

CSE423: Embedded Systems

Summer, 2020

Introduction to Embedded Systems





Today's Lecture

- What is the embedded system?
- Characteristic and application of Embedded Systems
- Use of micro-processor in embedded systems



What is Embedded Systems?

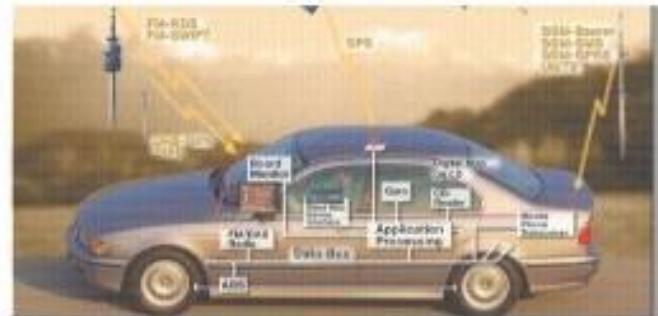
- **Definition:** An Embedded system is a microprocessor-based computer hardware system with software that is designed to perform a dedicated function, either as an independent system or as a part of a large system. At the core is an integrated circuit designed to carry out computation for real-time operations.

Embedded Systems (Contd.)



- ▶ We are surrounded by Embedded Systems.

- ▶ Cell Phones
 - ▶ Automatic Washing Machines.
 - ▶ Traffic Signals with Timers.
 - ▶ Automobile Electronics.



- ▶ Find a system that contains no electronic system.
 - ▶ How can a electronic system improve the functionality/efficiency of that system.
 - ▶ Custom design an embedded system for the same.

Embedded Systems (contd.)

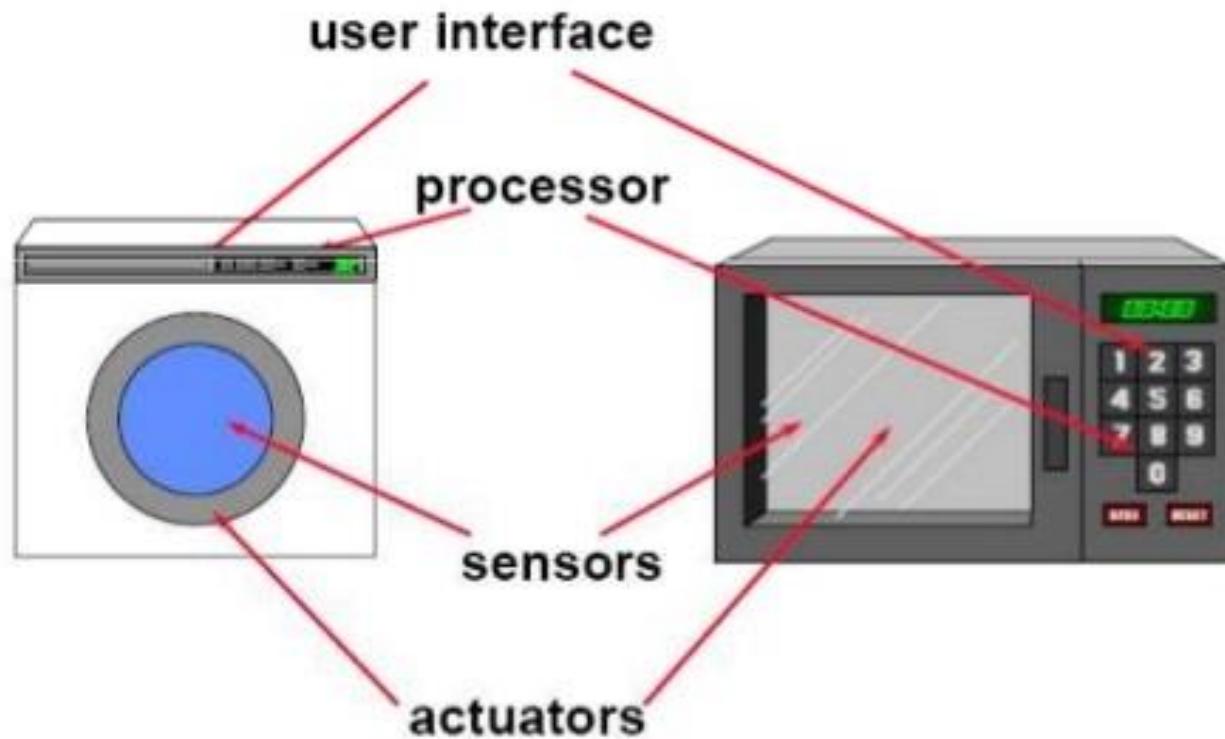
- ▶ Embedded system means the processor is embedded into that application.
- ▶ An embedded product uses a microprocessor or microcontroller to do one task only.
- ▶ In an embedded system, there is only one application software that is typically burned into ROM.
- ▶ Example : printer, keyboard, video game player



Embedded Systems (contd.)



Consumer electronics, for example MP3 Audio, digital camera, home electronics,



Characteristics of Embedded Systems

- The system must meet the following characteristics:
 - Performance
 - Cost/size
 - Real time requirements
 - Power consumption
 - Reliability



Characteristics of Embedded Systems

- ▶ Must be *efficient*:
 - *Energy* efficient
 - *Code-size* efficient (especially for systems on a chip)
 - *Run-time* efficient
 - *Weight* efficient
 - *Cost* efficient
- ▶ *Dedicated* towards a certain *application*: Knowledge about behavior at design time can be used to minimize resources and to maximize robustness.
- ▶ *Dedicated* user *interface* (no mouse, keyboard and screen).



Characteristics of Embedded Systems

- ▶ Many ES must meet *real-time constraints*:
 - A real-time system must *react to stimuli* from the controlled object (or the operator) within the time interval dictated by the environment.
 - For real-time systems, right answers arriving too late (or even too early) are wrong.

„A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe“ [Kopetz, 1997].

- All other time-constraints are called soft.
- A *guaranteed system response* has to be explained without statistical arguments.



Characteristics of Embedded Systems

- ▶ Frequently **connected to physical environment** through sensors and actuators,
- ▶ **Hybrid systems** (analog + digital parts).
- ▶ Typically, ES are **reactive systems**:

„A reactive system is one which is in continual interaction with its environment and executes at a pace determined by that environment“ [Bergé, 1995]

- Behavior depends on input and current state.
 - ☞ automata model often appropriate,



Characteristics of Embedded Systems

► Embedded Systems

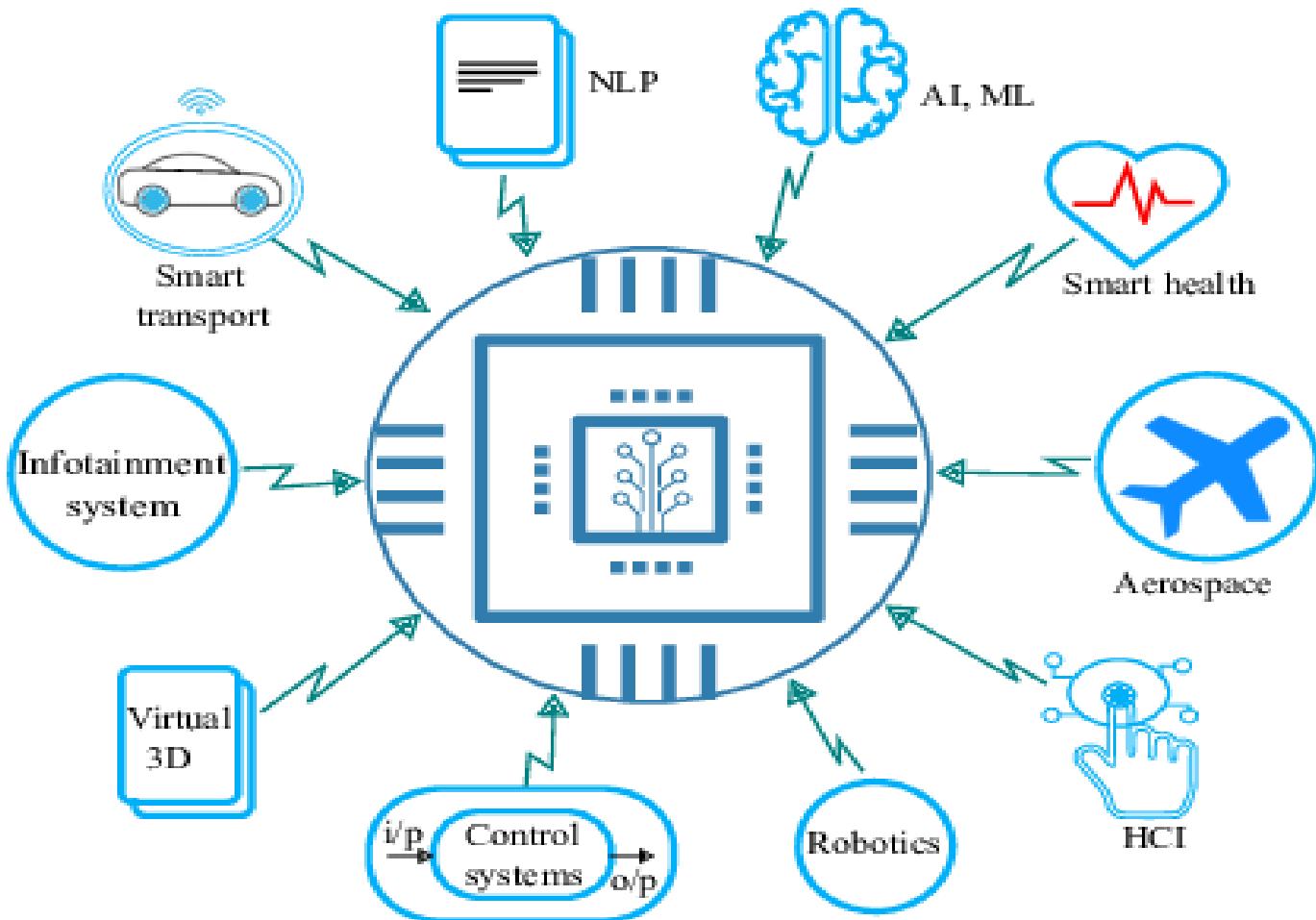
- Few applications that are known at design-time.
- Not programmable by end user.
- Fixed run-time requirements (additional computing power not useful).
- Criteria:
 - cost
 - power consumption
 - predictability

► General Purpose Computing

- Broad class of applications.
- Programmable by end user.
- Faster is better.
- Criteria:
 - cost
 - average speed



Current Domain of Embedded Systems



Components of Embedded Systems

- Analog Components
 - Sensors, Actuators, Controllers, ...
- Digital Components
 - Processor, Coprocessors
 - Memories
 - Controllers, Buses
 - Application Specific Integrated Circuits (ASIC)
- Converters – A2D, D2A, ...
- Software
 - Application Programs
 - Exception Handlers



Microprocessors in Embedded Systems?

- Alternatives: field-programmable gate arrays (FPGAs), custom logic, application specific integrated circuit (ASIC), etc.
- Microprocessors are often very efficient: can use same logic to perform many different functions.
- Microprocessors simplify the design of families of products.





Why Use Microprocessors?

- Two factors that work together to make microprocessor-based designs fast
 - First, microprocessors execute programs very efficiently. While there is overhead that must be paid for interpreting instructions, it can often be hidden by clever utilization of parallelism within the CPU
 - Second, microprocessor manufacturers spend a great deal of money to make their CPUs run very fast.



Why Use Microprocessors?

- Performance
 - Microprocessors use much more logic to implement a function than does custom logic.
 - But microprocessors are often at least as fast:
 - heavily pipelined;
 - large design teams;
 - aggressive VLSI technology.
- Power consumption
 - Custom logic is a clear winner for low power devices.
 - Modern microprocessors offer features to help control power consumption.
 - Software design techniques can help reduce power consumption.
- Heterogeneous systems: some custom logic for well-defined functions, CPUs+ software for everything else.

Microprocessor Varieties



- **Microcontroller:** includes I/O devices, on-board memory.
- **Digital signal processor (DSP):** microprocessor optimized for digital signal processing.
- Typical embedded word sizes: 8-bit, 16-bit, 32-bit.



Microprocessor Varieties

- 4-bit, 8-bit, 16-bit, 32-bit :
 - 8-bit processor : more than 3 billion new chips per year
 - 32-bit microprocessors : **PowerPC, 68k, MIPS, and ARM** chips.
 - **ARM-based chips** alone do about **triple the volume** that Intel and AMD peddle to PC makers.
- Most (**98%** or so) 32-bit processors are used in embedded systems, not PCs.
- RISC-type processor owns most of the overall embedded market [MPF: 2002].

Platforms



- Embedded computing platform: hardware architecture + associated software.
- Many platforms are multiprocessors.
- Examples:
 - Single-chip multiprocessors for cell phone baseband.
 - Automotive network + processors.



The physics of software

- Computing is a physical act.
 - Software doesn't do anything without hardware.
- Executing software consumes energy, requires time.
- To understand the dynamics of software (time, energy), we need to characterize the platform on which the software runs.



Challenges in Embedded Computing System Design

- How much hardware do we need?
- How do we meet deadlines?
- How do we minimize power consumption?
- How do we design for upgradability?
- Does it really work?

What does “Performance” mean?



- In general-purpose computing, performance often means average-case, may not be well-defined.
- In real-time systems, performance means meeting deadlines.
 - Missing the deadline by even a little is bad.
 - Finishing ahead of the deadline may not help.



Characterizing performance

- We need to analyze the system at several levels of abstraction to understand performance:
 - CPU: microprocessor architecture.
 - Platform: bus, I/O devices.
 - Program: implementation, structure.
 - Task: multitasking, interaction between tasks.
 - Multiprocessor: interaction between processors.

Characteristics of Embedded Systems

- Very high performance, sophisticated functionality
 - Vision + compression + speech + networking all on the same platform.
- Multiple task, heterogeneous.
- Real-time.
- Often low power.
- Low manufacturing cost..
- Highly reliable.
 - I reboot my piano every 4 months, my PC every day.
- Designed to tight deadlines by small teams.



Functional Complexity



- Often have to run sophisticated algorithms or multiple algorithms.
 - Cell phone, laser printer.
- Often provide sophisticated user interfaces.



Design methodologies

- A procedure for designing a system.
- Understanding your methodology helps you ensure you didn't skip anything.
- Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:
 - help automate methodology steps;
 - keep track of the methodology itself.

