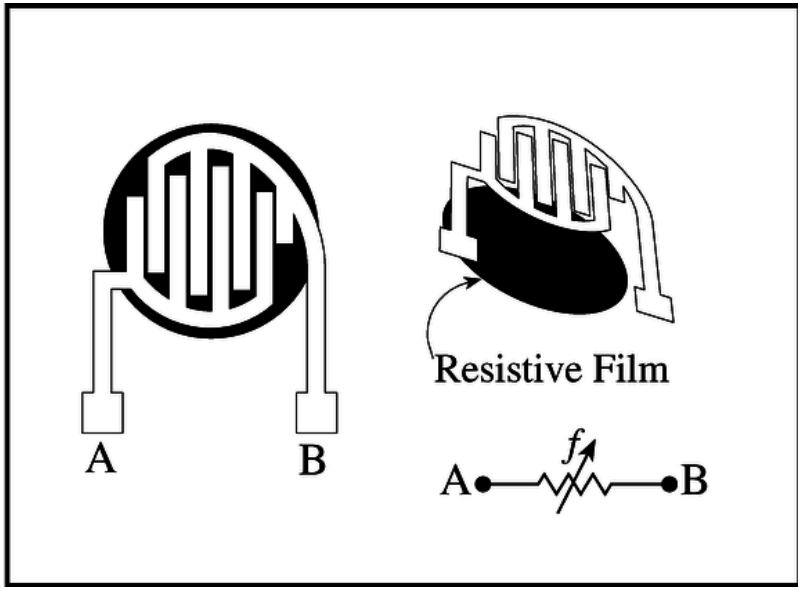


OTHER RESISTIVE SENSORS

- Some sensors have a varying resistance between their 2 leads, depending on some change in a physical property
- Need a fixed resistor to make a voltage divider
- Resistance of the fixed resistor affects range of output voltages



FORCE SENSING RESISTOR (FSR)

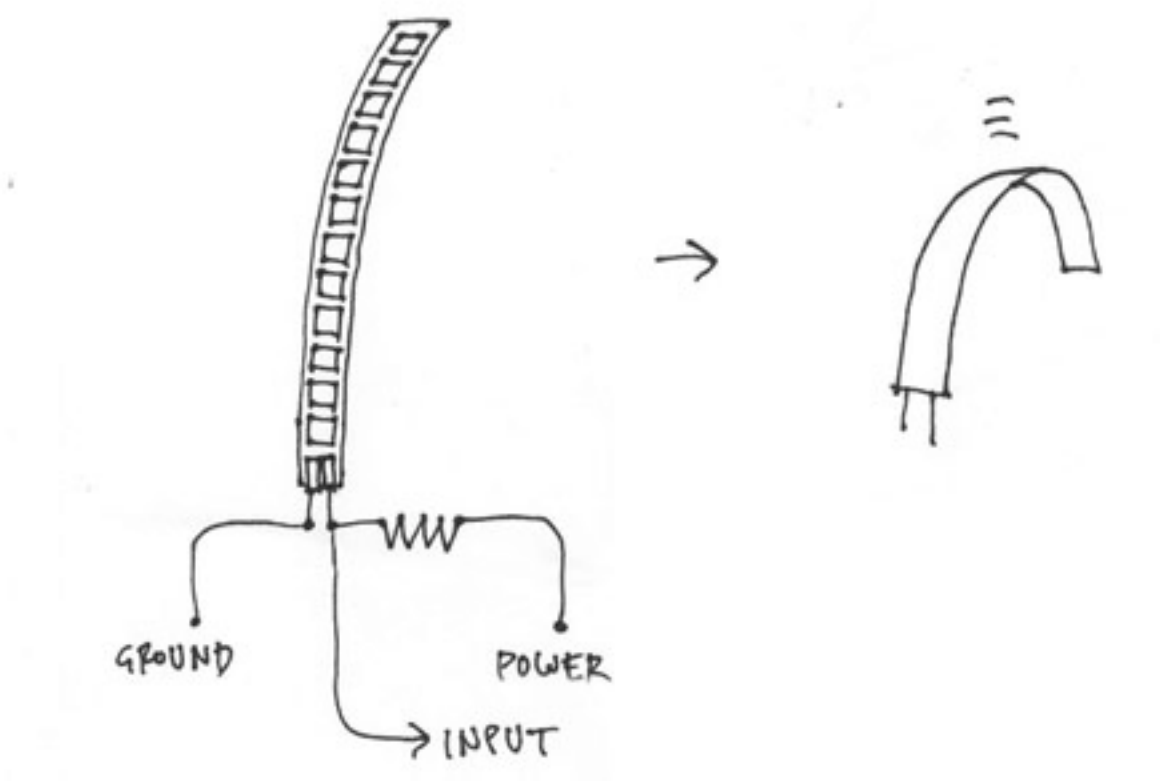


- **Resistive Film**
- **Interdigitating contacts**
- **Resistance inversely proportional to force**
- **Conductance proportional to force**

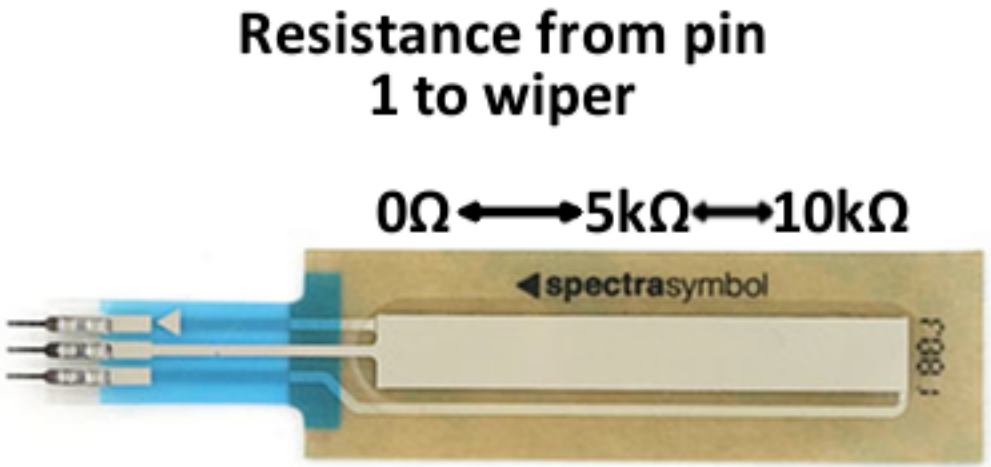
See http://www.openmusiclabs.com/learning/sensors/fsr/

FLEX SENSOR

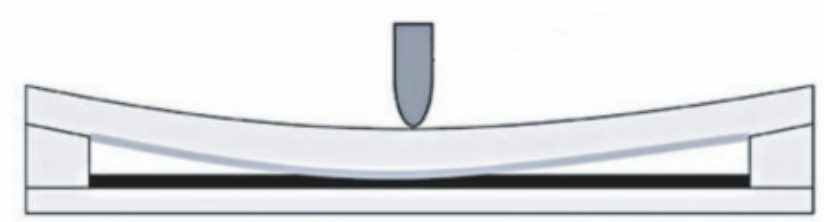
- A flex sensor changes resistance as you bend it
- Most sensors only work in 1 direction



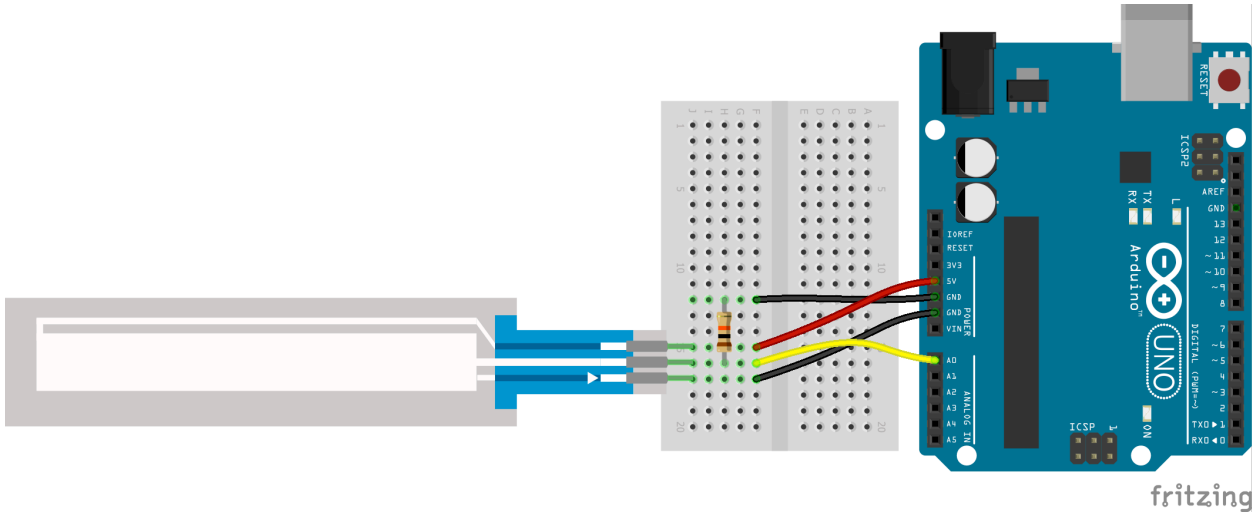
SOFTPOT



- Need a 10kΩ resistor between the wiper and ground to avoid “floating” voltage when not pressed

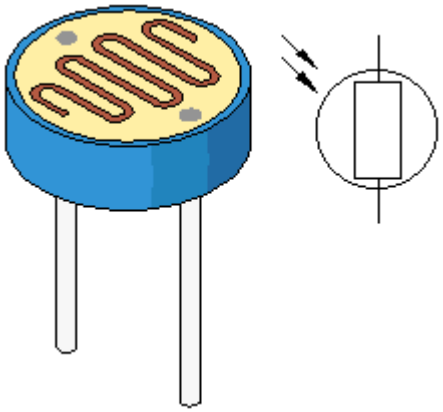


SOFTPOT



- Need a 10kΩ resistor between the wiper and ground to avoid “floating” voltage when not pressed

LIGHT DEPENDENT RESISTOR (LDR) OR PHOTOCELL



- **Resistance decreases with increasing light intensity**

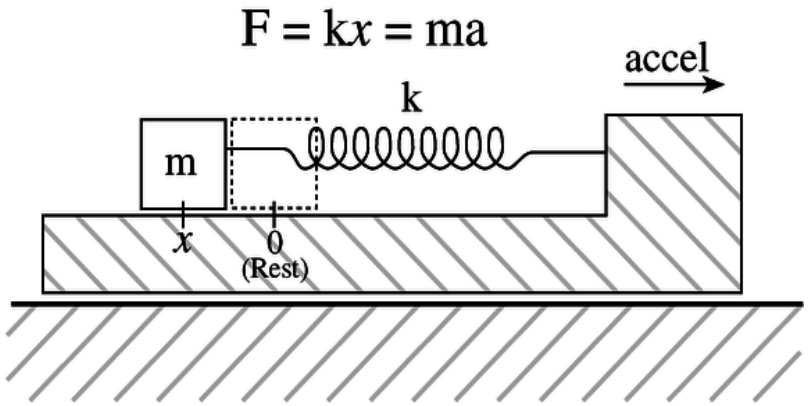
VOLTAGE-OUTPUT SENSORS

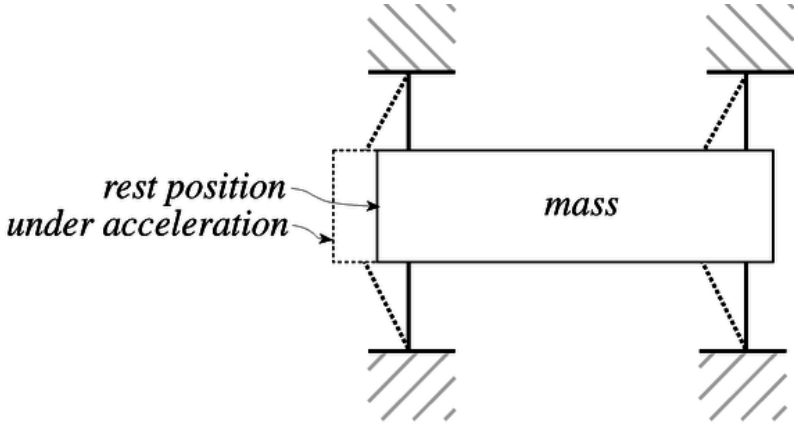
VOLTAGE-OUTPUT SENSORS

- Some sensors have integrated circuits built in that output a varying voltage directly.
- Pros:
  - easy to use
  - No external circuitry required, just power & ground connections
- Cons
  - expensive
  - Some have fixed voltage supply and/or output ranges