Levels of abstraction



requirements

specification

architecture

component
design

system
integration

Our requirements form



name

purpose

inputs

outputs

functions

performance

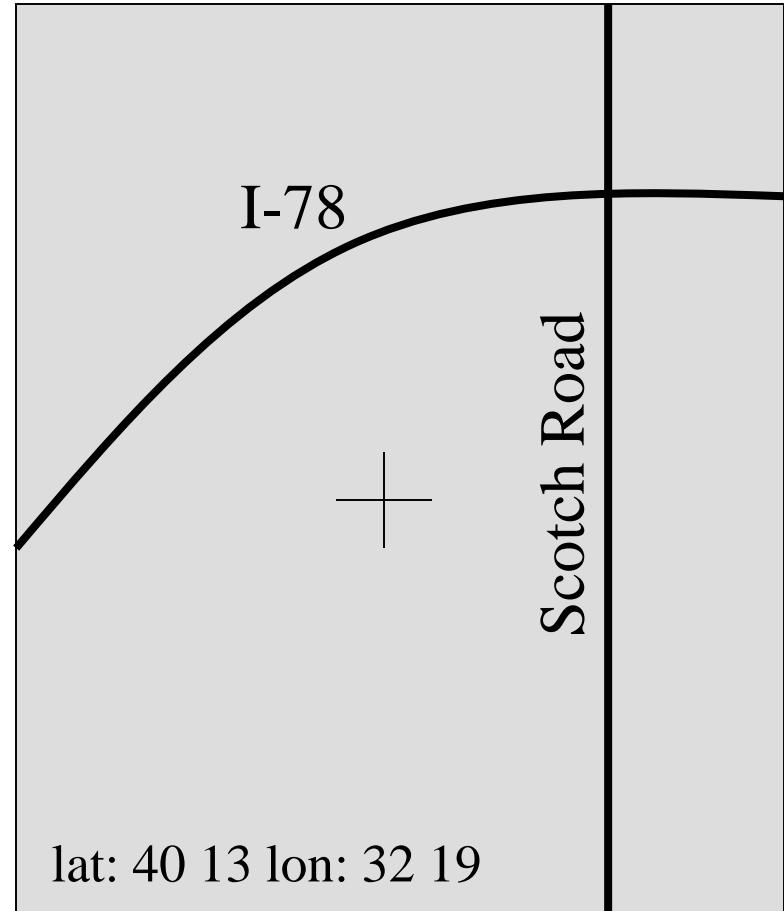
manufacturing cost

power

physical size/weight

Example: GPS moving map requirements

- Moving map obtains position from GPS, paints map from local database.





GPS moving map needs

- Functionality**: For automotive use. Show major roads and landmarks.
- User interface**: At least 400 x 600 pixel screen. Three buttons max. Pop-up menu.
- Performance**: Map should scroll smoothly. No more than 1 sec power-up. Lock onto GPS within 15 seconds.
- Cost**: \$120 street price = approx. \$30 cost of goods sold.

GPS moving map needs, cont'd.



- Physical size/weight:** Should fit in hand.
- Power consumption:** Should run for 8 hours on four AA batteries.

GPS moving map requirements form

name	GPS moving map
purpose	consumer-grade moving map for driving
inputs	power button, two control buttons
outputs	back-lit LCD 400 X 600
functions	5-receiver GPS; three resolutions; displays current lat/lon updates screen within 0.25 sec of movement
performance	
manufacturing cost	\$100 cost-of-goods- sold
power	100 mW
physical size/weight	no more than 2: X 6:, 12 oz.





Specification

- A more precise description of the system:
 - should not imply a particular architecture;
 - provides input to the architecture design process.
- May include functional and non-functional elements.
- May be executable or may be in mathematical form for proofs.



GPS specification

□ Should include:

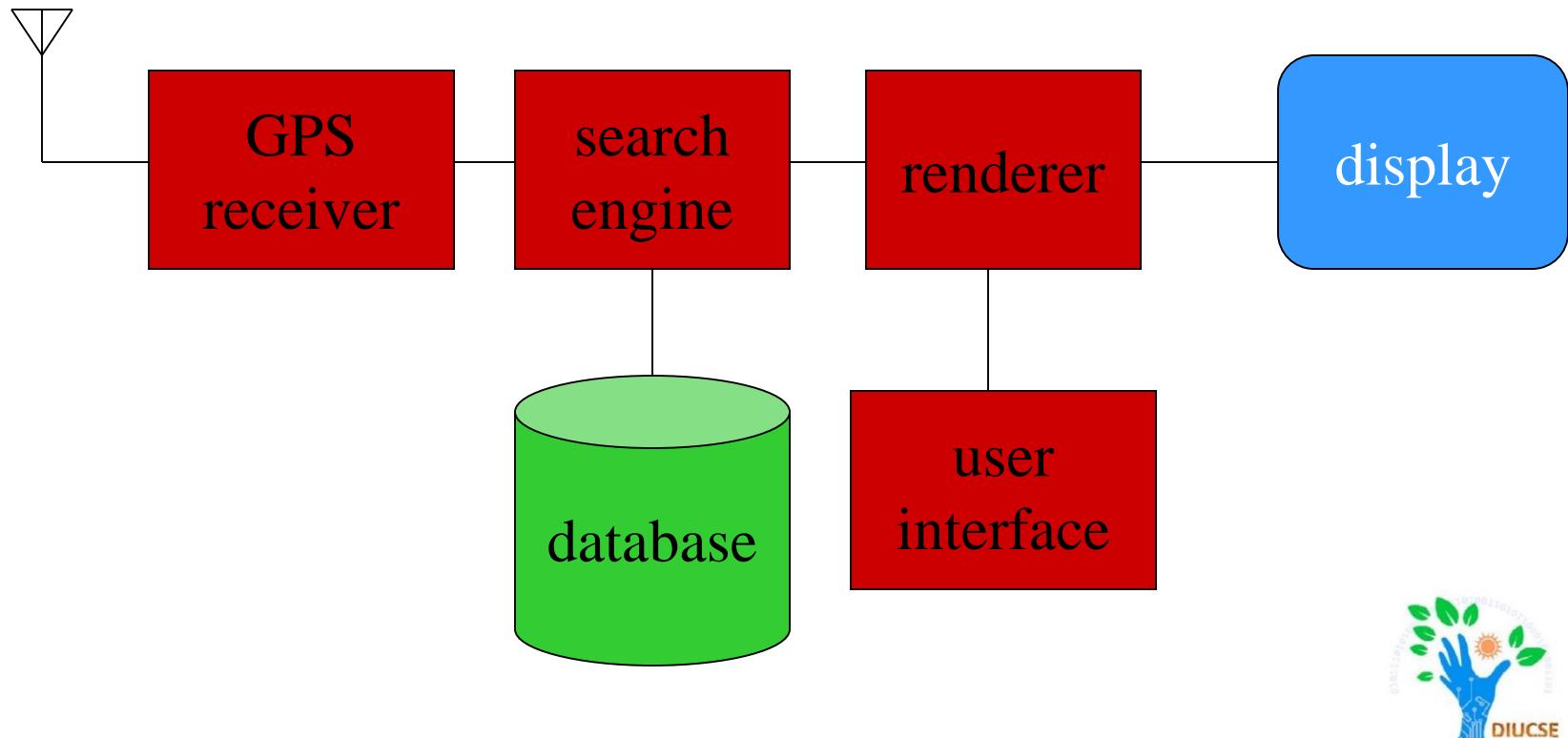
- What is received from GPS;
- map data;
- user interface;
- operations required to satisfy user requests;
- background operations needed to keep the system running.

Architecture design

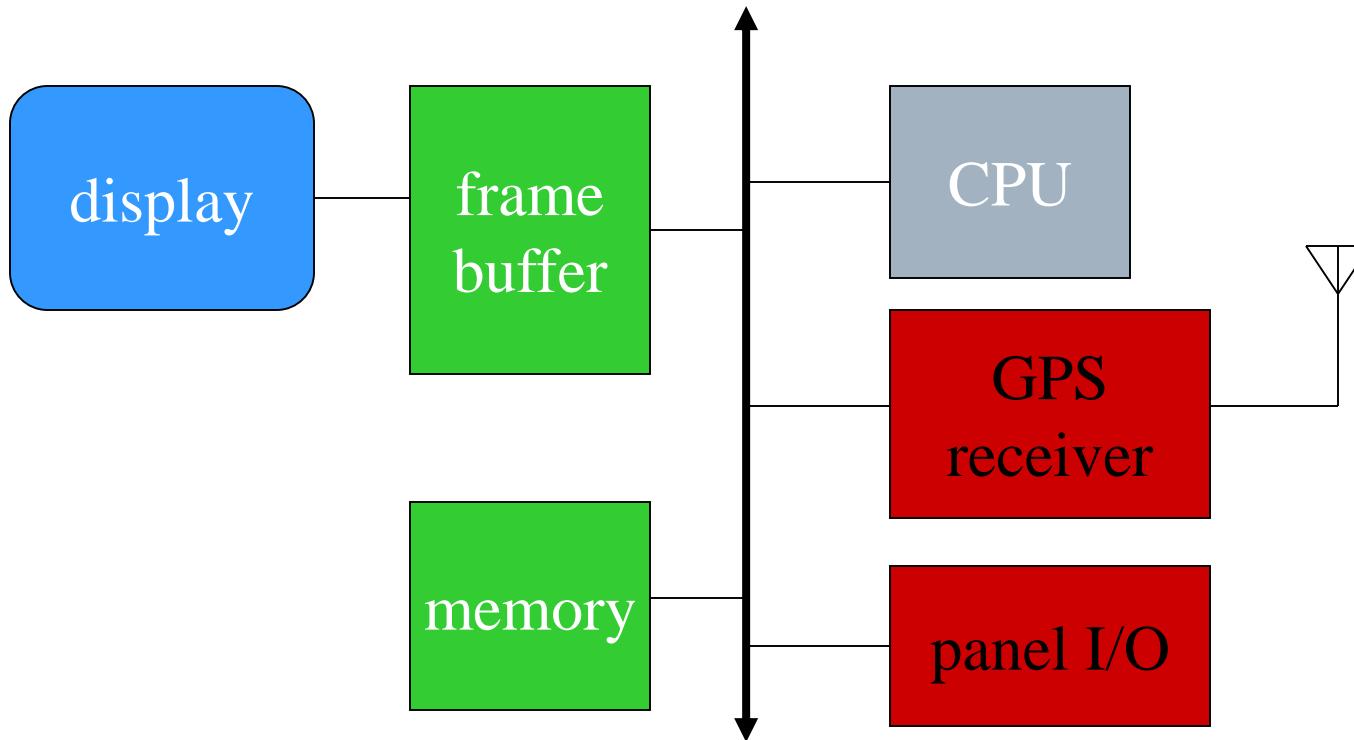


- What major components go satisfying the specification?
- Hardware components:
 - CPUs, peripherals, etc.
- Software components:
 - major programs and their operations.
- Must take into account functional and non-functional specifications.

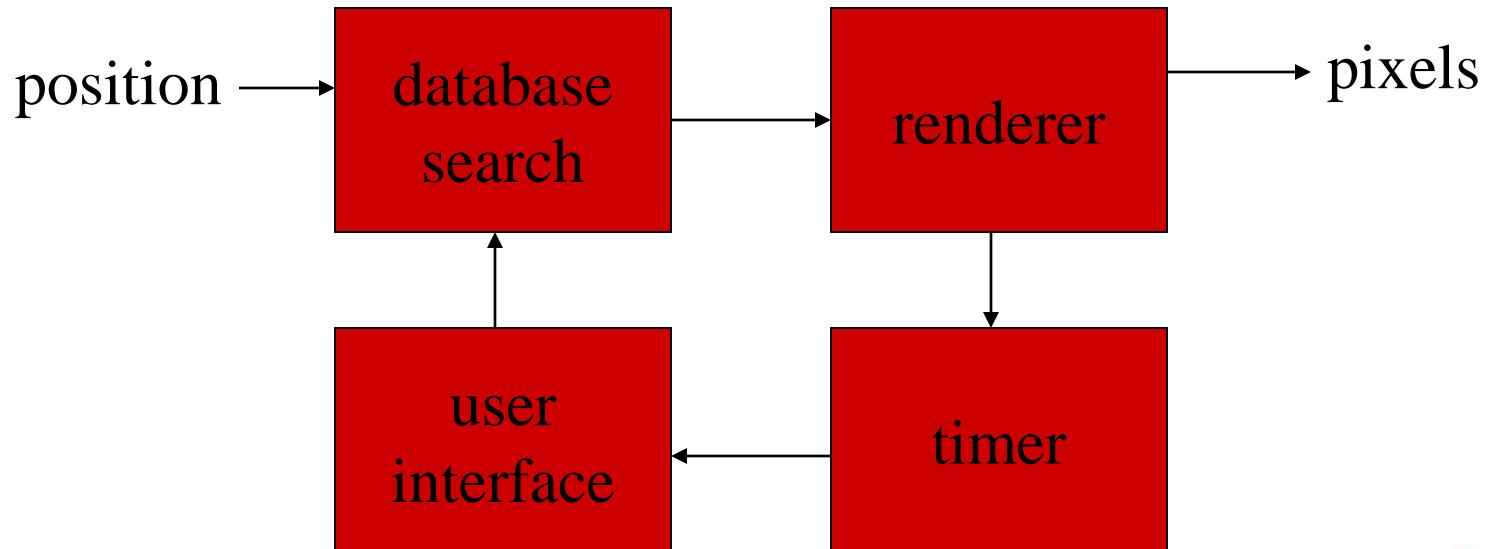
GPS moving map block diagram



GPS moving map hardware architecture



GPS moving map software architecture



Designing hardware and software components

- Must spend time architecting the system before you start coding.
- Some components are ready-made, some can be modified from existing designs, others must be designed from scratch.





System integration

- Put together the components.
 - Many bugs appear only at this stage.
- Have a plan for integrating components to uncover bugs quickly, test as much functionality as early as possible.



Summary

- Embedded computers are all around us.
 - Many systems have complex embedded hardware and software.
- Embedded systems pose many design challenges: design time, deadlines, power, etc.
- Design methodologies help us manage the design process.

CSE423: Embedded System

Summer-2020

Introduction to micro-controller and other basic components



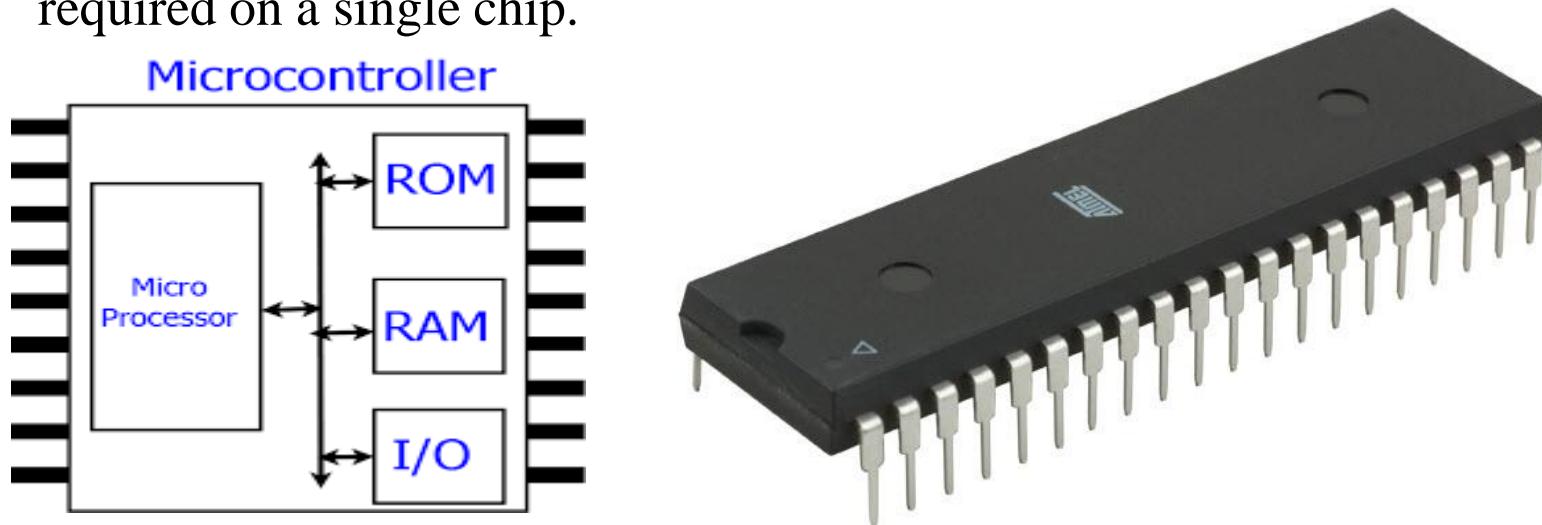
Lecture Objective



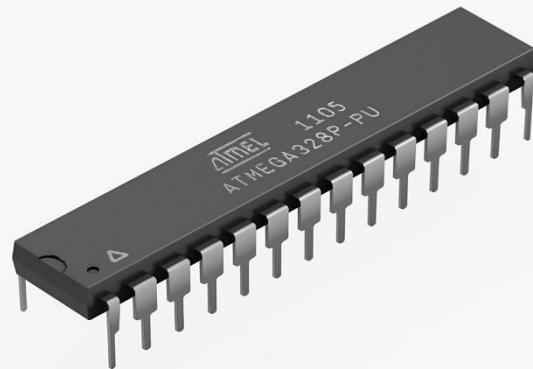
- At the end of this course, students should be able to:
 - *Interface a microcontroller to a variety of analog and digital inputs and output devices.*
 - *Program a microcontroller to implement a closed-loop automatic control.*
 - *Write and Troubleshoot programming language code for a microcontroller.*
 - *Analyze a problem to determine appropriate microcontroller use.*

What is Microcontroller?

- A microcontroller is an **integrated chip** that is often part of an embedded system.
- The microcontroller includes a **CPU, RAM, ROM, I/O ports** and **timers** like a standard computer, but because they are designed to execute only **a single specific task** to control a single system, they are much *smaller and simplified* so that they can include all the functions required on a single chip.



Microcontroller (Atmel ATmega 328P)



ATmega328P is a high performance yet low power consumption 8-bit AVR microcontroller that's able to achieve the most single clock cycle execution of 131 powerful instructions thanks to its advanced **RISC** architecture. It can commonly be found as a processor in Arduino boards such as [Arduino Fio](#) and [Arduino Uno](#).

Microcontroller (Atmel ATmega 328P)



Features:

High endurance non-volatile memory segments

- In system self-programmable flash program memory
- Programming Lock for software security

Peripheral features

- Two 8-bit Timer/Counter with separate prescaler, compare mode.
- One 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
- Temperature measurement
- Programmable serial USART and watchdog timer with separate on-chip oscillator.

Microcontroller (Atmel ATmega 328P)



Unique features compared to other microcontrollers (ARM, 8051, PIC):

- Power-on reset and programmable brown-out detection
- Internal calibrated oscillator
- External and Internal interrupt sources
- Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby, and extended standby



Microcontroller (Atmel ATmega 328P)

Advantages and Disadvantages:

Advantages	Disadvantages
<ul style="list-style-type: none">Processors are simpler to use, with the usage of 8bit and 16bit instead of 32/64bit which are more complex.Readily usable without additional computing components with 32k bytes of onboard self-programmable flash program memory as well as 23 programmable I/O linesCode Efficient, all 31 registers are directly connected to the arithmetic logic unit (ALU), making it 10 times faster than conventional CISC microcontrollersOptimized for AVR enhanced RISC instruction set.	<ul style="list-style-type: none">Lacks performance compared to higher bit microcontrollers

Microcontroller (Atmel ATmega 328P)

□ Comparison of different micro-controllers

	 ATmega328P	 STM32	 MSP430
Brand	MicroChip	Cortex	Texas Instruments
Cost	Low	High	Low
Architecture	Advanced RISC architecture	Power Architecture technology designed for embedded applications	Older, von-Neumann architecture
Power Consumption	Low, more efficient power consumption	Medium, higher clock speed may result in higher consumption power	Low
Performance	Medium, lower bit but suitable for complex projects	High, fast processing speed, packs more power. Running 32 bit ARM processor core with sufficient RAM	Low, more suitable for only simple projects
Ease of Usage	Easy to use, 8 bit and high compatibility with Arduino boards	Complicated due to its nature of being a 32 bit microcontroller	Complex relative to Arduino boards

Required Acquisition



- Arduino Uno**
- USB A to B Cable**
- Arduino IDE**
- Breadboard**
- Jumper Wire**
- Different Circuit Elements**

Required Acquisition



- Textbook
- [Arduino Uno](#)
- USB A to B Cable
- Arduino IDE
- Breadboard
- Jumper Wire
- Different Circuit Elements



Price: 590 BDT

- For online order: <https://www.techshopbd.com/product-categories/boards/1253/arduino-uno-r3-china-techshop-bangladesh>

Required Acquisition

- Textbook
- Arduino Uno
- USB A to B Cable
- Arduino IDE
- Breadboard
- Jumper Wire
- Different Circuit Elements

Price: 80 BDT



- For online order: <https://www.techshopbd.com/product-categories/cable/206/usb-cable-a-to-b-techshop-bangladesh>

Required Acquisition



- Textbook**
- Arduino Uno**
- USB A to B Cable**
- Arduino IDE**
- Breadboard**
- Jumper Wire**
- Different Circuit Elements**