ACCELEROMETER

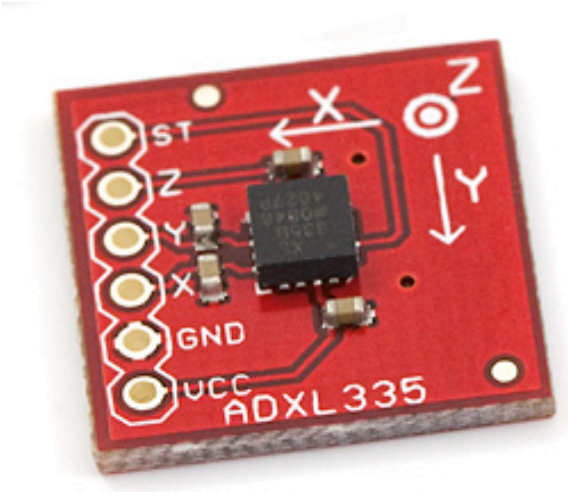
- Acceleration is proportional to the displacement of a mass attached to a spring.
- Micro-machined mass-springs allow us to measure acceleration (displacement) via capacitive sensing





ADXL335 ACCELEROMETER

- 3-axis accelerometer
- **Powered with 3.3V**
- Outputs 0-3.3V from -3 to +3g

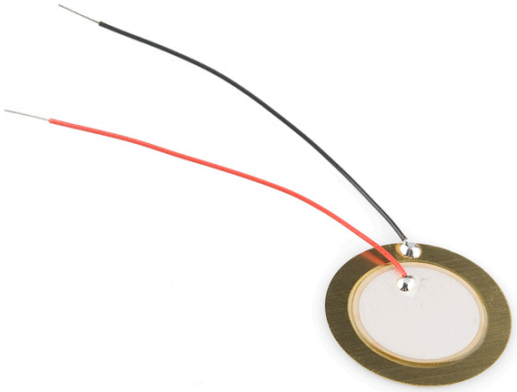


https://www.sparkfun.com/products/9269

PIEZOELECTRIC SENSORS

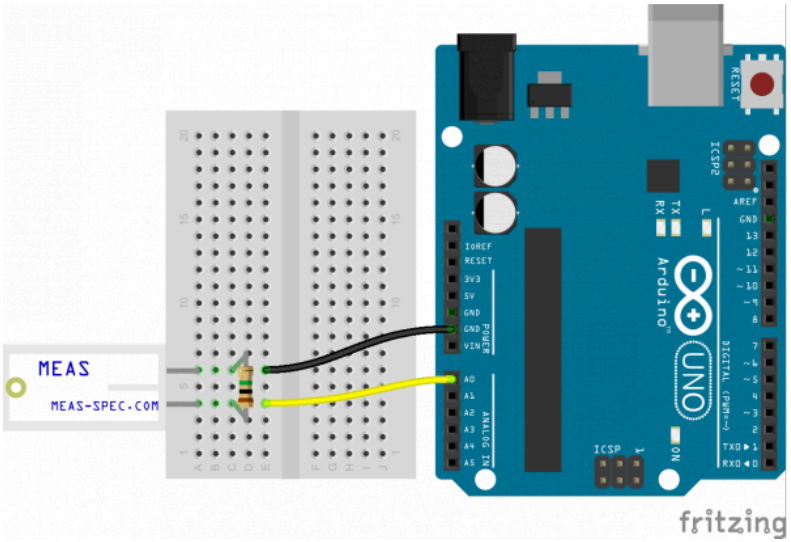
- Employ materials that exhibit the “piezoelectric effect:” generate a charge (voltage) when deformed or pressure is applied
- Usually a crystal or ceramic in the shape of a thin film or disc
- Effect is reversible: piezo element will deform in response to an applied voltage.
    - If that voltage varies regularly, the piezo will vibrate (and vice versa) – it can be a contact microphone or a loudspeaker

- Piezo vibration sensor



- Piezo element

PIEZOELECTRIC SENSORS



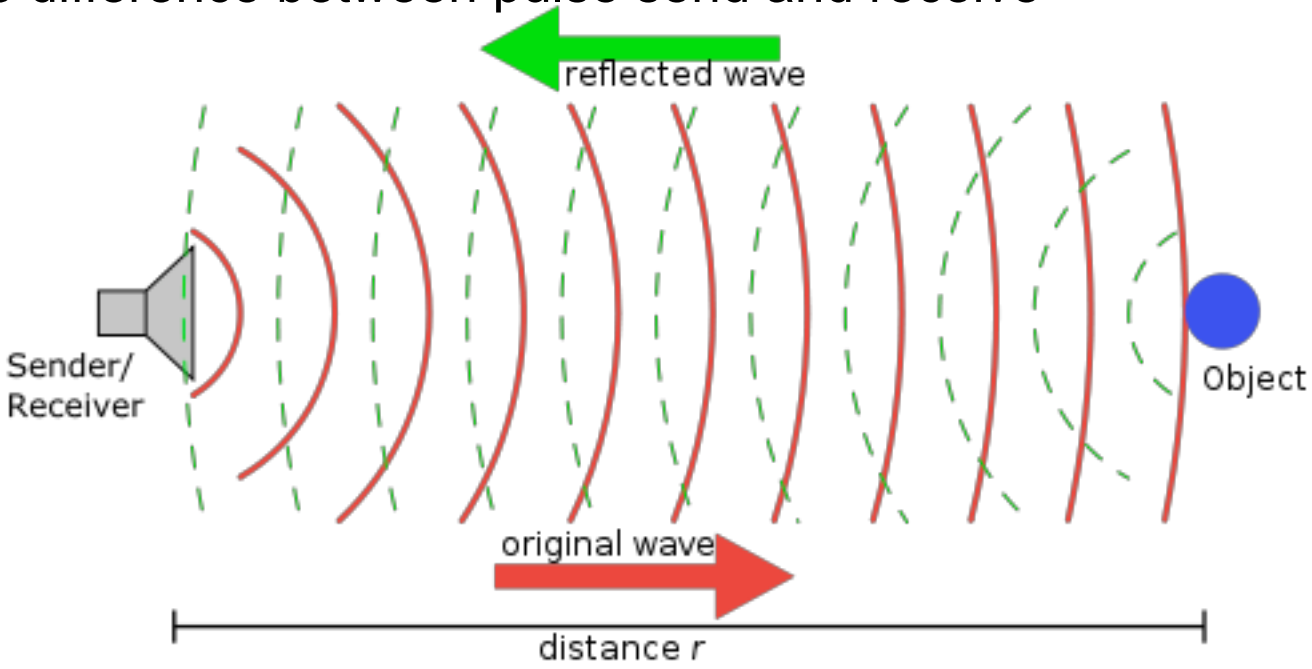
- Piezos can output very high voltages, which can damage your arduino. Use a 1MΩ resistor in parallel to reduce the voltage from the sensor.
- See arduino “Knock” tutorial: https://www.arduino.cc/en/tutorial/knock

EMITTER-DETECTOR PAIRS

REFLECTIVE SENSORS

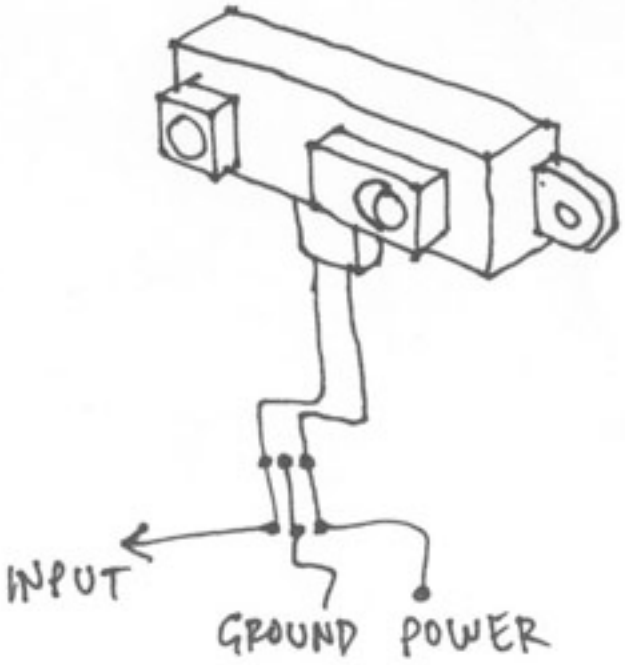
Principle: send out a pulse (light, sound), measure either

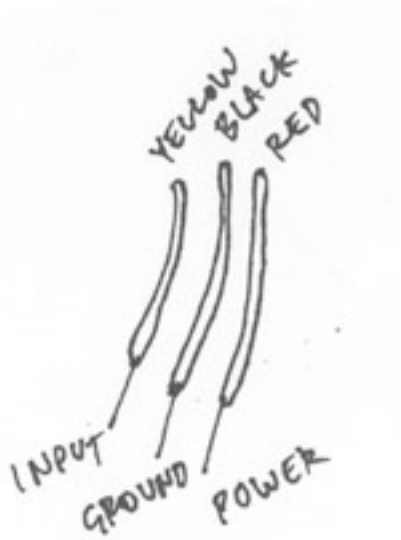
- Intensity of reflected signal
- Time difference between pulse send and receive



INFRARED PROXIMITY SENSOR

- Some sensors have integrated circuits built in that output a varying voltage directly. These are powered with +5V and Gnd
- Infrared Proximity Sensor





https://www.sparkfun.com/products/242

REFLECTIVE OPTICAL SENSOR

Turn on an infrared LED

Use an IR photodiode to measure intensity of reflected light

Advantages:

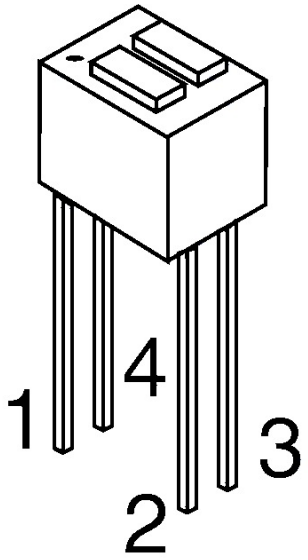
- Fast
- Inexpensive
- Minimal circuitry
- Some give direct voltage output

Disadvantages

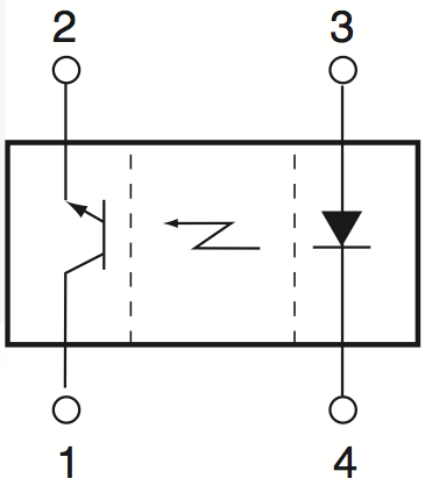
- Subject to interference
- Shorter range

https://www.sparkfun.com/products/246

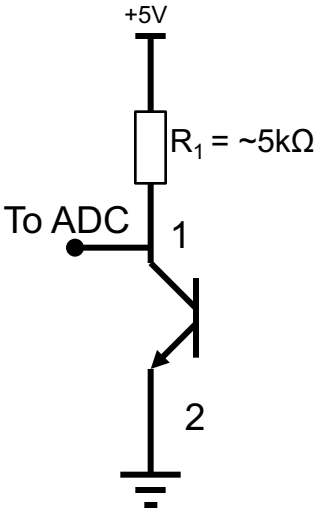
REFLECTIVE OPTICAL SENSOR

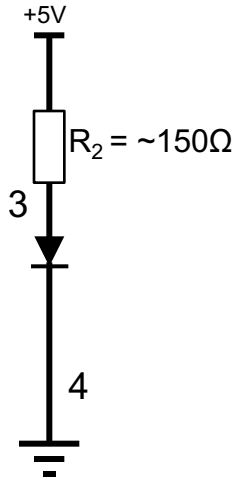


pinout



Internal schematic

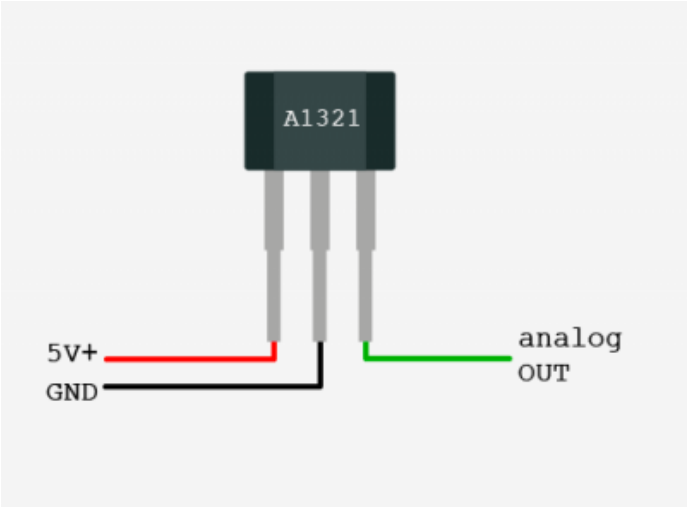




Circuit diagram

HALL EFFECT SENSOR

- Varying output voltage in response to a magnetic field
- 2 basic types:
  - **Switches**: on/off in presence/absence of a magnetic field
    - Can be "**latching**," i.e., output state remains steady after the field has been applied, until it sees a field of the reverse polarity
  - **Linear:** output is continuously variable, proportional to magnetic field strength.



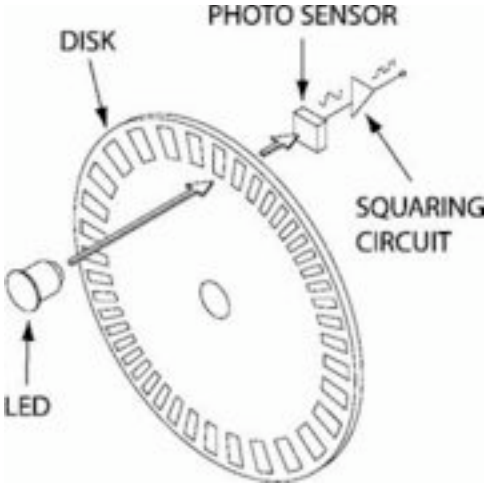
- Allegro 1321 Hall effect sensor.
- This sensor will output 2.5V with no magnetic field and go up/down from there based on magnetic polarity.

ENCODERS

ROTATION SENSING

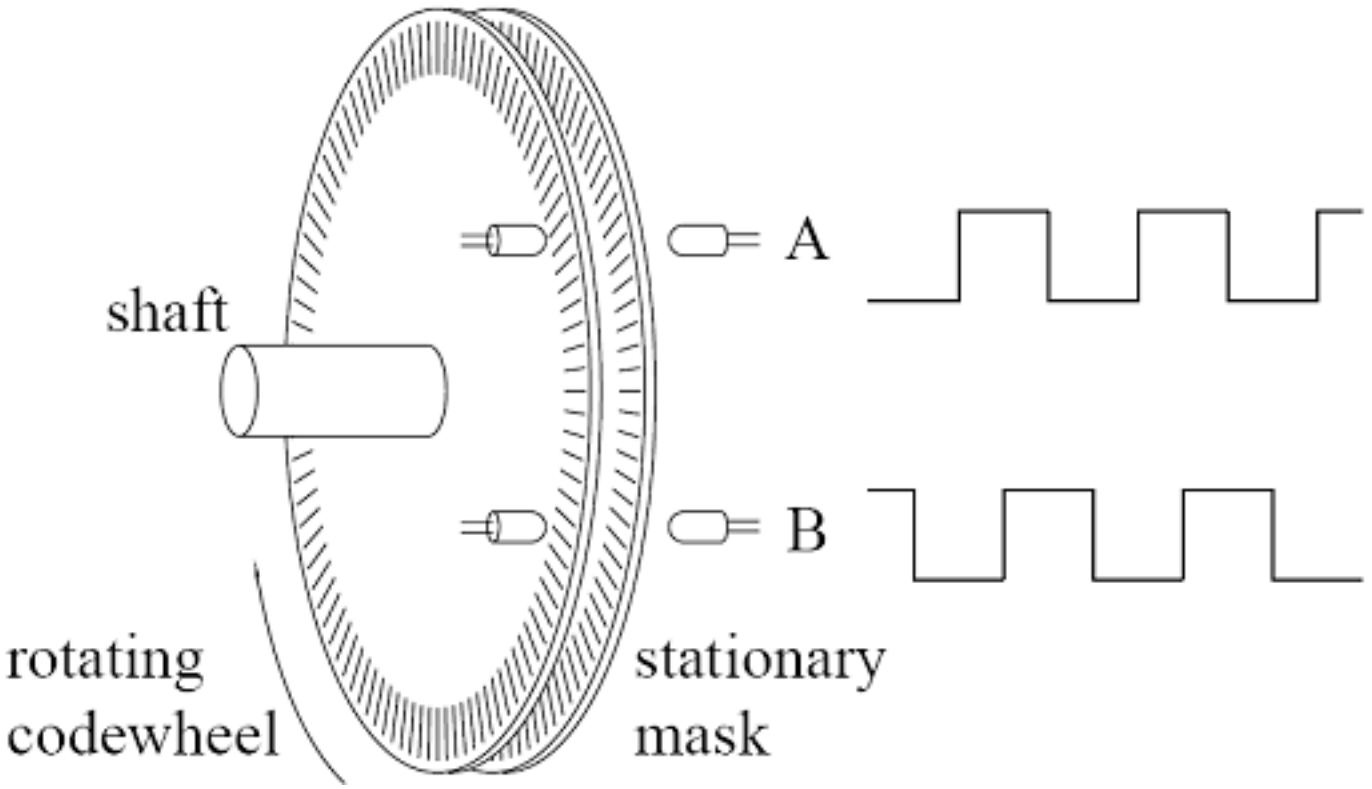
Rotary encoder

- Sequence of logic transitions
- Count number of transitions: tell how far you've traveled



- Can be non-contact
- Can turn infinitely
- Problems?

QUADRATURE ENCODER

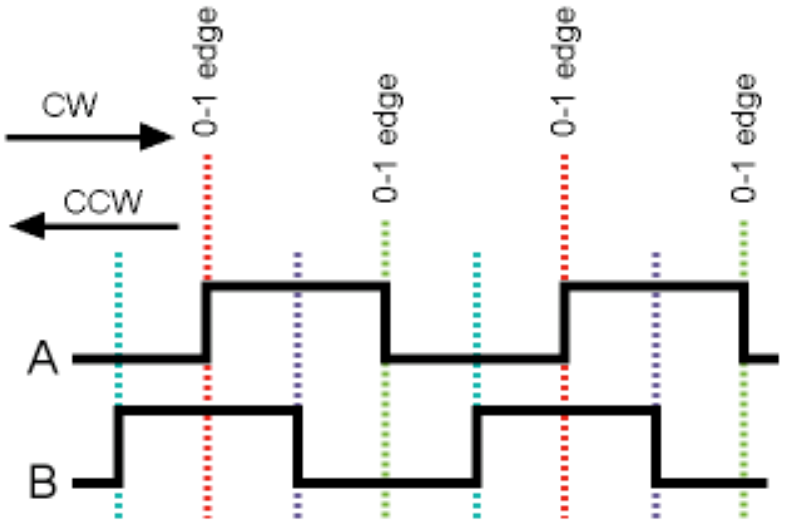


QUADRATURE ENCODER

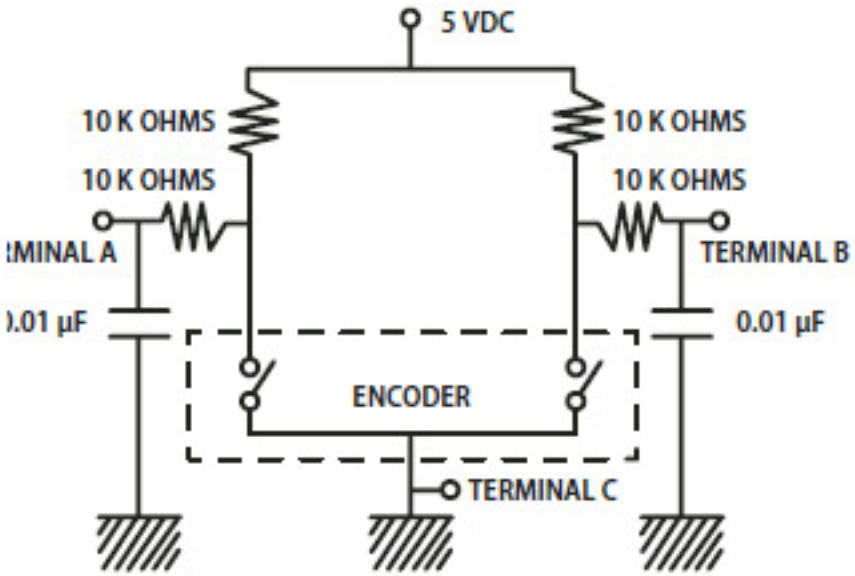
2 square wave signals, 90º out of phase

Basic operation:

- Read signal A
- If A goes from Low to High
  - Read Signal B
  - If B is High, you've moved 1 tick CW
  - If B is Low, you've moved 1 tick CCW



ROTARY ENCODERS



- Encoder circuit

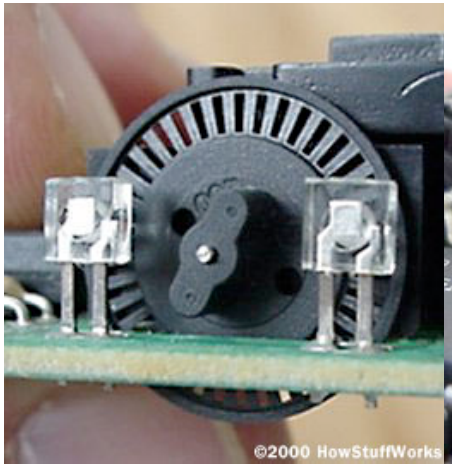
Several different software / programming approaches. See:

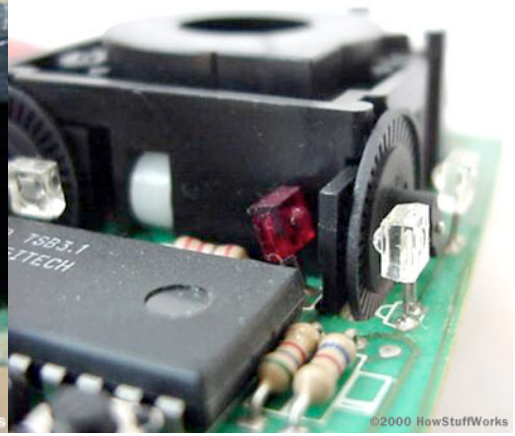
http://www.arduino.cc/playground/Main/RotaryEncoders

APPLICATIONS

UI controls: Volume knobs

Mouse





Industrial systems

- Robotic positioning
- Gear tooth rotation counting
    - Magnetic sensing with Hall Effect sensor

Rotation speed sensing

- Measure elapsed time between ticks (or every n ticks)

DIGITAL COMMUNICATION

Problem: We need to send data from one device to another?

- Data: A series of digital (binary) values, organized in bytes.

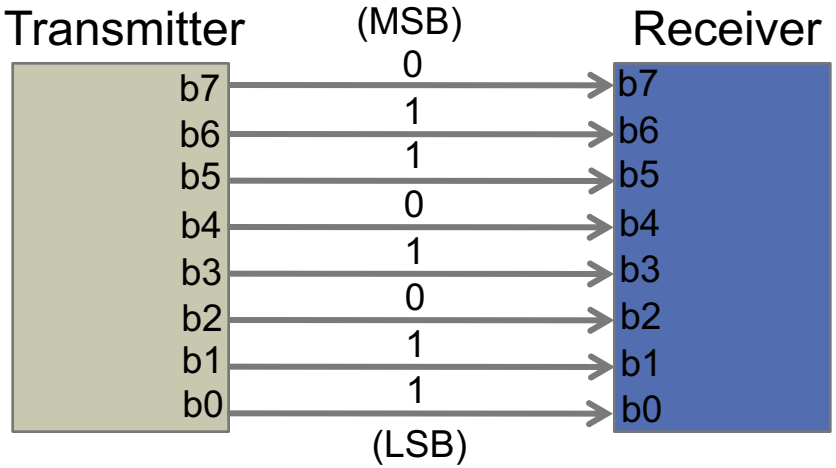
Why?