A screenshot of the Arduino IDE interface. The window title is "sketch_sep14a | Arduino 1.6.11 (Windows Store 1.6.11.0)". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for file operations. The main area shows the code for a sketch:

```
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

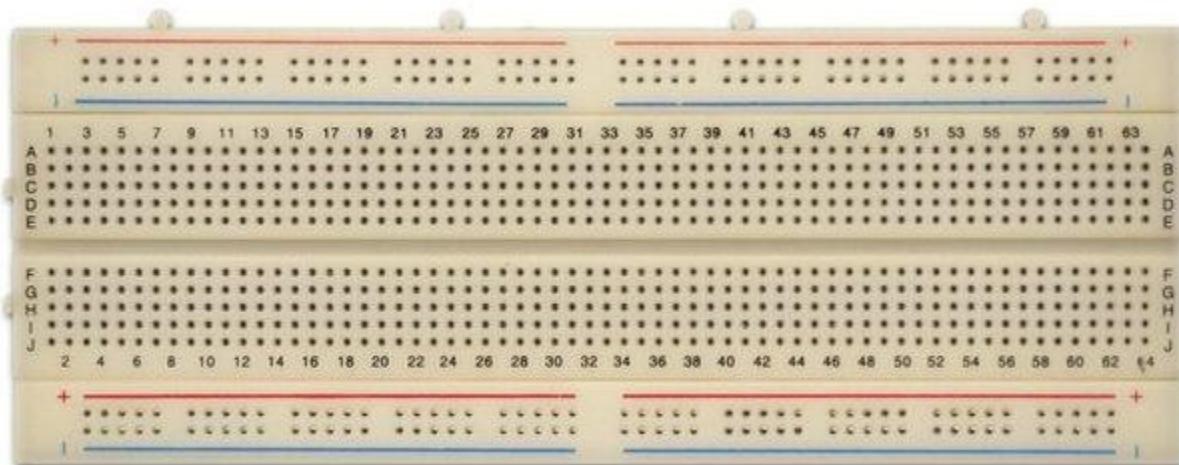
At the bottom of the IDE, it says "Arduino/Genuino Uno on COM1".

- **For download:** <https://www.arduino.cc/en/Main/Software>

Required Acquisition

- Textbook
- Arduino Uno
- USB A to B Cable
- Arduino IDE
- Breadboard
- Jumper Wire
- Different Circuit]

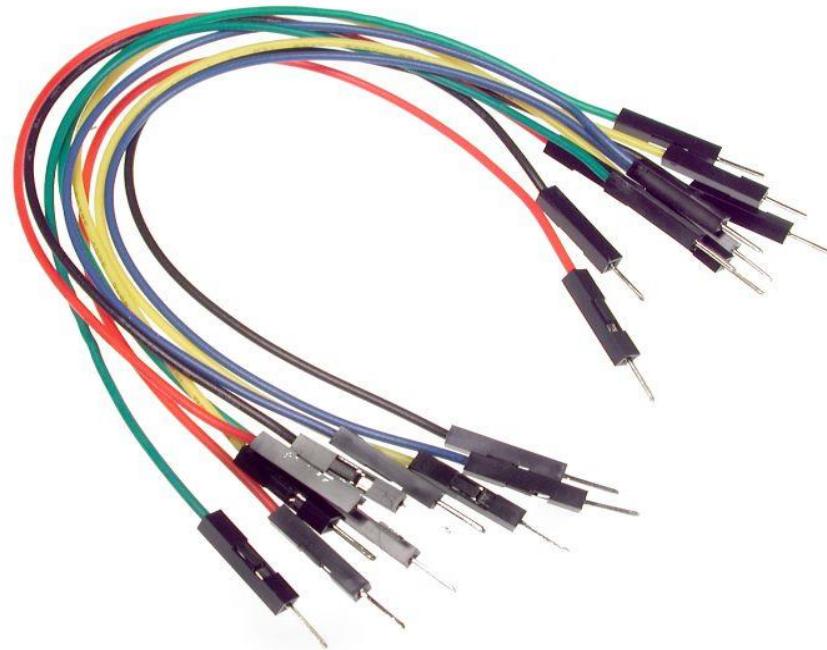
Price: 120 BDT



- **For online order:** <https://www.techshopbd.com/product-categories/project-board/231/breadboard-techshop-bangladesh>

Required Acquisition

- Textbook**
- Arduino Uno**
- USB A to B Cable**
- Arduino IDE**
- Breadboard**
- Jumper Wire**
- Different Circuit Elements**



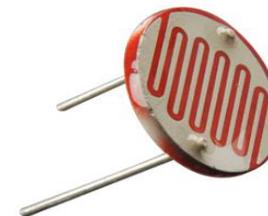
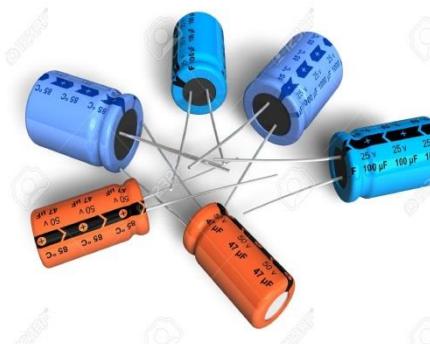
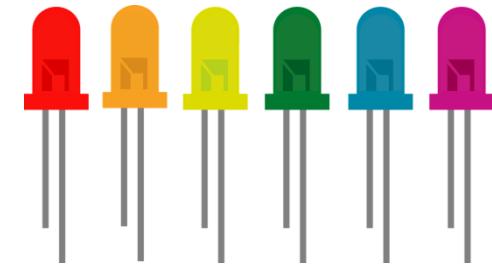
Price: 2.20 BDT pre piece

- **For online order:** <https://www.techshopbd.com/product-categories/cable/1083/male-to-male-jumper-wire-single-techshop-bangladesh>

Required Acquisition

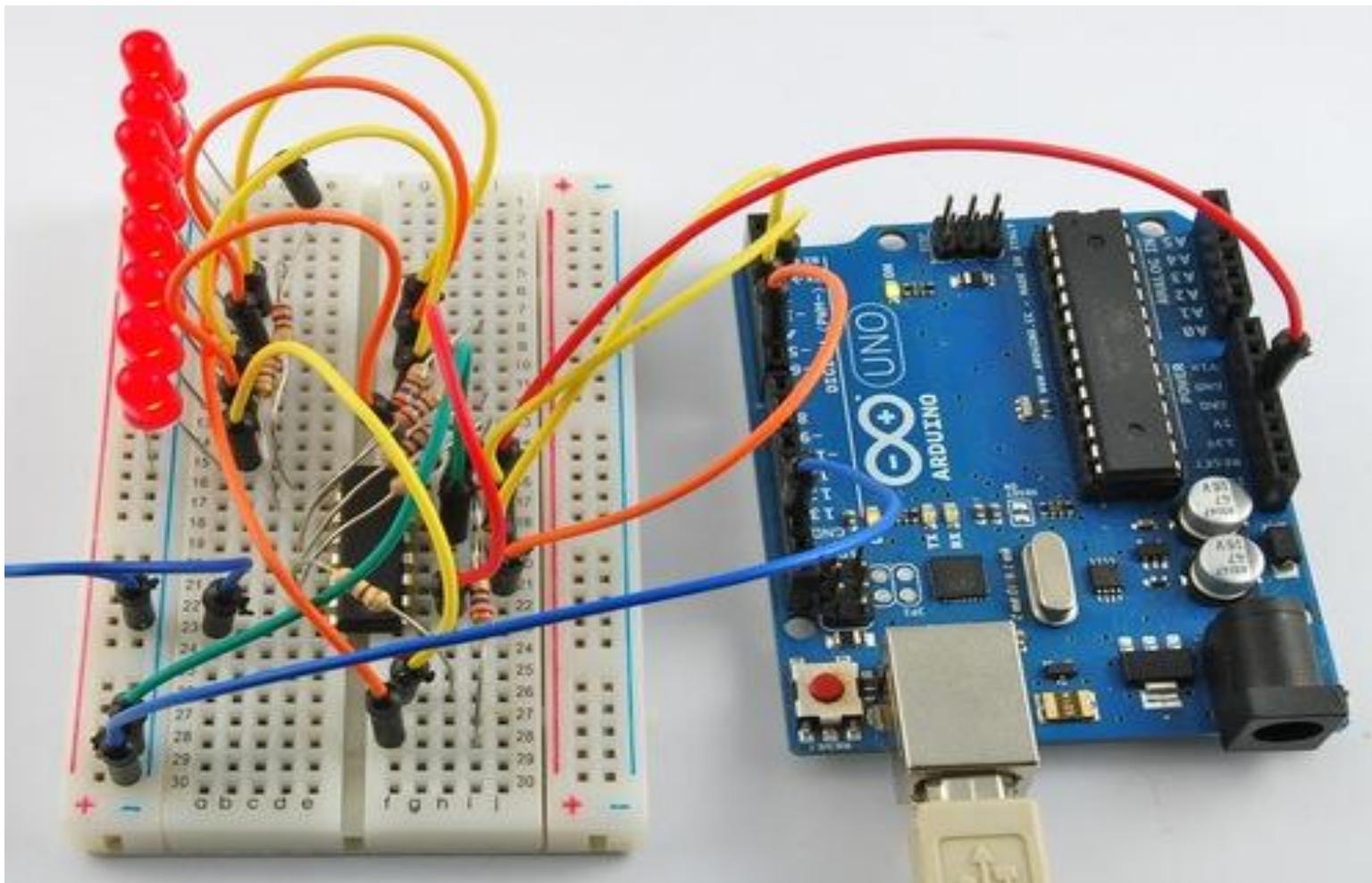


- Textbook**
- Arduino Uno**
- USB A to B Cable**
- Arduino IDE**
- Breadboard**
- Jumper Wire**
- Different Circuit Elements**

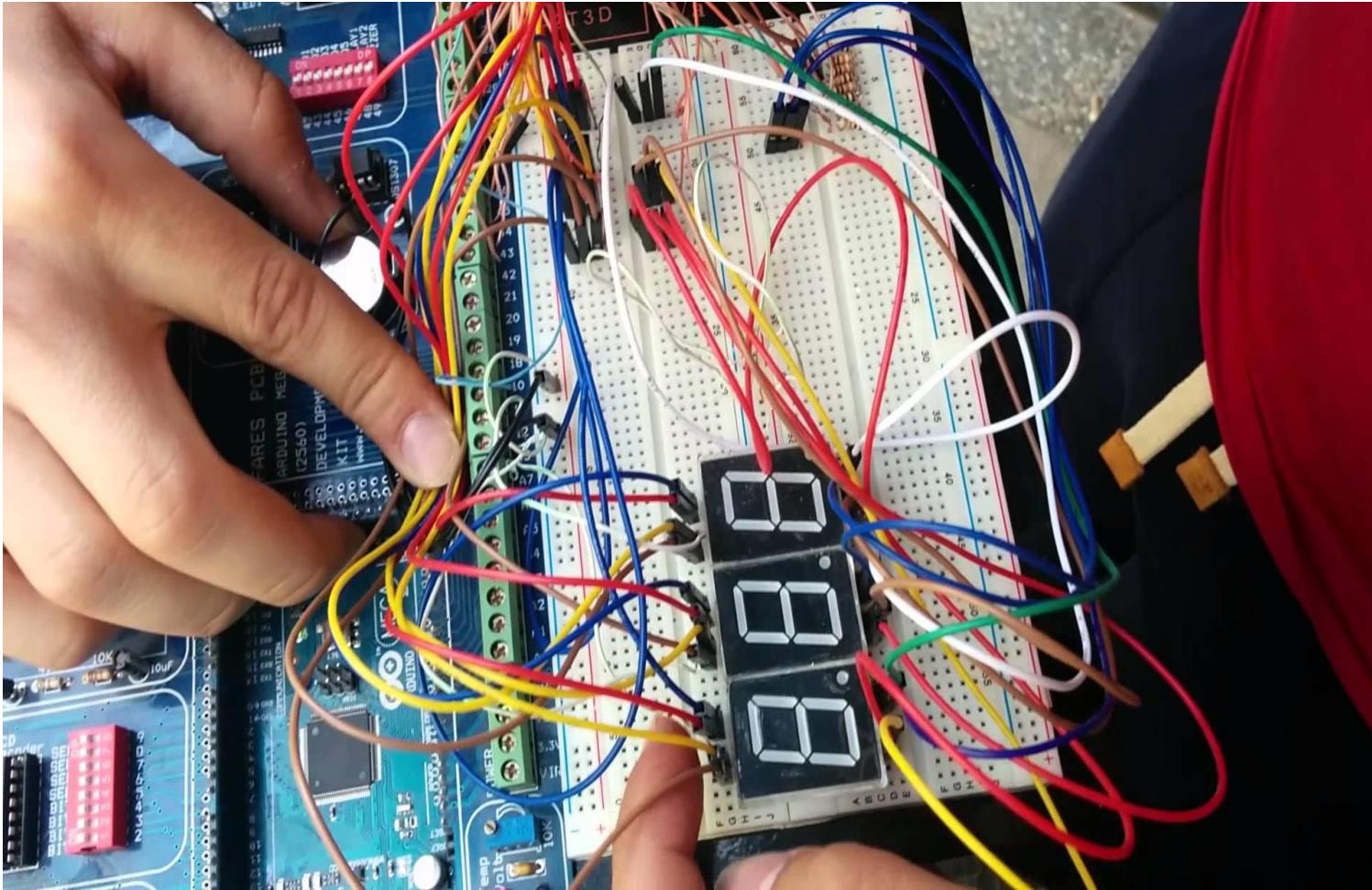


- **For online order:** <https://www.techshopbd.com/>

Example



Example



Appropriate Microcontroller Use



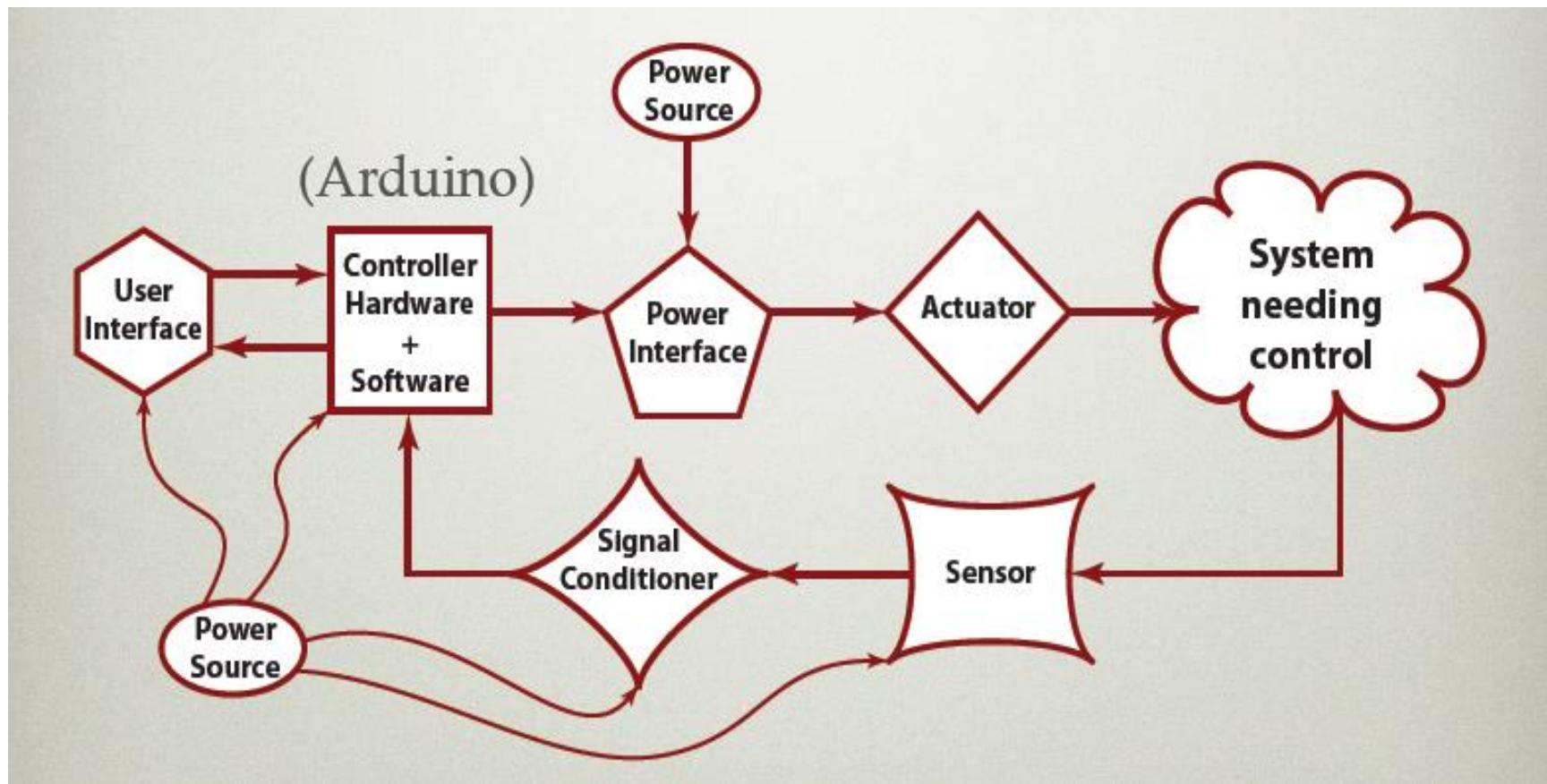
- A microcontroller is a correct tools to use when:
 - Intelligence is required in a system
 - The complexity of a system is reduced when using one.
 - The cost of the microcontroller is “less” then using discrete components to do the same job.
 - A variety of sensors and actuators must be integrated in the system
 - Communication with the other device is necessary

Appropriate Microcontroller Use



- A microcontroller is NOT the correct tool to use when:
 - System requires little or no intelligence.
 - The system can be made easier and/or cheaper using discrete components.
 - Microcontroller is undersized for the problem
 - Too slow
 - Too much number crunching required
 - Too many things going on

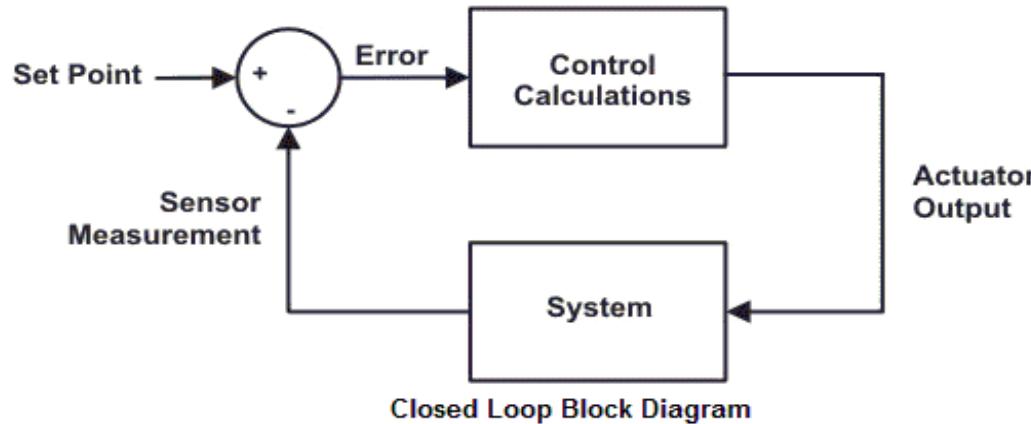
Embedded System Concept Map



Sensor + Signal Conditioner



- Required for “Closed Loop Control”.

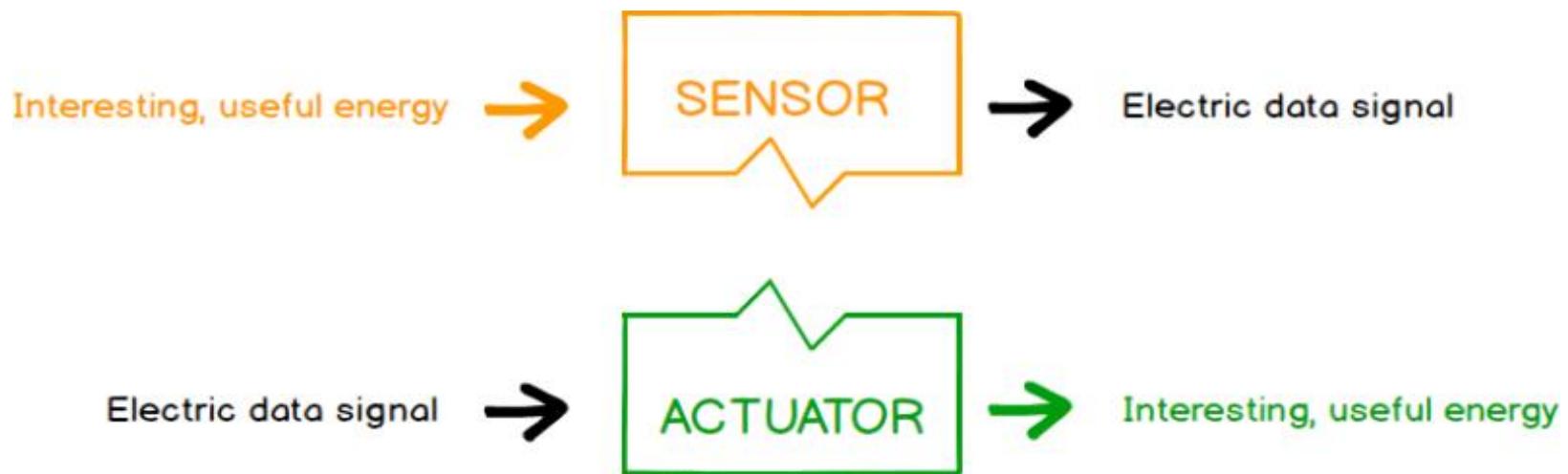


- (*What is the system called if it uses no sensors?*)
- Measures important system variable(s)
- This measurements may need to be conditioned for use by the “Brain”
- The conditioning involves **scaling, offsetting, filtering** etc. required for the controller to have meaningful data.

Actuator



- Energy conversion device
- Converts power to the kind needed by the controlled device.
- Motor, brake, pump, solenoid, linear actuator, flaps etc.
- **It is just opposite to sensor**



Controller Software



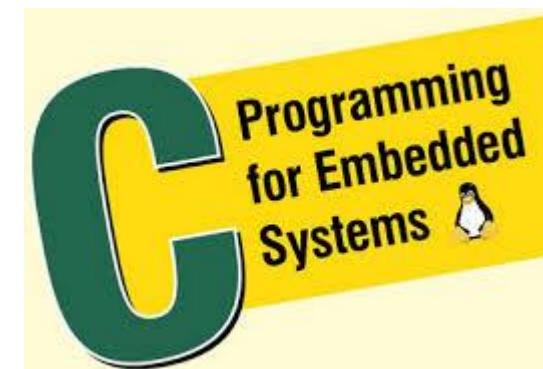
- C/C++/Arduino IDE for many functions
- Assembly language for high speed functions
- FPGA for highest speed (VHDL)



A screenshot of the Arduino IDE interface. The title bar says "sketch_sep14a". The code editor contains the following C++ code:

```
void setup() {
  // put your setup code here, to run once:
}

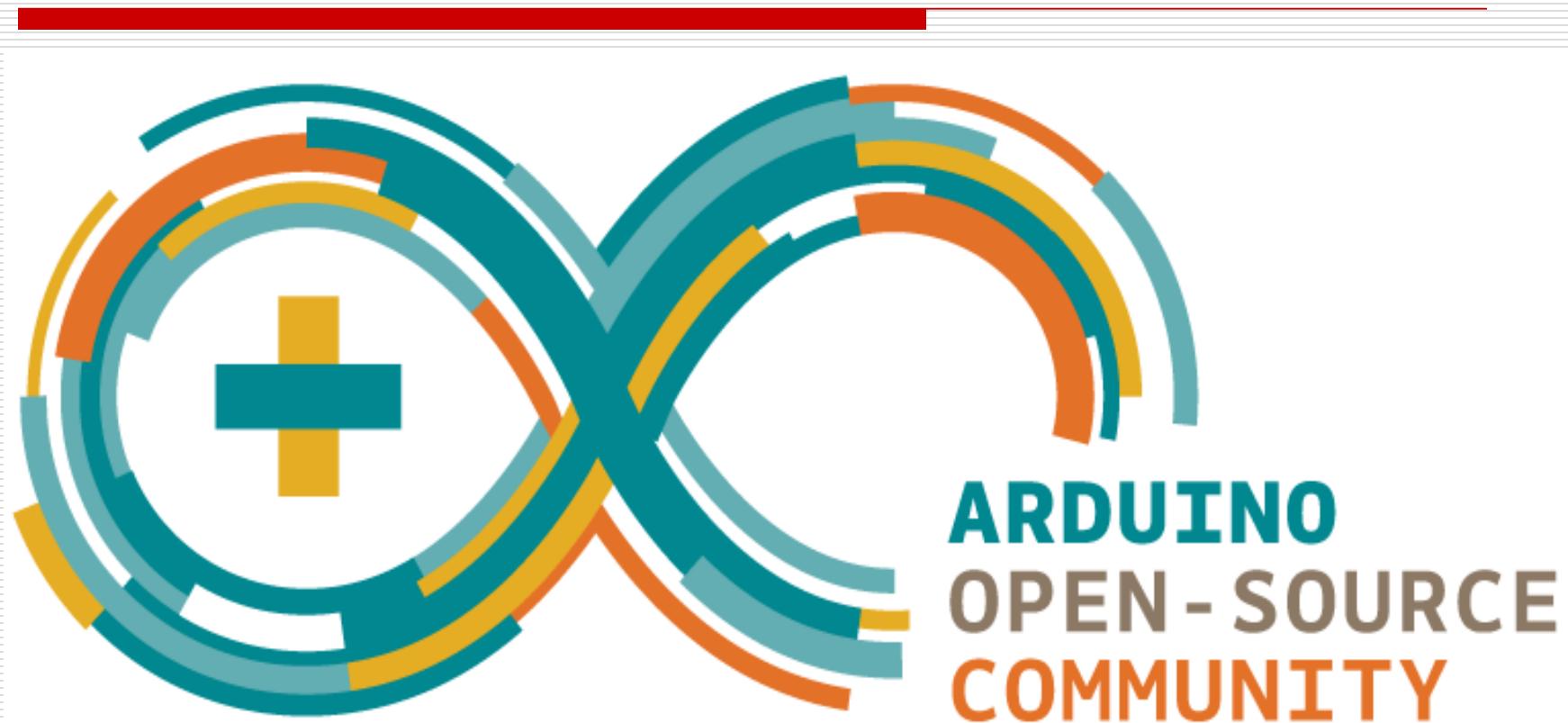
void loop() {
  // put your main code here, to run repeatedly:
}
```



CSE423: Embedded System

Summer-2020

Introduction to Arduino



Todays Lecture



- *Arduino Overview*
- *Arduino Board Types*
- *Arduino specifications*
- *Terminology*

What is Arduino ?



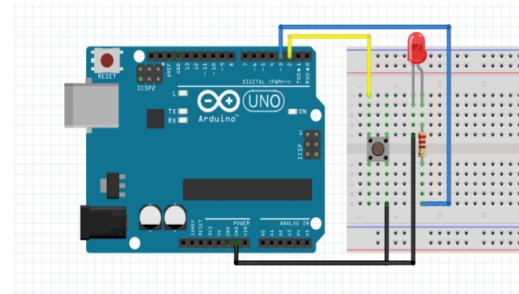
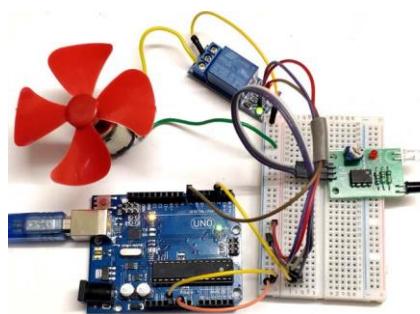
Arduino is a prototype platform (opensource) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.



What Arduino can do?



Arduino boards are able to read analog or digital input signals from different sensors and act accordingly to generate some output like activating a motor, turning LED on/off, connect to the cloud and many other actions.



Arduino Board Types



https://www.sparkfun.com/standard_arduino_comparison_guide

Arduino Board Types (Standard)



Arduino Uno - R3 	Arduino Pro Mini - 5V/16MHz 
Arduino Uno - R3 SMD 	Arduino Fio 
Arduino Leonardo 	Arduino Pro Mini - 3V/8MHz 
Arduino Mega 2560 R3 	



Arduino Board Types (contd.)

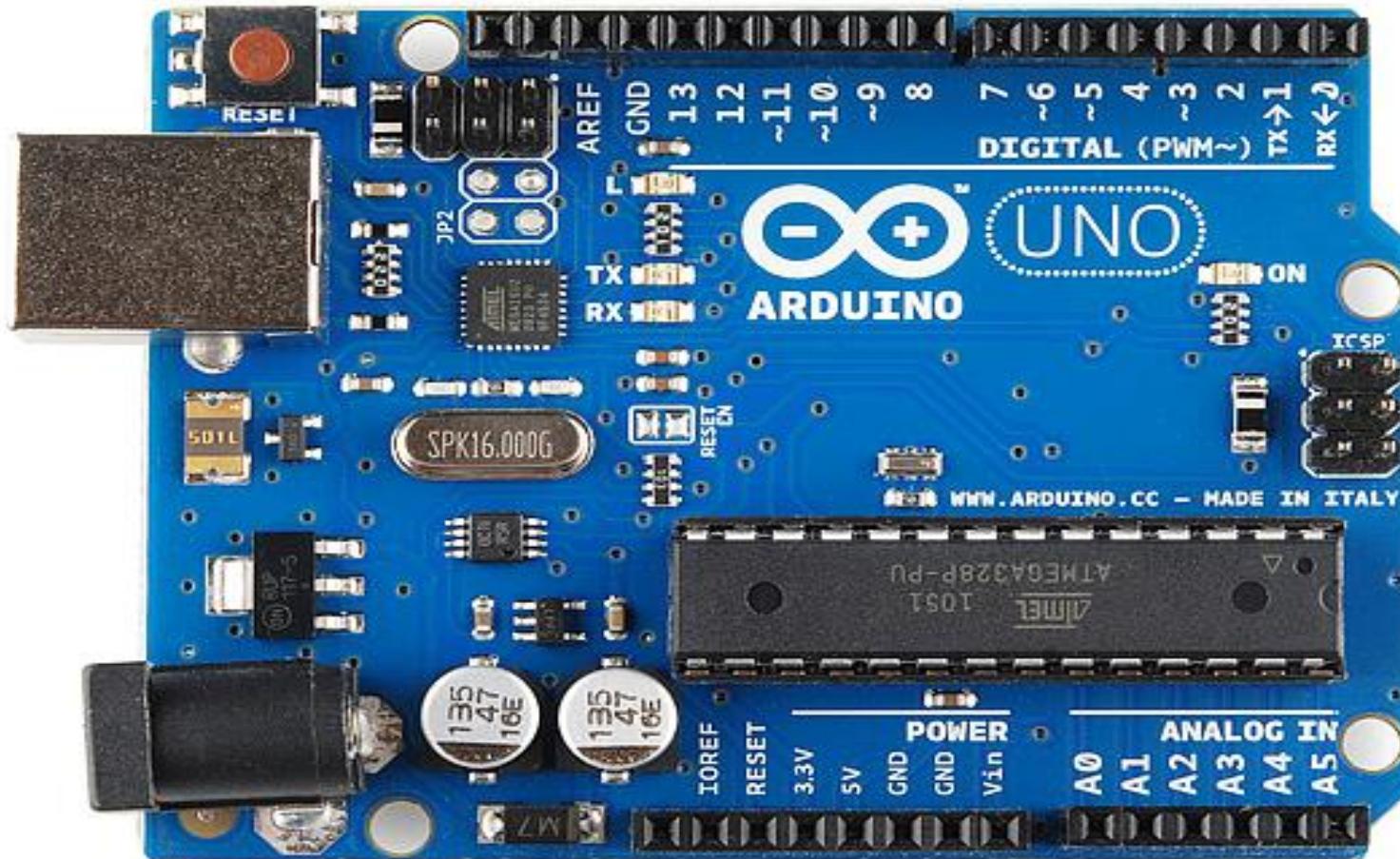
Name	Processor	CPU Speed	Analog In/Out	Flash [kB]	UART
<u>101</u>	Intel® Curie	32MHz	6/0	196	-
<u>Gemma</u>	ATtiny85	8 MHz	1/0	8	0
<u>LilyPad</u>	ATmega168V ATmega328P	8MHz	6/0	16	-
<u>LilyPad SimpleSnap</u>	ATmega328P	8 MHz	4/0	32	-
<u>LilyPad USB</u>	ATmega32U4	8 MHz	4/0	32	-
<u>Mega 2560</u>	ATmega2560	16 MHz	16/0	256	4
<u>Micro</u>	ATmega32U4	16 MHz	12/0	32	1
<u>MKR1000</u>	SAMD21 Cortex-M0+	48MHz	07-Jan	256	1
<u>Pro</u>	ATmega168 ATmega328P	8 MHz 16 MHz	6/0	16 32	1
<u>Pro Mini</u>	ATmega328P	8 MHz 16 MHz	6/0	32	1
<u>Uno</u>	ATmega328P	16 MHz	6/0	32	1
<u>Zero</u>	ATSAMD21G18	48 MHz	06-Jan	256	2



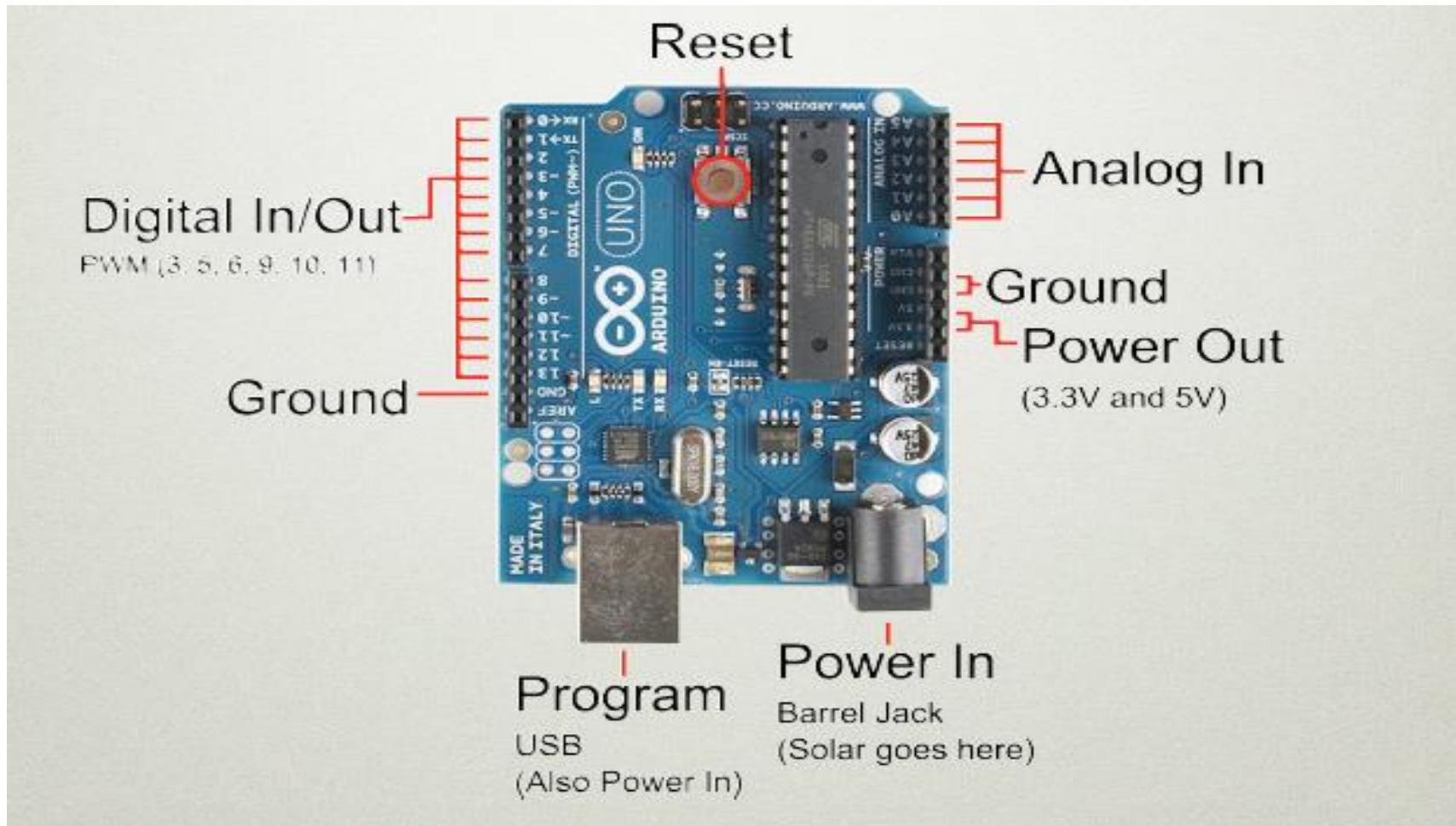
Arduino Board Types (contd.)

Name	Processor	CPU Speed	Analog In/Out	Flash [kB]	UART
<u>Due</u>	ATSAM3X8E	84 MHz	12-Feb	512	4
<u>Esplora</u>	ATmega32U4	16 MHz	-	32	-
<u>Ethernet</u>	ATmega328P	16 MHz	6/0	32	-
<u>Leonardo</u>	ATmega32U4	16 MHz	12/0	32	1
<u>Mega ADK</u>	ATmega2560	16 MHz	16/0	256	4
<u>Mini</u>	ATmega328P	16 MHz	8/0	32	-
<u>Nano</u>	ATmega168 ATmega328P	16 MHz	8/0	16 32	1
<u>Yún</u>	ATmega32U4 AR9331 Linux	16 MHz 400MHz	12/0	32 64MB	1
<u>Arduino Robot</u>	ATmega32u4	16 MHz	6/0	32 KB (ATmega32u4) of which 4 KB used by bootloader	1