- Mouse ➔ computer
- Computer ➔ printer
- Synth keyboard ➔ sampler
- Control surface ➔ protools
- Game controller ➔ Xbox
- Arduino ➔ Max/MSP

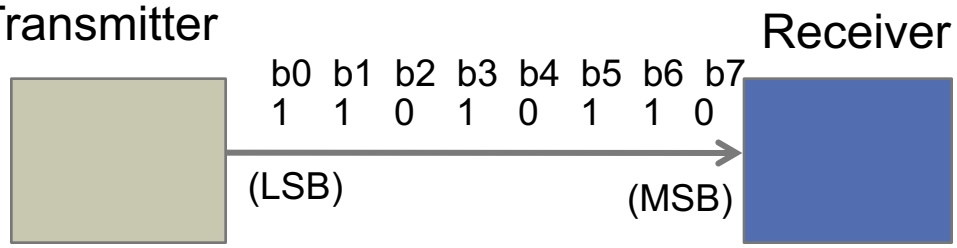
PARALLEL VS. SERIAL

Say we want to send the byte 0110 1011

Parallel (simultaneous, happening together)



Serial (sequential, in series)



PARALLEL VS. SERIAL

Parallel

Pros

- Fast
- Don't need to serialize/deserialize data

Cons

- Lots of cables/wires/pins
- Crosstalk
- Clock skew
- Fixed data size

Serial

Pros

- Fewer wires
- No crosstalk
- Cheaper
- Scalable

Cons

- Need to serialize/deserialize
- Slower

PARALLEL COMMUNICATION ON THE ARDUINO

Traditionally, microcontrollers used Parallel communication for communicating between devices

- AVR on Arduino is organized into 8-pin “Ports”
- If I want to write the byte 0110 1011 to Port A, I use the code

```
PORTA = 0110 1011;
```

- And then I can set the state of 8 pins at a time
- Same goes for reading
- There are still some devices we can interact with in this way

SERIAL COMMUNICATION

Nowadays (a very recent development) most digital communication is becoming serial

- USB
- Firewire
- MIDI
- Ethernet
- Morse Code
- SATA
- PCI Express

Faster clocks, skew resistance, less crosstalk, low cost make it practical

REQUIREMENTS FOR SERIAL COMMUNICATION

Protocol

- Word size – how many bits at a time?
- Transmission rate (baud rate)
- Endian-ness (MSB first or LSB first)
- Start/stop signals
- Error correction
- Voltage level
- Bit representation
- Physical connectors

ASYNCHRONOUS VS SYNCHRONOUS SERIAL COMMUNICATION

Asynchronous

- Data is transmitted at any time
- When transmission begins, bits still need to be sent at a regular rate (known amount of time for each bit)
- This is what we've been doing!

Synchronous

- Devices share a clock
- Data is only transmitted at fixed times in relation to the clock signal

SERIAL COMMUNICATION ON THE ARDUINO

We primarily use the AVR's UART

- Universal **Asynchronous** Receiver/Transmitter
- Serial

Uses pins D0 and D1 (Rx and Tx)

Uses (ancient, but cheap & easy) RS-232 protocol

- On a computer, commonly called a "Serial Port"
- Has a fixed range of baud rates

On the Arduino, this is connected to a chip that converts RS-232 into USB and vice-versa

A driver on your computer makes the USB data look like a Serial Port

SERIAL COMMUNICATION ON THE ARDUINO

- The UART allows us to send a stream (series) of binary numbers in 8-bit chunks (bytes)
- We need to know how to interpret that stream
- Arduino Serial Monitor interprets each byte as an ASCII character code, and displays the corresponding character

```
Serial.print("Foo");
```

Sends ASCII character codes for 'F' (70), then 'o' (111), then 'o' (111) again.

```
Serial.println("Bar");
```

Sends ASCII character codes for 'B' (66), then 'a' (97), then 'r' (114), followed by the 'carriage return' (13) and 'newline' character (10).

SERIAL COMMUNICATION ON THE ARDUINO

- When we send numerals, e.g. sensor data, `Serial.print` does not send the binary data itself.
- Rather, it sends the ASCII codes for the numerals representing the data.

```
int fred =  249;
Serial.print(fred);
```

Sends ASCII character codes for ‘2’ (50), then ‘4’ (52), then ‘9’ (57).

SERIAL COMMUNICATION WITH MAX

- Max's [serial] object outputs the numerical value of each byte it receives
- If you try to send sensor data, Max will receive ASCII codes

```
int sensor =  249;
Serial.print(sensor);
```

- The [serial] object in Max will output: 50, 52, 57
- You can convert this to characters using the [itoa] object, but more often, you want to **use** the binary data, i.e., to control sound parameters.
- Instead, send the raw binary data using `Serial.write(sensor);`

```
int sensor =  249;
Serial.write(sensor);
```

Sends the byte 249 (1111 1001)

SERIAL COMMUNICATION WITH MAX

- Great! Now we can send raw sensor data to Max.
- But what if we want to send data from 2 sensors?

```
void loop () {
        int reading0;
        int reading1;
        reading0 = analogRead(0);
        reading1 = analogRead(1);
        Serial.write(reading0);
        Serial.write(reading1);
}
```

- When I receive these 2 numbers, how do I know which corresponds to sensor 0 and which corresponds to sensor 1? (Assuming the values are unknown in advance, i.e., the result of A/D conversions.)
- Must assume that the receiver starts listening at an arbitrary time, therefore can't rely on order

SERIAL COMMUNICATION WITH MAX

- Answer: You can't know which number is which.

```
void loop () {
        int reading0;
        int reading1;
        reading0 = analogRead(0);
        reading1 = analogRead(1);
        Serial.write(reading0);
        Serial.write(reading1);
}
```

- So what do we do?

SOLUTION: DESIGN A PROTOCOL

- Let's use a special character to delineate the start of set of sensor readings. We call this a DELIMITER
- We'll use the number 255 (b1111 1111)

```
void loop () {
        int reading0;
        int reading1;
        reading0 = analogRead(0);
        reading1 = analogRead(1);
        Serial.write(255);
        Serial.write(reading0);
        Serial.write(reading1);
}
```

- Now, every time we receive the number 255, we know that the next byte we receive will correspond to the reading from sensor 0, and the following will correspond to the reading from sensor 1

TWO PROBLEMS

```
void loop () {
        int reading0;
        int reading1;
        reading0 = analogRead(0);
        reading1 = analogRead(1);
        Serial.write(255);
        Serial.write(reading0);
        Serial.write(reading1);
}
```

1. `analogRead()` returns values from 0-1023. These are bigger than 8 bits.

2. What if our data – one of the readings – happens to be 255, the same value our delimiter?

SOLUTION: SCALE YOUR DATA

```
void loop () {
        int reading0;
        int reading1;
        reading0 = analogRead(0);
        reading1 = analogRead(1);
        Serial.write(255);
        Serial.write(reading0 >> 3);
        Serial.write(reading1 >> 3);
}
```

- The bit shift right operator >> takes every bit in a binary number and moves it to the right by the number of places specified in the second operand.
- Assume infinite leading zeroes.
- When the least significant bit gets shifter right, it disappears

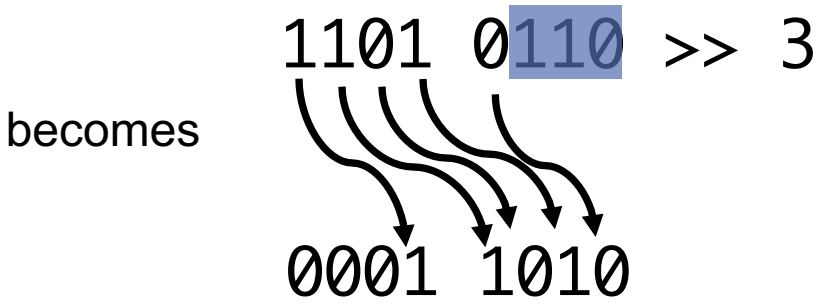
SOLUTION: SCALE YOUR DATA

- Therefore

```
(214 >> 3)
```

In binary is:

```
(1101 0110 >> 3)
```



Or, in decimal: 26

SOLUTION: SCALE YOUR DATA

- Therefore

```
(1023 >> 3)
```

In binary is:

```
(11 1111 1111 >> 3)
```

```
11 1111 1111 >> 3
```

is equal to

```
0111 1111
```

Or, in decimal:

```
127
```

SOLUTION: SCALE YOUR DATA

- Effectively, a bitshift by 3 to the right is equivalent to integer division by 8
- We can also think of it as retaining the 7 MSB of our original number
- We've scaled our values in the range of 0-1023 to a range of 0-127
- We've also guaranteed that our delimiter (255) will never appear in our data stream

BUT WHAT IF I WANT MY 10-BIT RESOLUTION?

- Decompose 10-bit number into a 7-bit number and a 3 bit number

- Still guarantees that your delimiter (255) will not appear in the data stream

DECOMPOSING A 10-BIT NUMBER

- We know how to get the 7 most significant bits of a number. It's just:

```
(n >> 3)
```

- To get the 3 least significant bits, we use the binary AND operator & and a **BIT MASK**

- The binary AND operator works like this

```
0 & 0 = 0
0 & 1 = 1
1 & 0 = 0
1 & 1 = 1
```

- So, if we want to retain only certain bits of a number, we can AND that number with 1s in the position of the bits we want.

DECOMPOSING A 10-BIT NUMBER

- Therefore, if we want the 3 least significant bits, we can & the number with binary 111

- For example:

```
  0110 1011
&
  0000 0111
-----------
  0000 0011
```

- In C code, we could write something like:

```
int reading, msb, lsb;
reading = analogRead(0);
msb = reading >> 3; // 7 msb of a 10-bit number
lsb = reading & 7;  // 3 lsb of a 10-bit number
                    // (111 in decimal is 7)
```

DECOMPOSING A 10-BIT NUMBER

- Putting it all together, here is our communication protocol:

```
void loop () {
        int readings[NUMSENSORS];
        Serial.write(255);
        for (int i=0; i<2; i++) {
                val[i] = analogRead(i);
                Serial.write(val[i] >> 3);
                Serial.write(val[i] & 7);
        }
}
```

- Works with NUMSENSORS from 1 to 6 (as many analog inputs as are on the Arduino Uno)
- Receiver needs to know how the data is formatted.
- See a2dToSerialMaxMultiple in Arduino Examples

SPI SERIAL PROTOCOL

- Serial Peripheral Interface (Synchronous)
- http://arduino.cc/en/Reference/SPI
- “Master/Slave” protocol
- Minimum 3 connections between devices:
  - **SCK** (Serial Clock) - The clock pulses which synchronize data transmission generated by the master
  - **MOSI** (Master Out Slave In) - The Master line for sending data to the peripherals,
  - **MISO** (Master In Slave Out) - The Slave line for sending data to the master,
- One additional line per device (not needed if only 1 peripheral)
  - **SS** (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.
  - If LOW, device communicates with master, if HIGH, it ignores commands and doesn’t send data
  - Allows multiple slave devices with a single master

I²C SERIAL PROTOCOL

- Inter-Integrated Circuit, also known as “Two-Wire Interface”
- More recent than SPI
- http://www.arduino.cc/en/Reference/Wire
- “Master/Slave” protocol that allows for multiple masters
- Instead of SS, uses a **Bus**, with each device having an **Address.**
- 2 connections between devices:
  - **SCL**(Serial Clock) – Clock pulses generated by master to synchronize data transmission
  - **SDA** (Serial Data) – Bidirectional data line for communication between master and slave. Each device has transmit and receive mode.
- All devices on the bus share the same 2 connections. Devices know the target of communication according by address.

EXAMPLE SPI/I2C DEVICES

- Accelerometers, barometers and other sensors
- Audio players
- DACs / ADCs
- GPS
- Input devices like barcode scanners
- Wireless transceivers
- Controllers for flash memory, network or wireless devices
- LED or LCD displays

ALL ABOUT CONNECTORS

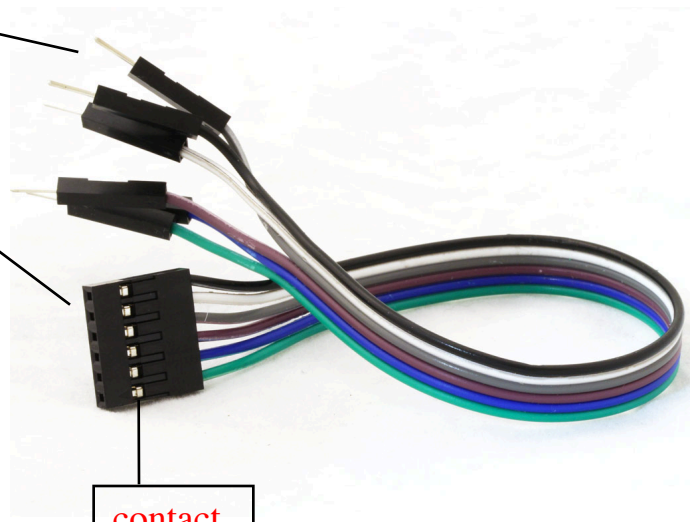
CONSIDERATIONS

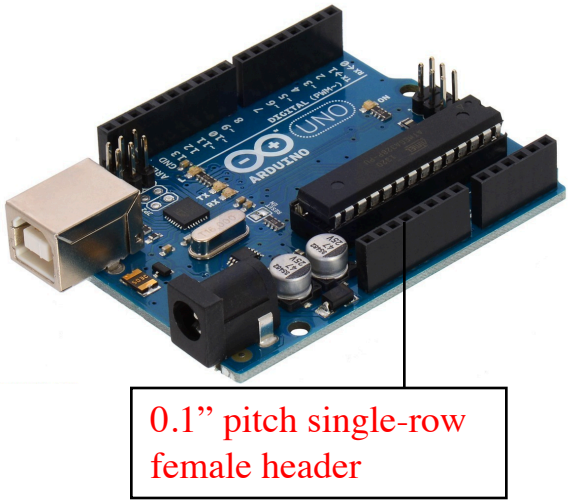
There is no single perfect solution for all interconnect applications. Need to weigh:

- **Number of connections**
- **Length of wire / distance**
- **Rigidity**
- **Force / torque on the connection**
- **Permanence (ability to connect/disconnect)**
- **Modularity**
- **Cost**
- **Time**

SOME TERMS AND FACTS

- **Gender**
  - Plugs in, has pins (male)
  - Is plugged into, has receptacles (female)
- **Contacts**
  - The part of the connector (almost always metal) where the current flows (sometimes shrouded)
- **Header**
  - An array of equally-spaced identical connectors, in one or more rows, either male or female
- **Pitch**
  - In connectors with an array of contacts (headers), the spacing between centers of adjacent contact
  - Arduino headers and many others we use are 0.1” or 2.54mm pitch





MORE TERMS AND FACTS

- **Mount**
  - Indicates where a component is intended to be mounted, e.g.:
  - **Panel Mount** — attached to the front/back panel of a piece of gear
  - **Board or PCB Mount** — soldered to a circuit board
  - **Cable Mount or Free-Hanging** — attached only to a cable/wire

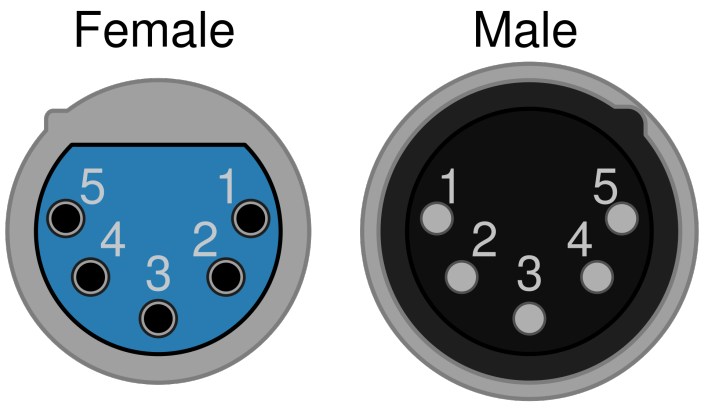
L to R Board-mount, cable-mount, and panel mount versions of the same connector

- **Cable Assembly**
  - A cable that you purchase preassembled with connectors already attached.
  - A normal USB A-to-B cable we use for the Arduino is technically called a cable-assembly.
  - One or both ends may be finished , “single-ended” or “double-ended”

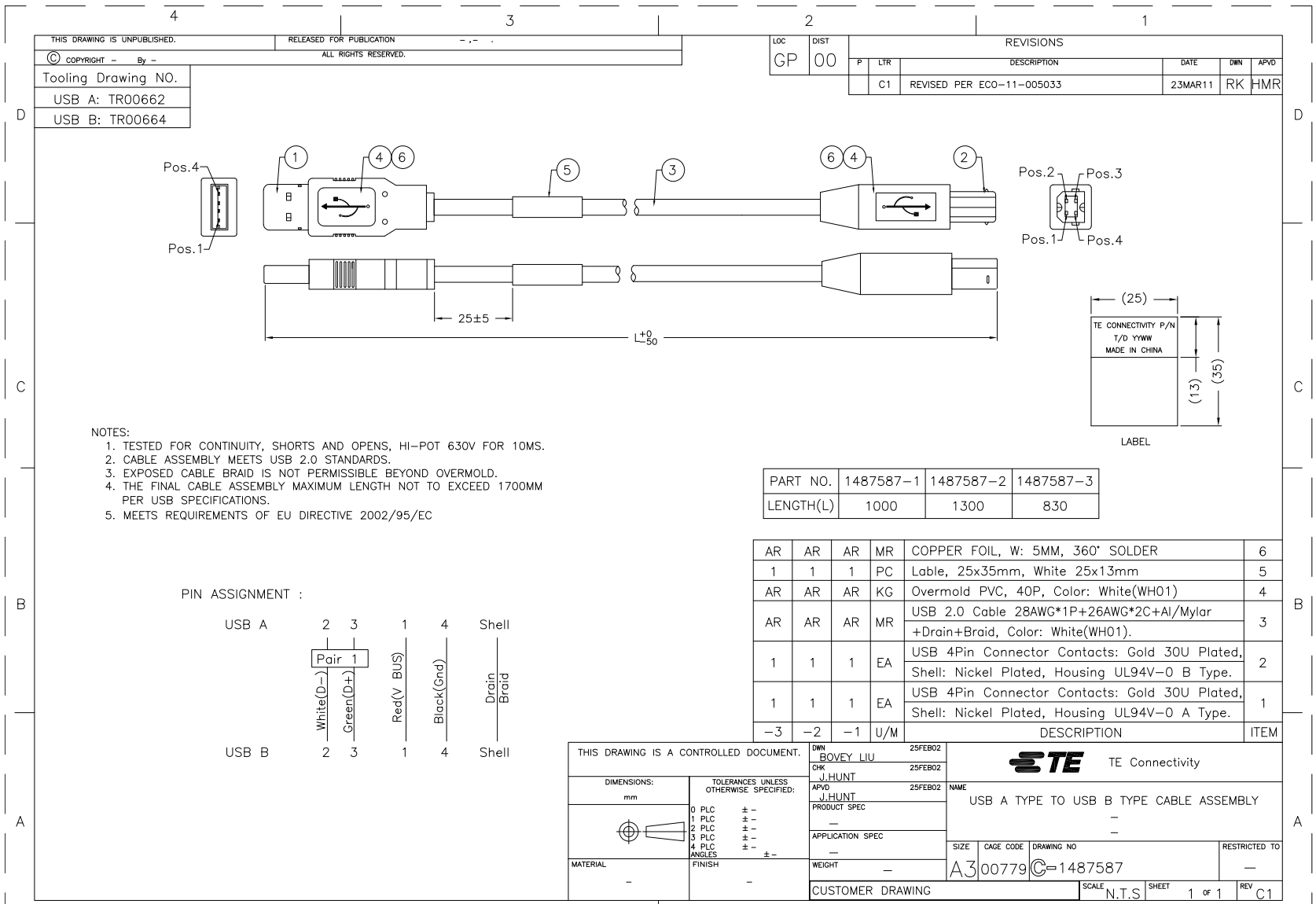
STILL MORE TERMS AND FACTS

- **Position**
  - Numerical identifier to differentiate pins in a header, component, or connector
  - Allows us to relate physical pins to their functionality on schematics or data sheets
  - Physical positions always start at ‘1’ (i.e., not 0-based)
  - Sometime synonymous with “pin”, i.e., “position 1” = “pin 1”
  - Also refers to the number of connections in a connector or assembly

5-position XLR connector



TECHNICAL DRAWING

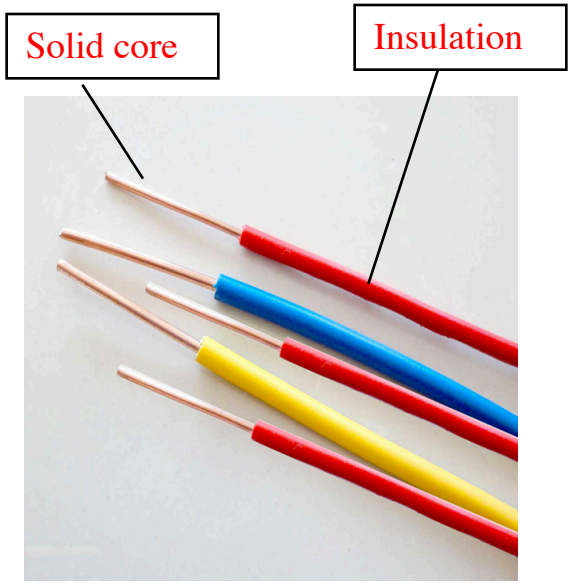


ABOUT WIRE

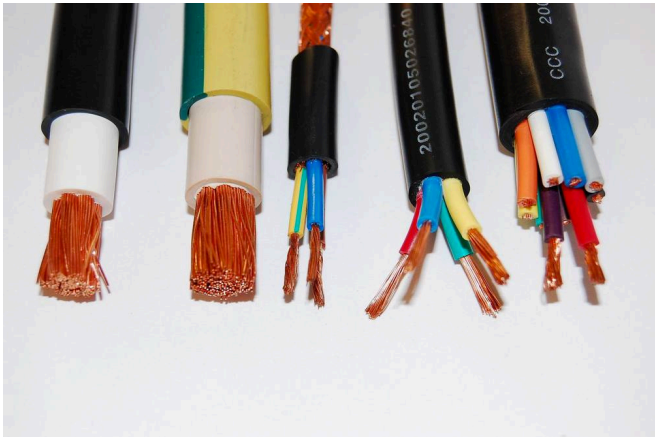
- **Gauge**
  - Refers to the diameter of wire
  - AWG or American Wire Gauge is the standard
  - Diameter **decreases** as gauge increases
  - Some tools are intended for specific wire gauge
  - Higher current applications require lower gauge (thicker wire)
  - We most commonly use 24AWG wire, suitable for low-current, low-voltage applications

Amps @ 12 Volts	LENGTH OF WIRE American Wire Gauge (AWG)						
	3'	5'	7'	10'	15'	20'	25'
0 to 1	18	18	18	18	18	18	18
1.5	18	18	18	18	18	18	18
2	18	18	18	18	18	18	18
3	18	18	18	18	18	18	18
4	18	18	18	18	18	18	18
5	18	18	18	18	18	18	18
6	18	18	18	18	18	18	16
7	18	18	18	18	18	18	16
8	18	18	18	18	18	16	16
10	18	18	18	18	16	16	14
11	18	18	18	18	16	16	14
12	18	18	18	18	16	16	14
15	18	18	18	18	14	14	12
18	18	18	16	16	14	14	12
20	18	18	16	16	14	12	10
22	18	18	16	14	12	12	10
24	18	18	16	14	12	12	10
30	18	16	14	12	10	10	10
36	16	14	14	12	10	10	10
40	16	14	12	12	10	10	8
50	16	14	12	10	10	10	8
100	12	12	10	10	6	6	4
150	10	10	8	8	4	4	2
200	10	8	8	6	4	4	2