<u>MKRZero</u>	SAMD21 Cortex-M0+ 32bit low power ARM MCU	48 MHz	7 (ADC 8/10/12 bit)/1 (DAC 10 bit)	256 KB	1

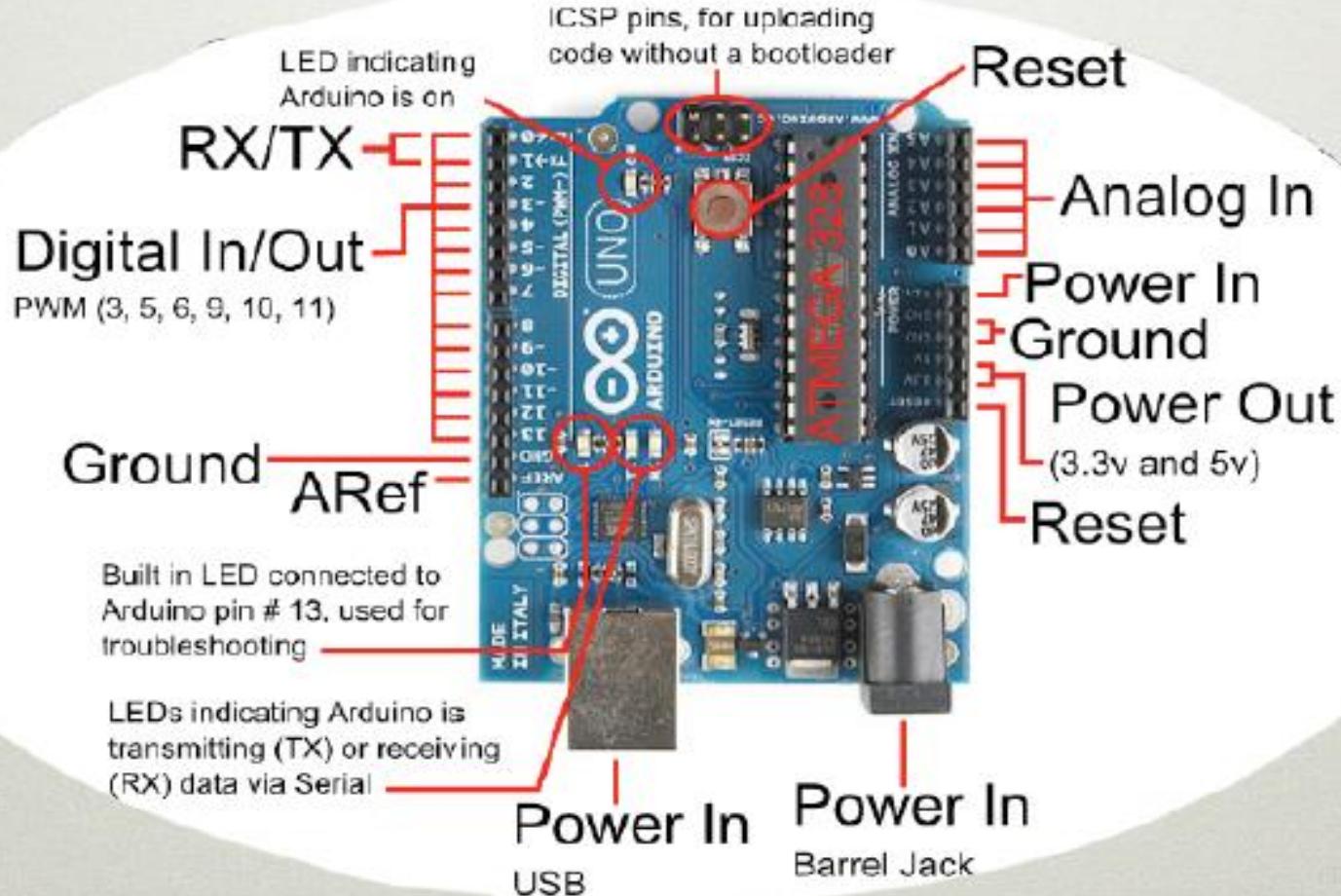
Arduino UNO



Arduino UNO



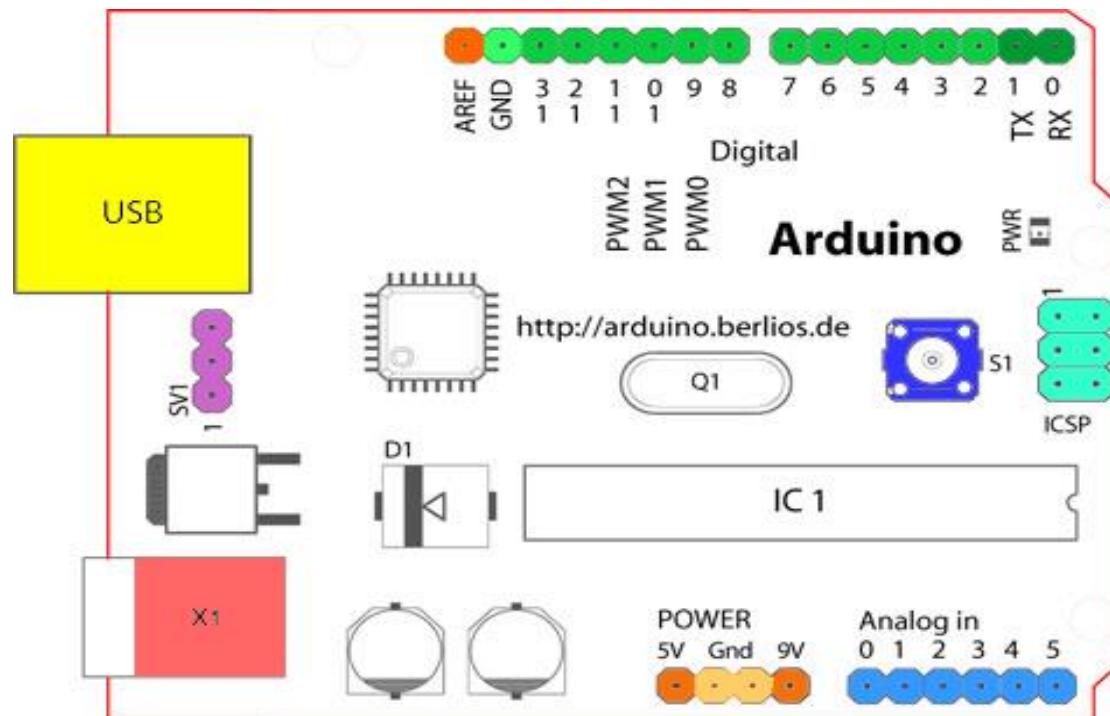
Arduino UNO



Arduino UNO



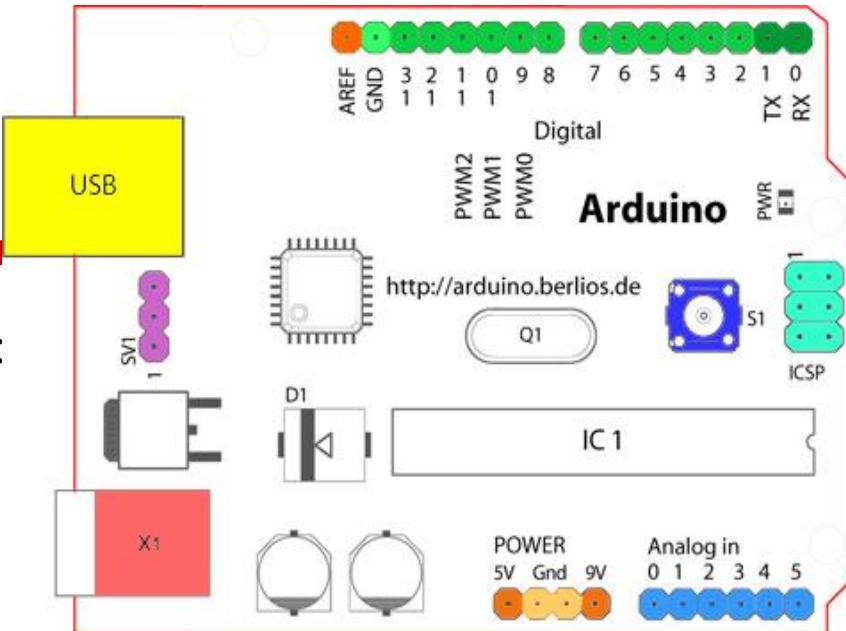
- Looking at the board from the top down, this is an outline of what you will see (parts of the board you might interact with in the course of normal use are highlighted):



Arduino UNO

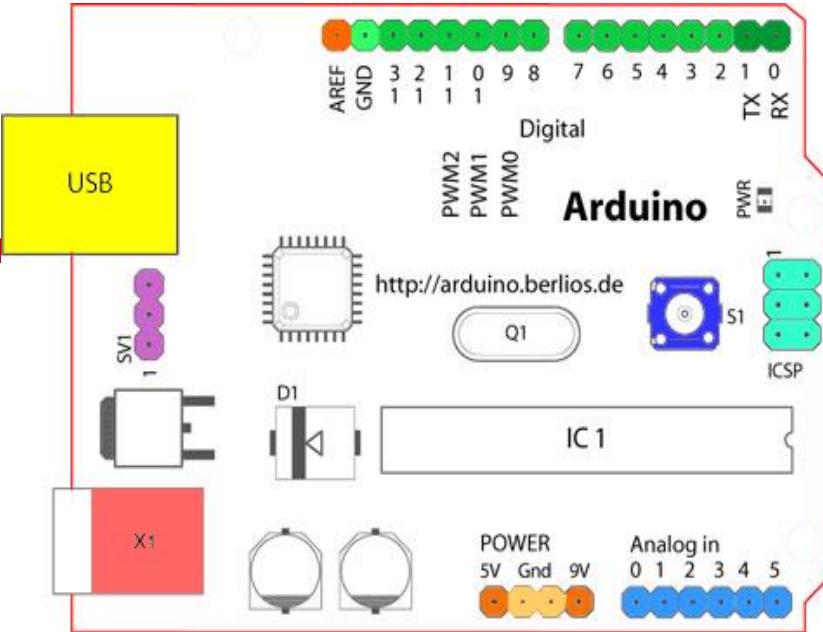
- Starting clockwise from the top center:

- Analog Reference pin (**orange**)
- Digital Ground (**light green**)
- Digital Pins 2-13 (**green**)
- Digital Pins 0-1/Serial In/Out - TX/RX (**dark green**)
 - ❖ *These pins cannot be used for digital I/O (**digitalRead** and **digitalWrite**) if you are also using serial communication (e.g. **Serial.begin**).*
- Reset Button - S1 (**dark blue**)
- In-circuit Serial Programmer or ICSP (**blue-green**)



Arduino UNO

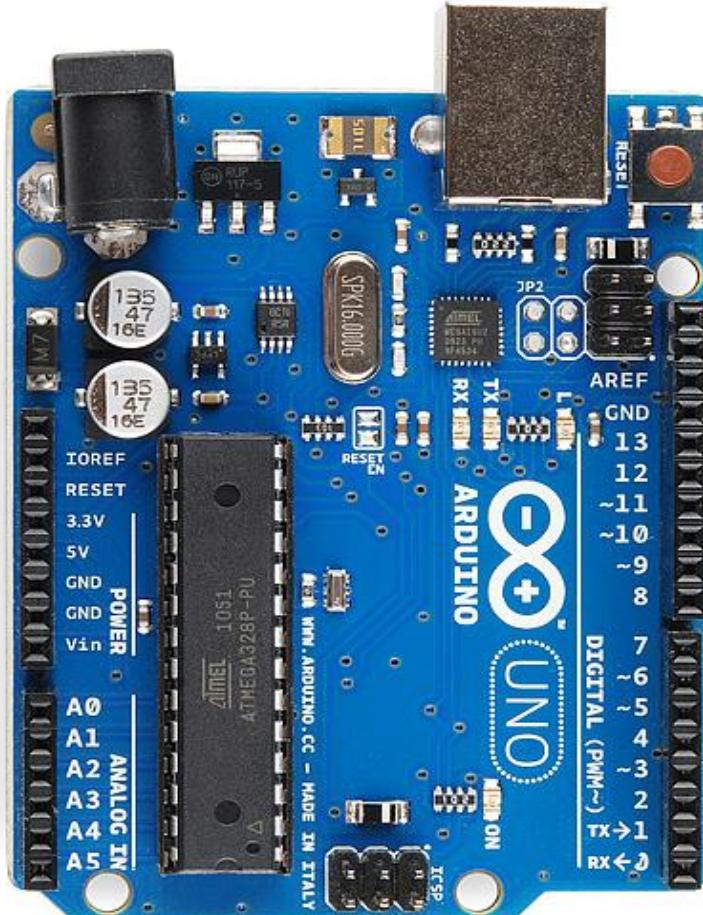
- Analog In Pins 0-5 (light blue)
- Power and Ground Pins
(power: orange, grounds: light orange)



- External Power Supply In (9-12VDC) - X1 (pink)
- Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1 (purple)
- USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow)



Arduino UNO Specifications



Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recom)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (6 PWM)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (0.5 KB boot loader)
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g



Arduino Terminology

- “**sketch**” – a program you write to run on an Arduino board
- “**pin**” – an input or output connected to something.
e.g. output to an LED, input from a knob.
- “**digital**” – value is either HIGH or LOW. (on/off, one/zero)
e.g. switch state
- “**analog**” – value ranges, usually from 0-255.
e.g. LED brightness, motor speed, etc.



Memory

- There are three pools of memory in the microcontroller used on AVR(ATmega 328)-based Arduino boards :
 - **Flash memory**: Flash memory known as **program space**, is where the Arduino sketch is stored.
 - **SRAM (static random access memory)**: SRAM is where the sketch **creates and manipulates variables** when it runs.
 - **EEPROM**: EEPROM is memory space that programmers can use to **store long-term information**.