WIRE CORE TYPES

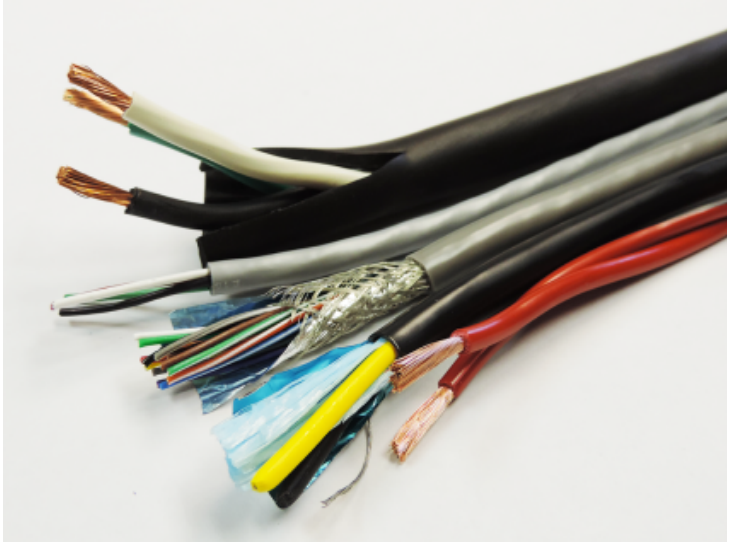


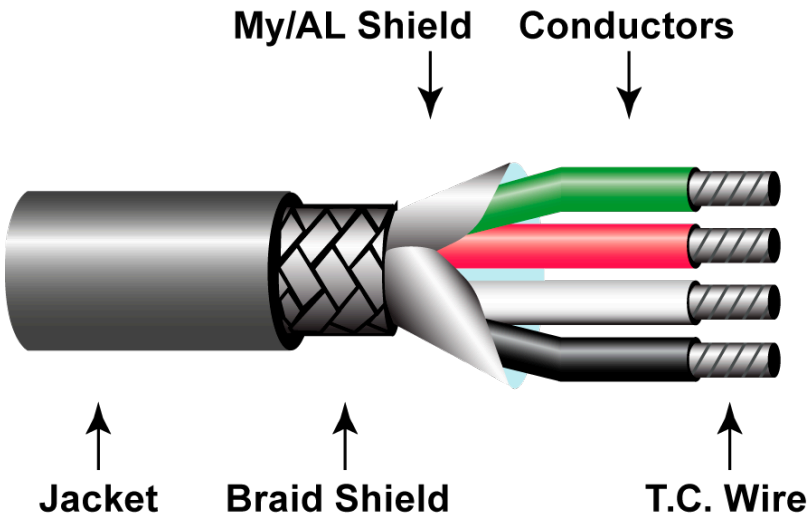
- **Solid core wire**
  - Solid core wire has a single, solid conductor in the center
  - Solid core wire is rigid, appropriate for connections where the wire will never move
- **Stranded wire**
  - The conductive core of stranded wire consists of multiple, closely spaced wire strands
  - For applications where wire will flex or move
  - In addition to wire gauge, stranded wire is specified in the format A/B, where
    - A=number of strands
    - B=gauge of each strand
  - e.g. 24AWG wire may have 7/30 or 19/36 stranding: 7 strands of 30AWG wire or 19 strands of 36AWG wire



CABLE CONSIDERATIONS

- **Multi-conductor cable**
  - Contains multiple insulated wires inside an additional insulated jacket
  - Internal wires are normally color-coded
  - Internal wires are most often stranded to make cables flexible
  - Sometimes wires are paired, and possibly twisted

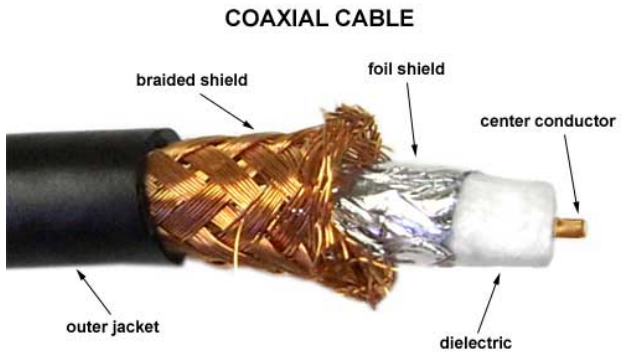




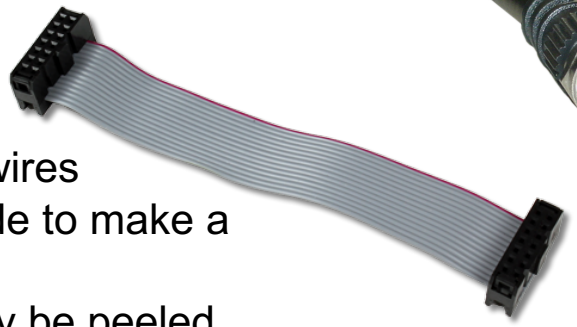
- **Shielding**
  - Multi conductor wire may have a mylar/aluminum foil shield, and/or a copper braided wire shield
  - These either reflect electromagnetic interference or collect it and route it to ground
  - A Drain Wire is sometimes connected to the foil/braid to help connect it to ground

MORE CABLES

- **Coaxial cable**
  - "Coax" wire has multiple conductors (normally just 2) that are concentric, one of which serves as a shield
  - Often used in video (e.g. cable TV), RF transmission (antennas), instrumentation (oscilloscopes)

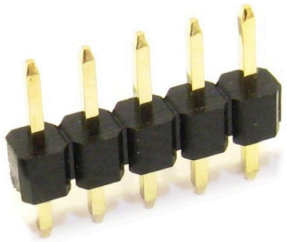


- **Ribbon cable**
  - Multiple, unshielded wires connected side-by-side to make a wide, flat cable
  - Individual strands may be peeled away to separate ends
  - But DO NOT strip away 2-3 wires to make a cable (this is very inefficient/wasteful)
  - Most often used with IDCs: insulation displacement connectors

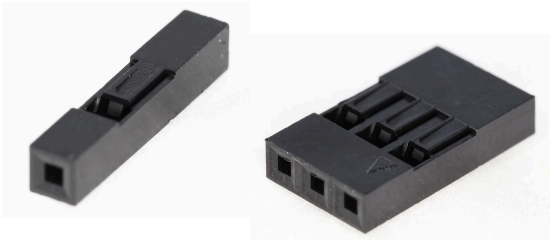


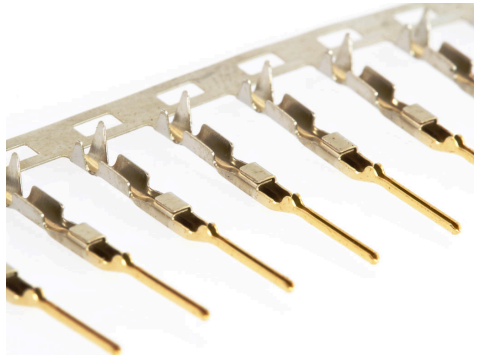
CRIMP CONNECTORS

- **Most common: “DuPont” Connector**
  - A.k.a. “DuPont Mini-PV” or “FCI” or “Berg” connector
  - Commonly copied such that they are a sort of generic standard
  - Technically female-only, mate with square post male **pin headers**

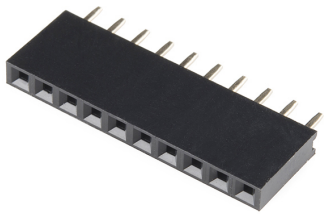


- Normally 0.1” or 2.54mm pitch
- Crimp is mechanically connected to a wire using a specialized crimping tool
- Crimp is then inserted into a plastic housing
- Housing may be 1 to 10 pins wide, single- or double-row





- Generic male pin versions do exist
- Mate with female Dupont connectors or female headers



CRIMPING

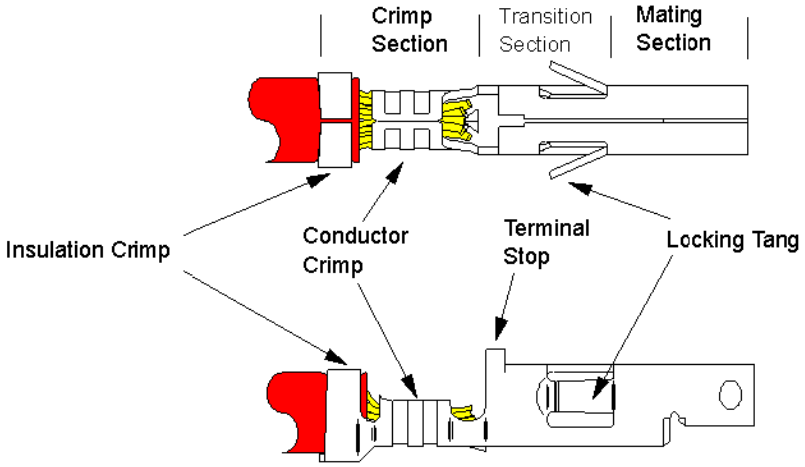
Inexpensive generic crimp tool

ratcheting crimp tool

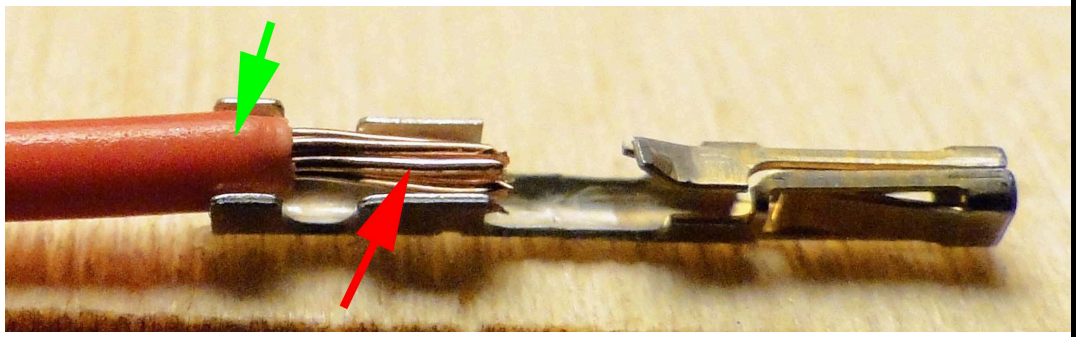
Historic “official” crimp tool (~$1200)

molex

ANATOMY OF A TERMINAL



Insulation crimp

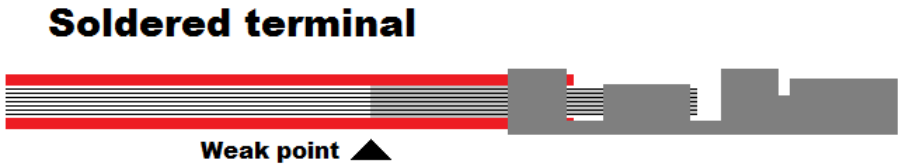


Conductor crimp

CRIMP CONNECTORS

- **Why crimp?**
  - When well-done, makes a very robust connection

  - Vs. a solder connection that is brittle and can break when flexed

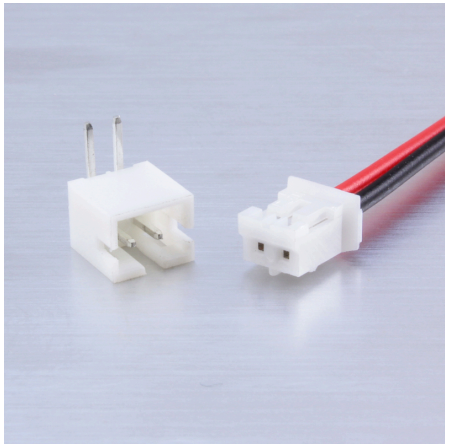


  - When done right, makes a very robust connection
  - Customizable cable length vs a pre-fabricated cable assembly
  - For small numbers of connectors, crimps are often very cost-effective:
    - in quantities we purchase, a crimp connector costs about $0.04 and a 3-pin housing about $0.10
    - BUT an 8-pin housing costs about $0.40, so a double-ended 8-pin crimp cable costs about $1.50 in connectors, plus wire.
    - Worth thinking twice if you are making large quantities of something

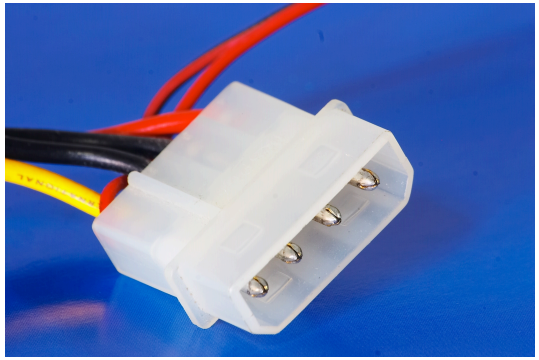
CRIMP CONNECTORS

- **Why not crimp?**
  - Can be time consuming
  - Not always most cost effective
  - Not very robust connection, cables can get inadvertantly pulled out easily

- **BEWARE! There are many types of crimps out there. Not all are compatible.**
  - Molex
  - JST



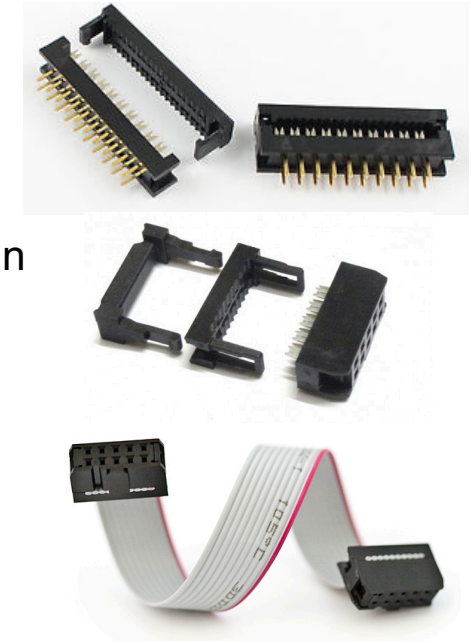
JST



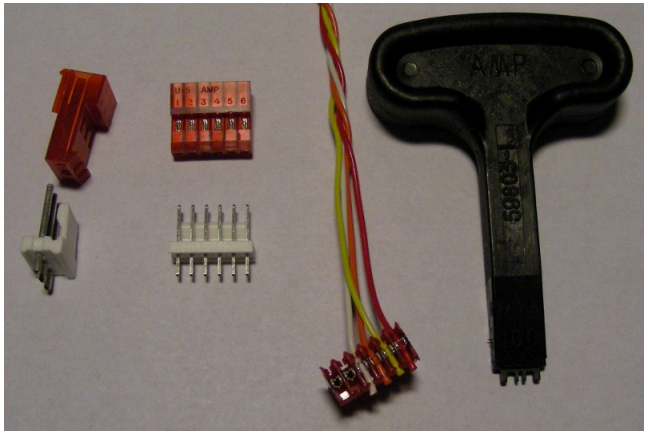
Molex

IDC CONNECTORS

- **Ribbon cable IDC connectors**
  - Insulation Displacement Connector
  - Normally double-row, 0.1” pitch
  - Most often female, mating with double-row male pin header
  - 10, 14, 16, 20 pin are common
  - Very secure
  - Good for short runs of many connectors
  - Not shielded, bad for long runs and expensive for long runs

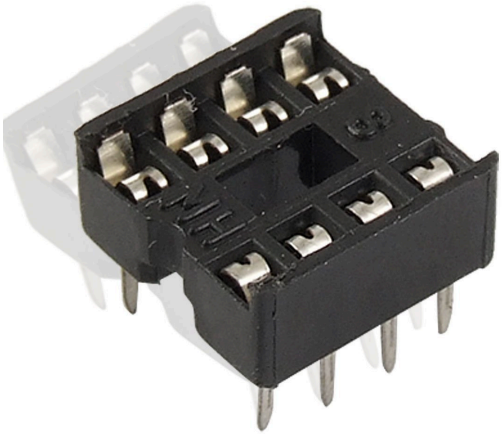


- **MTA Connectors**
  - MTA-100 are 0.1” pitch
  - Male header has a locking ramp
  - Female headers are intended for a specific wire gauge, often color-coded
  - Our white ones are for 24awg
  - T-handle (cheap) tool or $400 (fast) mechanical tool
  - Generally for board-mount only
  - Not great for breadboards

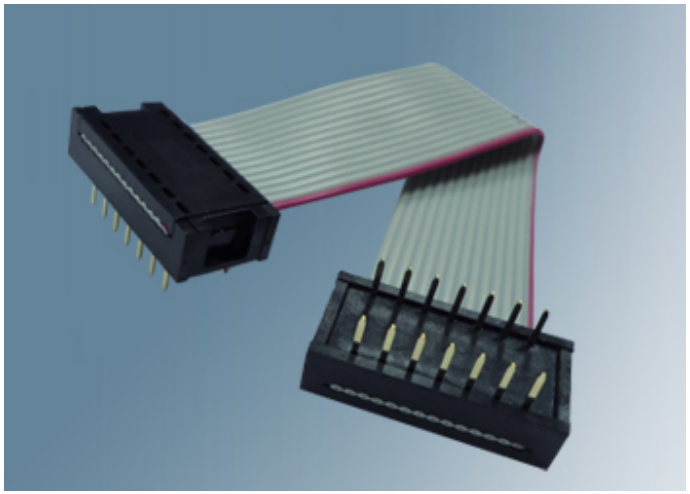


IDC CONNECTORS

- **IDC DIP Plug**
  - Cable-mount connector that copies the pin configuration of a DIP IC
  - DIP: Dual Inline Package
  - Standard width/spacing for integrated circuits
  - Commonly: 8, 14, 16, 18, 28 pins
  - Pins often not long enough for good connection with breadboard
  - Better for plugging into an IC Socket

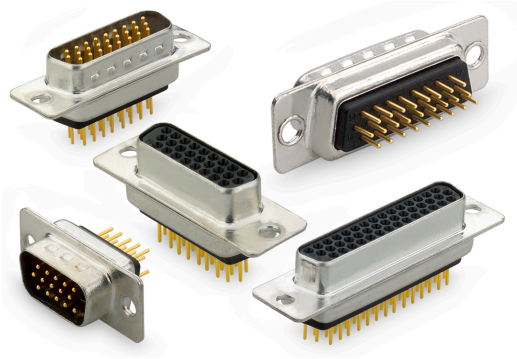


IC Socket

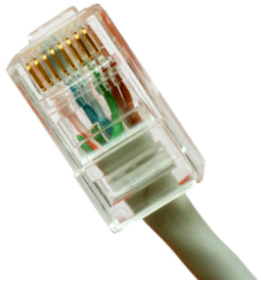


IDC DIP Plug

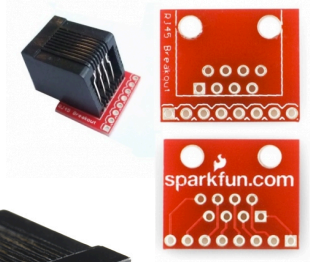
OTHER CONNECTORS



- **D-SUB**
  - D-subminiature; name from the D-shape of the connector
  - Most often cable-mount and panel-mount
  - Difficult to make, requires specialized tools
  - Good for high-density connections



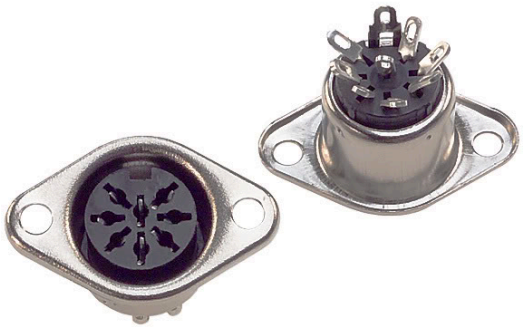
- **“Modular Plug and Jack” or “Registered Jack” (RJ)**
  - Includes telephone jacks and plugs (RJ11, RJ12, RJ25)
  - Ethernet jacks and plugs (RJ45)
  - Specified by the number of positions (P) and the number of actual connectors (C), where C<=P
  - E.g. Ethernet (RJ45) is 8P8C
  - Telephones can be 6P6C, 6P4C, 6P2C. Telephone handset plugs are 4P4C
  - Cable assemblies are very inexpensive
  - Very efficient way to connect 8 signals
  - Sparkfun Breakout Board gives convenient access to 8 pins ($0.75)
  - But really only for board-mount application



OTHER CONNECTORS

- **DIN Connector**
  - “Deutsches Institut für Normung”
  - MIDI connector is a 5-pin DIN connector
  - 3 to 8 pin and even a 13-pin standard connector
  - Female panel-mount connectors are very inexpensive

5-pin DIN Female right-angle board mount



6-pin DIN Female Panel-mount

5-pin DIN Male (MIDI)

GENERAL INTERCONNECT STRATEGIES

- **Arduino -> cable assembly with male pins**
  - Pros:
    - Less connections to debug
    - May be cheap
    - Customizable cables
    - Less space, no breadboard
  - Cons:
    - Not very secure
    - May be time consuming if using crimps
    - No room for external circuitry
    - What to do about multiple power connections?

GENERAL INTERCONNECT STRATEGIES

- **Arduino -> breadboard -> cable assembly with male pins**
  - Pros
    - No strain on Arduino-breadboard connection
    - Breadboard connection may be more secure
    - Modular
    - Power rails available
    - Room for external circuitry/components
  - Cons
    - Extra connections to debug
    - External circuitry/components may not be secure
    - Still potentially time consuming to make crimps
    - Uses more space

GENERAL INTERCONNECT STRATEGIES

- **Arduino -> Panel Mount Connector -> Cable assembly**
  - Pros
    - No strain on Arduino connections
    - No breadboard, less space
    - Panel Mount connectors are most likely secure
    - Can use preassembled cables, save time
  - Cons
    - Need to make/buy an enclosure
    - Often need soldering expertise for panel mount connector
    - Still possibly power rail concerns
    - What to do about external circuitry/components?

GENERAL INTERCONNECT STRATEGIES

- **Arduino -> Perfboard/PCB -> Board-mount Cable assembly**
  - Pros
    - No strain on Arduino connections
    - Perfboard can be custom sized
    - Board-mount connectors are most likely secure
    - MTA cables are very fast to make
    - Secure external circuitry
    - Customize cables
  - Cons
    - Not good for long connections to external devices
    - Often need soldering expertise for panel mount connector
    - Still possibly power rail concerns

GENERAL INTERCONNECT STRATEGIES

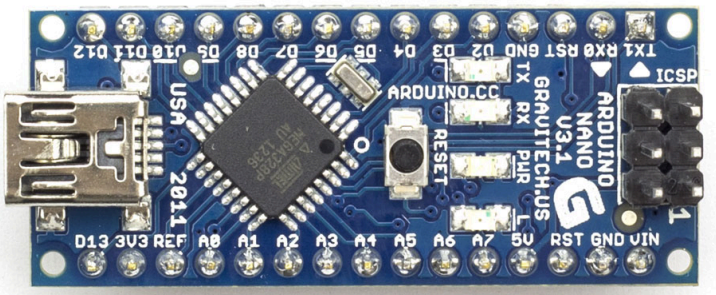
- **Arduino -> Perfboard/PCB -> Board-mount connector**
  - Pros
    - No strain on Arduino connections
    - Perfboard can be custom sized
    - Board-mount connectors are most likely secure
    - Secure external circuitry
    - Use standard cables, save time
  - Cons
    - Difficult to design / need the connector to be accessible
    - Board mount connectors not always standard 0.1” pitch

OTHER RESOURCES

- Molex Good Crimps and How to Recognize Them: http://www.molex.com/tnotes/crimp.html
- JST Check-points for Good Crimping: https://cdn-shop.adafruit.com/datasheets/JST_CrinpChart+(English).pdf