```
Flash 32k bytes (of which .5k is used for the bootloader)
SRAM 2k bytes
EEPROM 1k byte
```



Memory

- Flash memory and EEPROM memory are **non-volatile** (the information persists after the power is turned off). SRAM is **volatile** and will be lost when the power is cycled.

- Notice that there's not much SRAM available in the Uno. It's easy to use it all up by having lots of strings in your program. For example, a declaration like:

- `char message[] = "I support the Cape Wind project.;"`

- puts 33 bytes into SRAM (each character takes a byte, plus the '\0' terminator). This might not seem like a lot, but it doesn't take long to get to 2048, especially if you have a large amount of text to send to a display, or a large lookup table, for example.



CSE 423: Embedded Systems

Lecture: 3

Intro to the Arduino Sketch

Topics:

Serial Communication

Loop

Function

Terminology

“*sketch*” – a program you write to run on an Arduino board

“*pin*” – an input or output connected to something.
e.g. output to an LED, input from a knob.

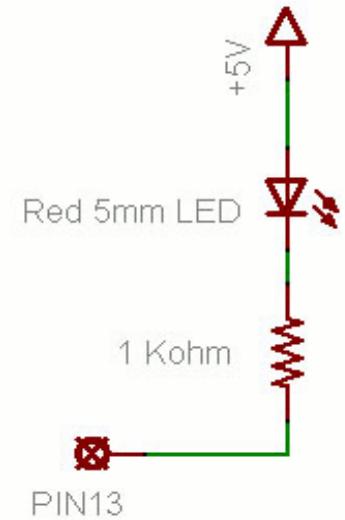
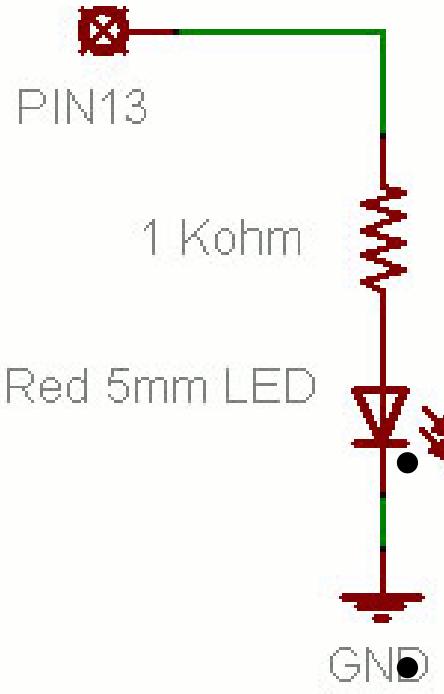
“*digital*” – value is either HIGH or LOW.
(aka on/off, one/zero) e.g. switch state

“*analog*” – value ranges, usually from 0-255.
e.g. LED brightness, motor speed, etc.

Arduino Timing

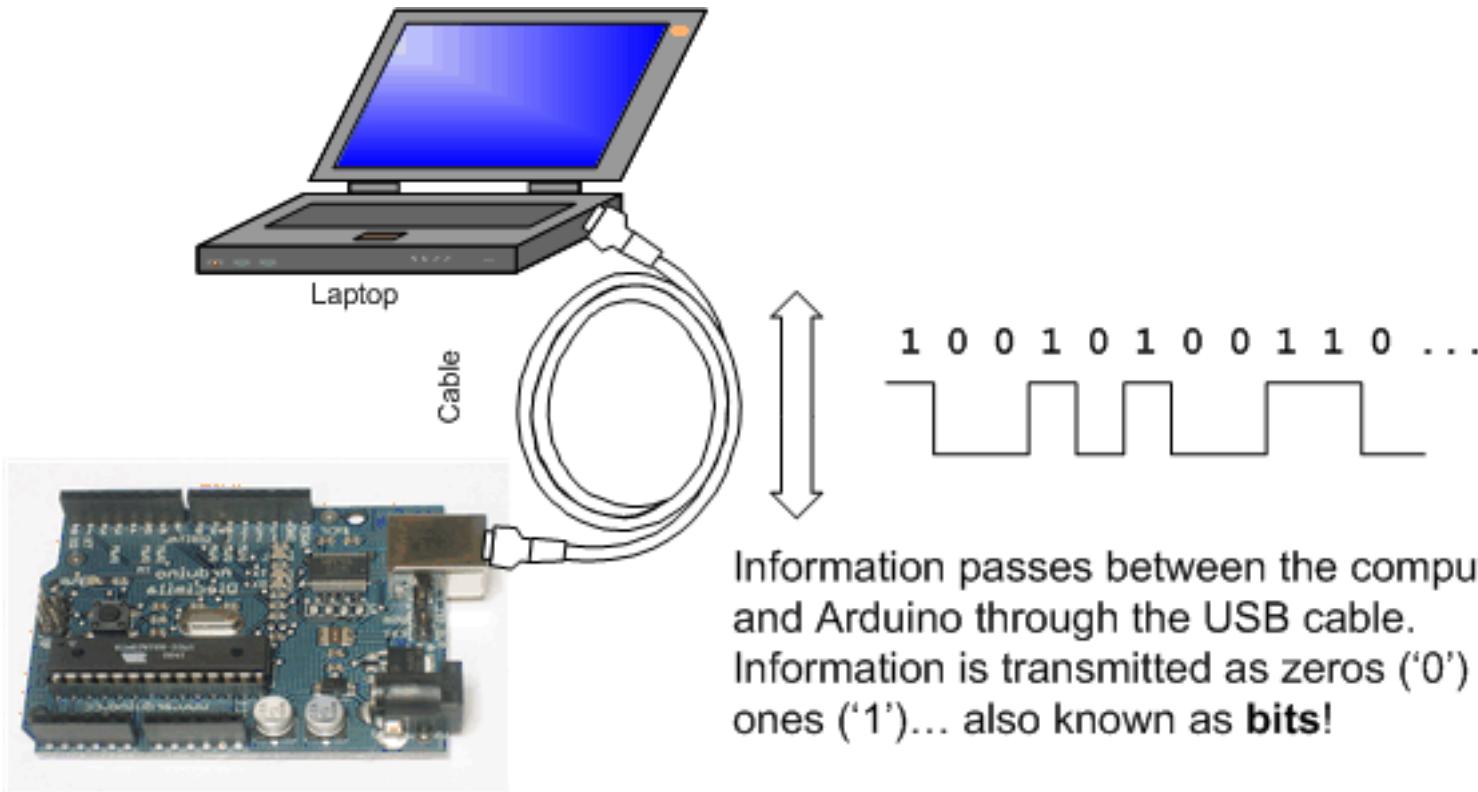
- `delay (ms)`
 - Pauses for a few milliseconds
- `delayMicroseconds (us)`
 - Pauses for a few microseconds

Putting It Together



- Complete the sketch (program) below.
- What output will be generated by this program?
- What if the schematic were changed? →

Serial Communication



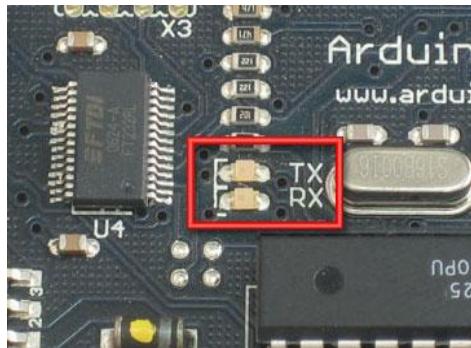
Serial Communications

- “Serial” because data is broken down into bits, each sent one after the other down a single wire.
- The single ASCII character ‘B’ is sent as:

‘B’ = 0 1 0 0 0 0 1 0
= L H L L L L H L
= 

- Toggle a pin to send data, just like blinking an LED
- You could implement sending serial data with `digitalWrite()` and `delay()`
- A single data wire needed to send data. One other to receive.

Serial Communication



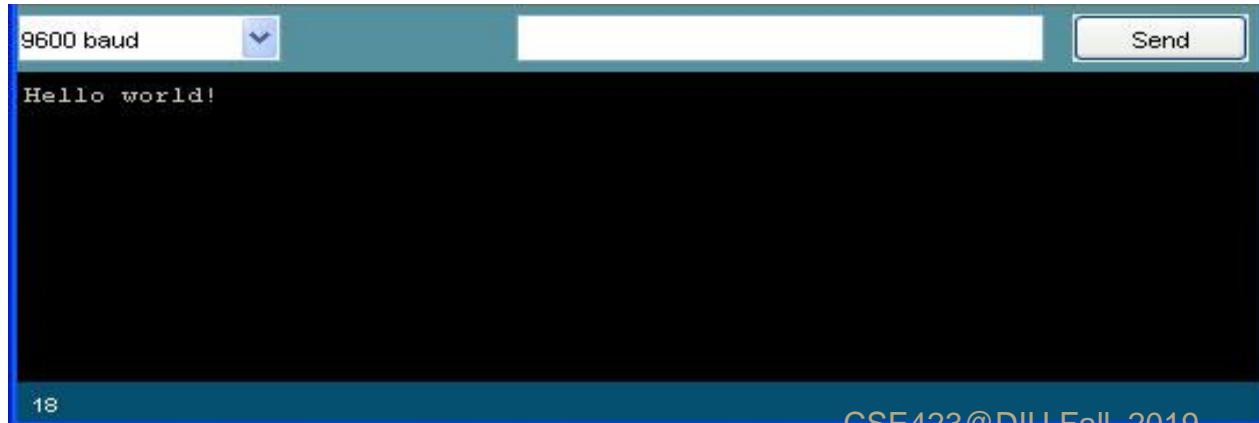
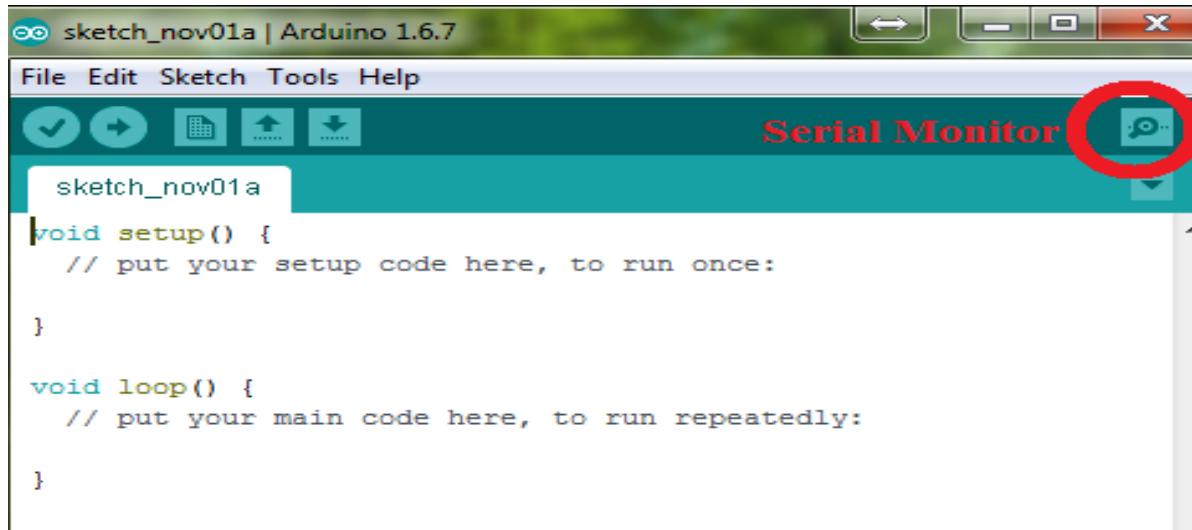
- ***Compiling*** turns your program into binary data (ones and zeros)
- ***Uploading*** sends the bits through USB cable to the Arduino
- The two LEDs near the USB connector blink when data is transmitted
 - **RX** blinks when the Arduino is receiving data
 - **TX** blinks when the Arduino is transmitting data

First Program