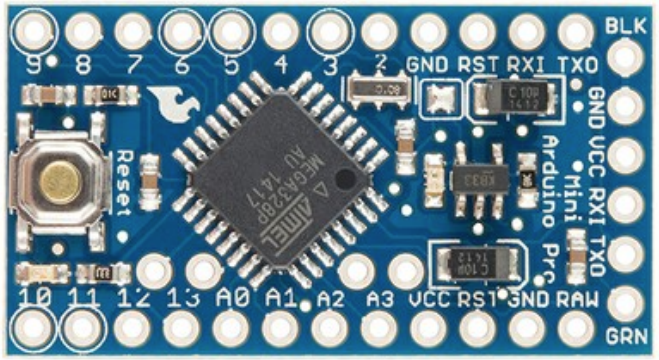
OTHER WAYS OF DOING SIMILAR STUFF

ARDUINO NANO



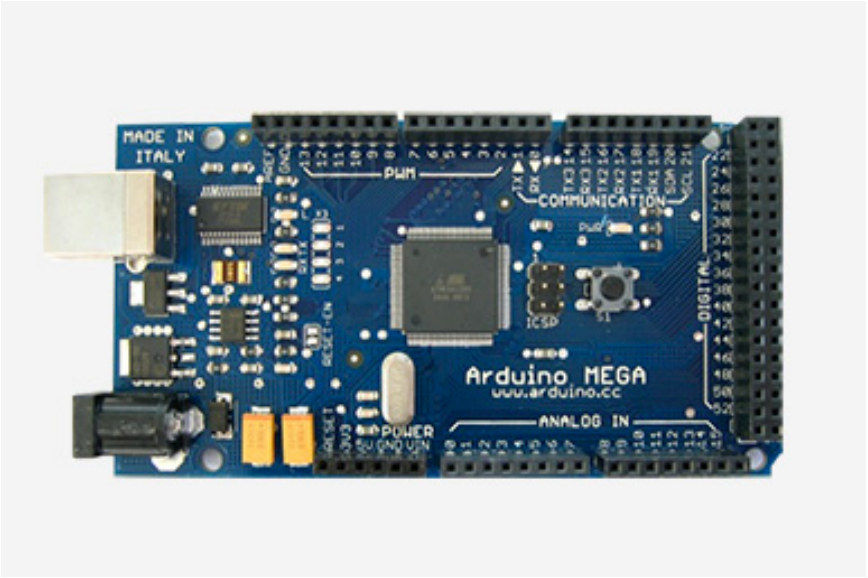
- Basically same functionality as Arduino UNO
- Breadboardable, smaller package
- No power jack, but pin available for external in

ARDUINO PRO MINI



- Similar functionality as Arduino UNO
- No USB
- Programmed with external programmer (or an Arduino UNO
- Only $10
- Good for permanent installations, devices where you don’t need USB serial I/O

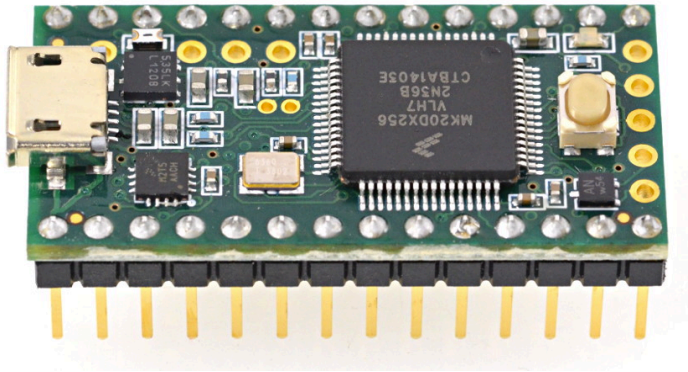
ARDUINO MEGA



- Similar functionality as Arduino UNO
- More Digital, analog, PWM I/O

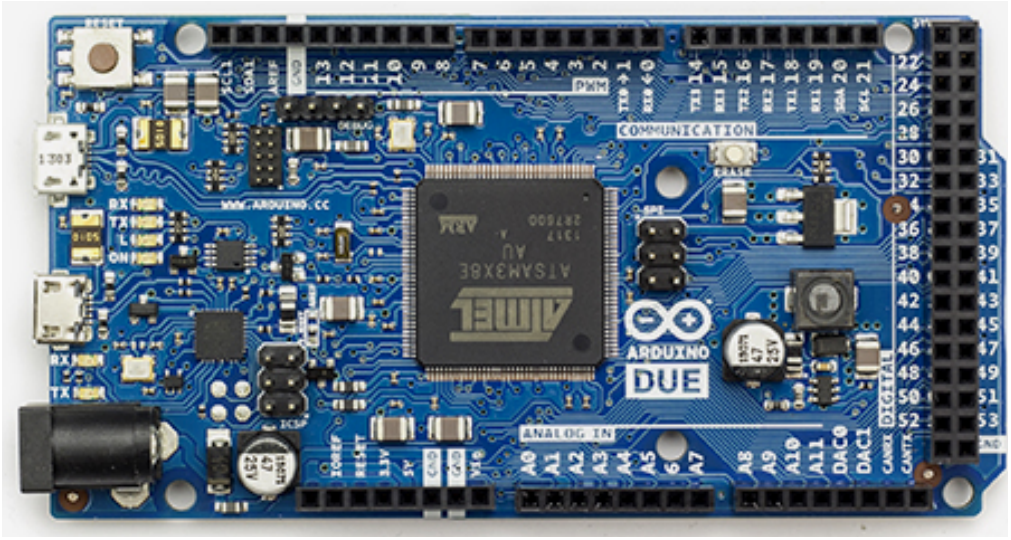
TEENSY

https://www.pjrc.com/teensy/



- 32-bit ARM processor
- $20
- Compatible with Arduino IDE
- Native USB 1.0 – can appear as a keyboard, mouse, MIDI device, serial device, etc.
- Simple $15 Audio board add-on for basic audio functionality

ARDUINO DUE



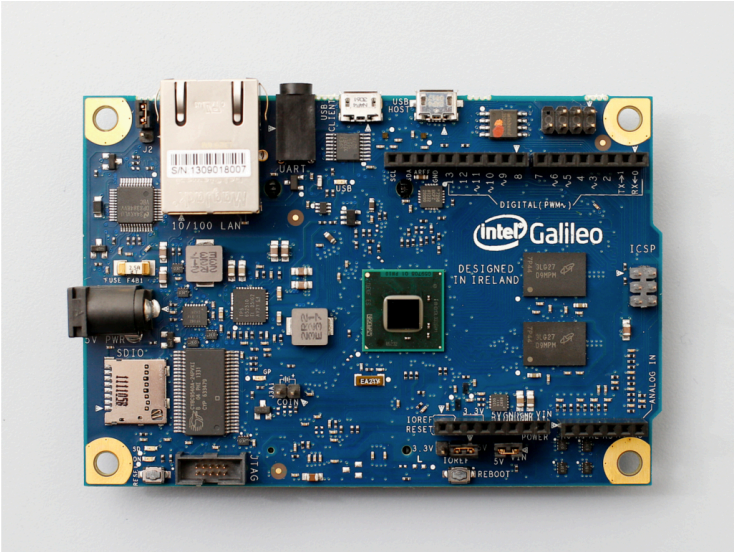
- Like a Mega, but with a 32-bit ARM processor
- Still program it in the Arduino IDE
- Allows processing of 32-bit data

ARDUINO YUN

https://www.arduino.cc/en/Main/ArduinoBoardYun3
2-bit ARM processor

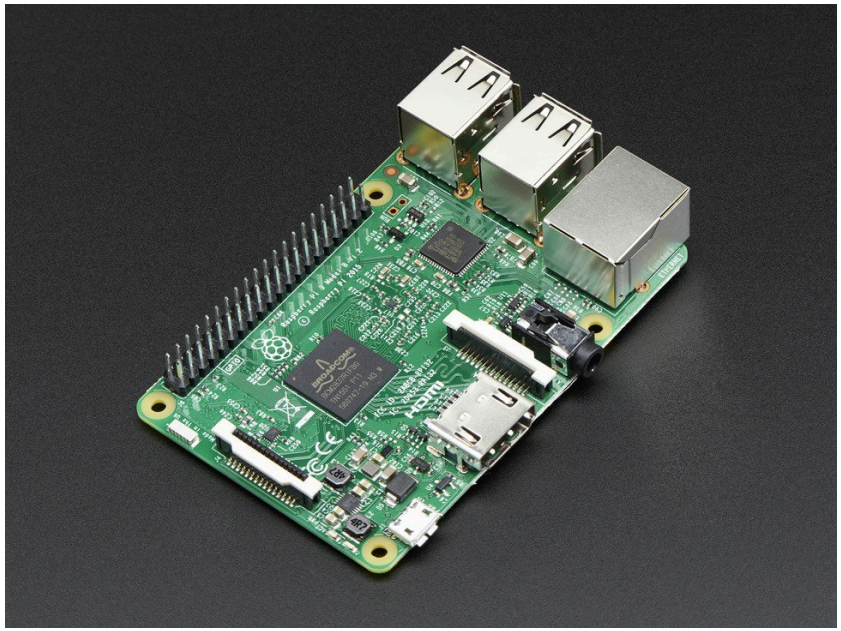
- $70
- ATMEGA + ARM 32 bit linux
- Ethernet, wifi
- Basically an Arduino + a low-power Linux computer on the same board
- Serial communication between the 2

INTEL GALILEO



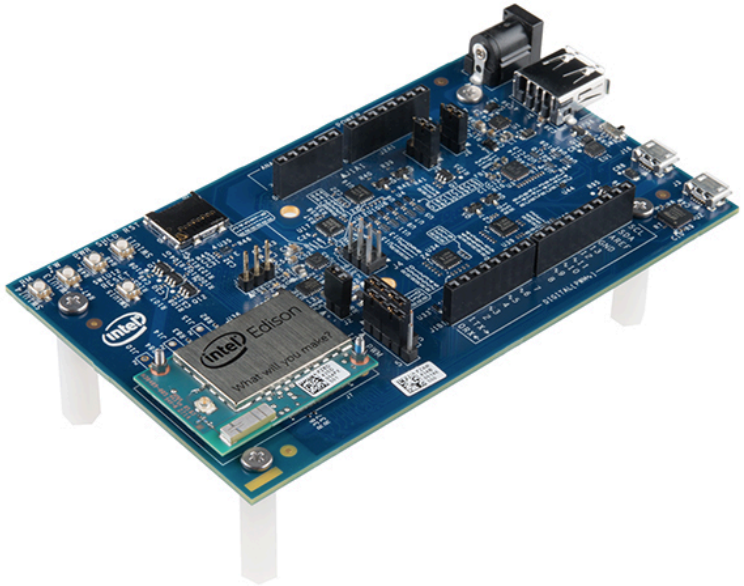
- https://www.arduino.cc/en/ArduinoCertified/IntelGalileo
- Runs linux
- $60
- Arduino pinout compatible, programmable with arduino IDE
- No built-in audio, can use a USB sound card with ALSA

RASPBERRY PI 3



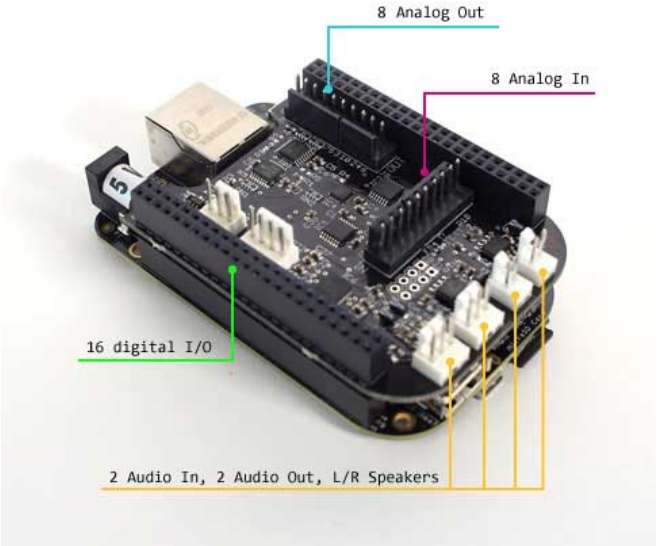
- 64-bit ARM processor
- Runs linux
- $40
- Ethernet, USB, bluetooth, HDMI…
- No analog inputs
- Audio out; latency problems for real-time

INTEL EDISON + ARDUINO



- https://www.sparkfun.com/products/13097
- $100
- Dual-core cpu with wifi, bluetooth
- Block/Shield architecture: add on functionality

BELA



http://bela.io/

Beaglebone Black Cape

- ~$100
- Runs embedded linux
- 8 analog I/O, 16 digital I/O, stereo audio I/O
- Ultra-low latency; audio and sensor data handled on same device