```
void setup( )                  // run once, when the sketch starts
{
    Serial.begin(9600);        // set up Serial library at 9600 bps
    Serial.println("Hello world!"); // prints hello with a line break
}

void loop( )                  // run over and over again
{                            // do nothing!
}
```

Open the Serial Monitor and Upload the Program

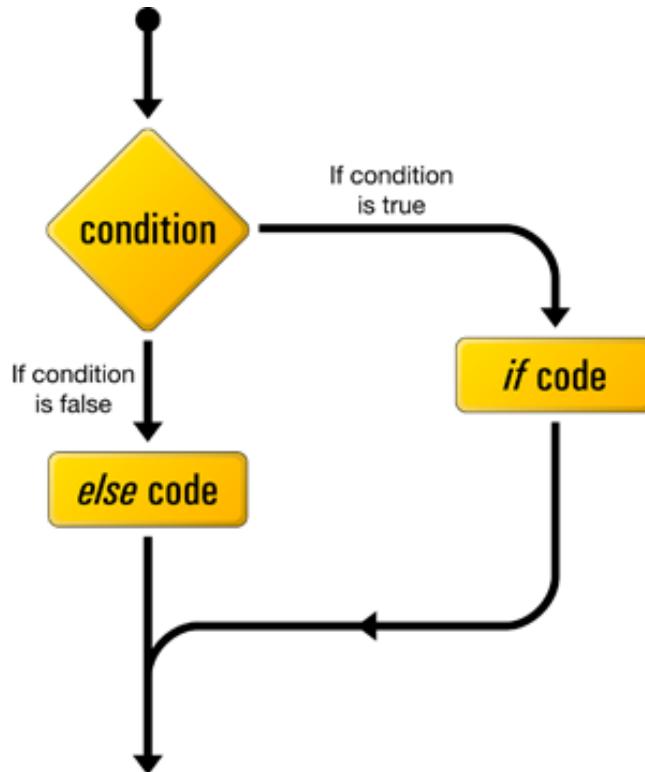


Modify the Program (each bullet is a different mod)

- Move `Serial.println("Hello world!");` to `loop()`
- Add the following to `setup()`:
`int a = 5;`
`int b = 10;`
`Serial.print("a + b = ");`
`Serial.println(a + b);`
- Replace the code above with the following:

```
int val = 33;  
Serial.print(val);  
Serial.print(val, DEC);  
Serial.print(val, BIN);
```

Conditional Statement



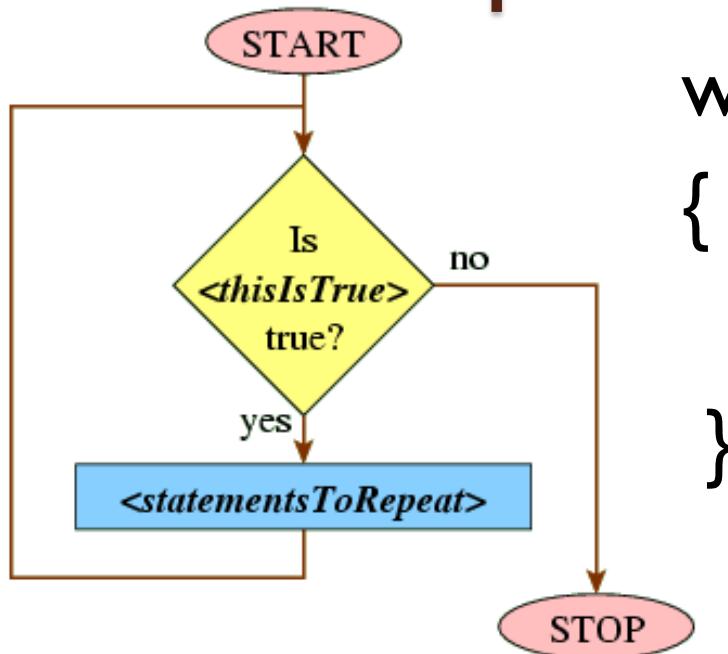
```
if (some Condition)
{
    // do stuff if the condition is true
}
else
{
    // do stuff if the condition is false
}
```

Conditional Statement

```
int printMessage = 1;  
void setup()  
{ Serial.begin(9600);  
}  
  
void loop()  
{  
    if (printMessage == 1) {  
  
        Serial.println("Message");  
        printMessage= 0;  
    }  
}
```

```
int printMessage = 1;  
void setup()  
{ Serial.begin(9600);  
}  
  
void loop()  
{  
    if (printMessage == 1)  
    {  
        Serial.println("Message");  
        printMessage= 0;  
    }  
    else {  
        Serial.println("NO Message");  
        printMessage= 1;  
    }  
}
```

while Loop



```
while(expression)
{
    statement(s);
}
```

Example

```
int var = 0;
while (var < 200) {
    // do something repetitive 200 times
    var = var + 1;
}
```

while Loop

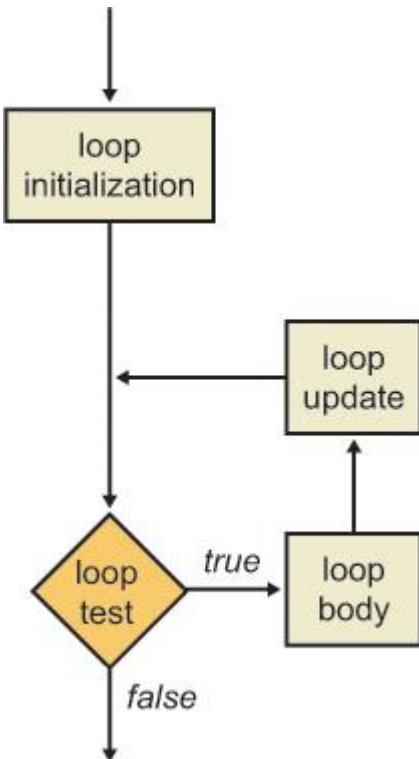
```
void setup()
{
    Serial.begin(9600);

    int count = 0;

    while (count < 5) {
        Serial.println("Hello world!");
        count = count + 1;
    }
}
```

```
void loop()
{
}
```

for loop



initialisation iteration step
condition

```
for (int i = 0; i < 10; i++) {  
    // This is the loop body  
    // add your code here  
}
```

for Loop

```
void setup()
{
    Serial.begin(9600);
    for (int count = 0; count < 5; count++) {
        Serial.println("Hello world!");
    }
}
```

```
void loop()
{
}
```

Functions

- `loop()` and `setup()` are procedures
- You can create your own functions

Anatomy of a C function

Datatype of data returned,
any C datatype.

"void" if nothing is returned.

Parameters passed to
function, any C datatype.

```
int myMultiplyFunction(int x, int y){  
    int result;  
    result = x * y;  
    return result;  
}
```

Function name

Return statement,
datatype matches
declaration.

Curly braces required.

`void setup()`

{
}

`void loop()`

{
}

Both `setup()` and `loop()`
have no parameters and
return no values

Functions: Example 1

```
void setup() {  
    Serial.begin(9600);  
  
    Serial.println("Before example function call.");  
    delay(1000);  
    example(); Function call...  
}  
  
Serial.println("After example function call.");  
delay(1000);  
}  
  
void loop() {  
}  
  
void example() Function starts here.  
{  
    Serial.println("During example function call.");  
    delay(1000);  
} When done, sketch returns to next instruction after function call.
```

Functions: Example 2

```
void setup() {  
    Serial.begin(9600);  
    pitch(3500);  
    Serial.println("Playing high pitch tone...");  
    delay(1000);  
  
    (5) Pass 2000 to Hz.  
  
    pitch(2000);  
    Serial.println("Playing lower pitch tone...");  
    delay(1000);  
}  
  
void loop()  
{  
    (6) Function  
    executes with  
    Hz = 2000  
}  
  
void pitch(int Hz)  
{  
    Serial.print("Frequency = ");  
    Serial.print(Hz);  
    tone(4, Hz, 1000);  
    delay(1000);  
}  
}  
  
(4) No more code—return to next instruction in sketch.
```

(1) Call sends sketch to function...

(2)...passing 3500 to Hz.

(3) Function executes with Hz = 3500.

CSE423: Embedded System

Summer-2020



Installation of Arduino IDE and Work with very First Code





Todays Lecture

- *Installation of IDE*
- *Power-up the board*
- *Step by step execution of code*

How to interact with Arduino?



At first we need Arduino IDE (Integrated Development Environment) to interact with the Board which is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. It is available at:

<https://www.arduino.cc/en/main/software>

Download the Arduino IDE

The screenshot shows the Arduino Software (IDE) download page. On the left, there's a large teal circular icon containing a white infinity symbol with a minus sign on the left and a plus sign on the right. To the right of the icon, the text "ARDUINO 1.8.12" is displayed in bold. Below this, a paragraph explains the software's purpose: "The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software." It also notes that the software can be used with any Arduino board and refers to the "Getting Started" page for installation instructions. On the right side of the page, there are download links for different operating systems. For Windows, there are links for "Windows Installer, for Windows 7 and up" and "Windows ZIP file for non admin install". For Mac OS X, it says "Mac OS X 10.10 or newer". For Linux, it lists "Linux 32 bits", "Linux 64 bits", "Linux ARM 32 bits", and "Linux ARM 64 bits". At the bottom right, there are links for "Release Notes", "Source Code", and "Checksums (sha512)".

How to interact with Arduino?



- If we want, we can even use the Arduino web editor:

https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduino-web-editor-on-various-platforms-4b3e4a?f=1

The screenshot shows the Arduino Create web editor interface. On the left, a breadboard diagram is displayed with various components and connections. In the center, a code editor window shows the following Arduino sketch:

```
void setup(){
}

void loop(){
}
```

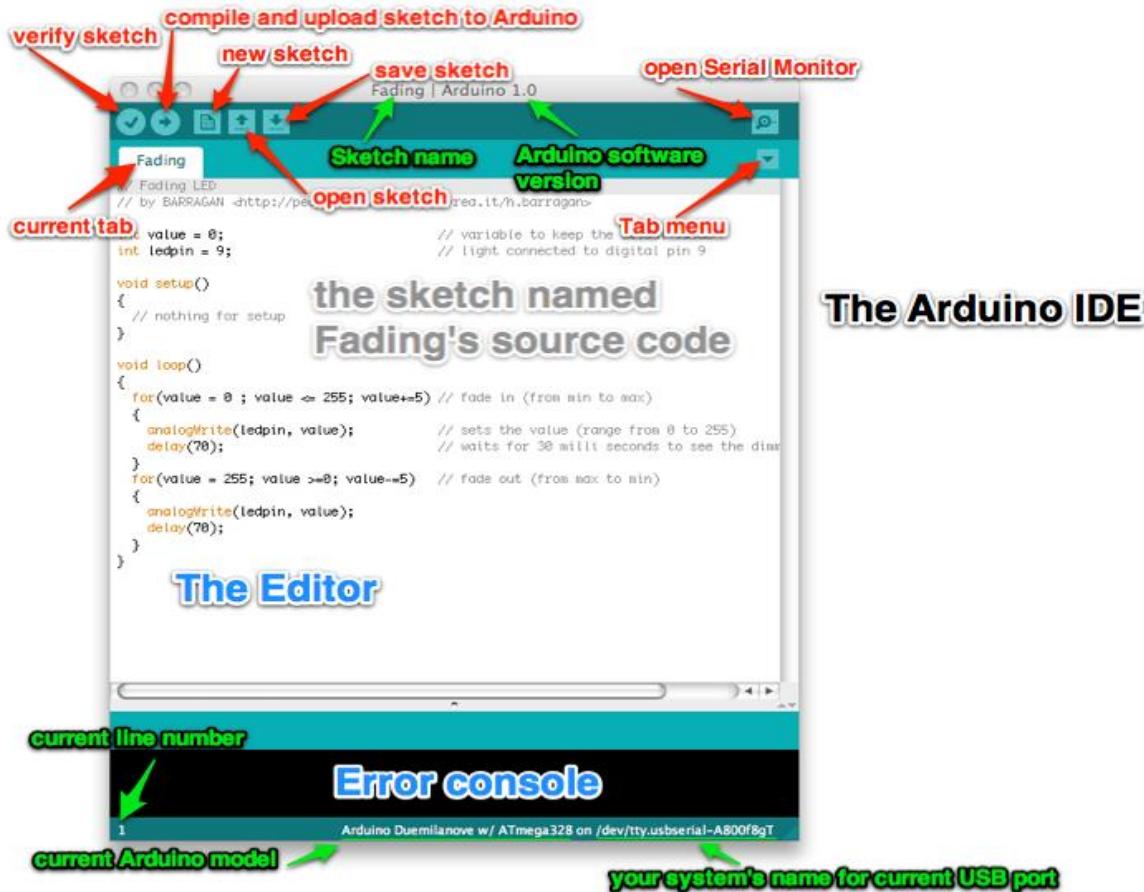
On the right, there is a sidebar with the following information:

- AUTHOR**: Arduino_Genuino (67 PROJECTS, 7,604 FOLLOWERS) with a **FOLLOW** button.
- PUBLISHED ON**: November 21, 2016.
- RESPECT PROJECT** and **WRITE A COMMENT** buttons.
- Share** button at the bottom.

Installing Arduino IDE



- After unzipping and running the `arduino.exe` file, the arduino IDE will appear.



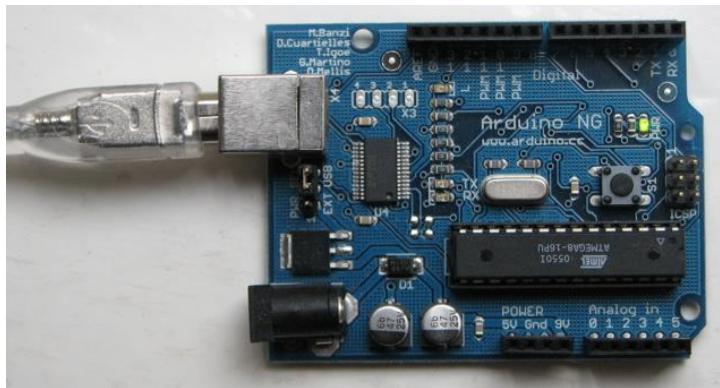


Coding in Arduino IDE

- *Get the Arduino Software and unzip it*
- *Run arduino.exe*
- *Do some setting changes*
- *Write the first sketch (Blinking LED)*
- *Verify the syntax*
- *Board and Serial Port selection*

Power-up the board

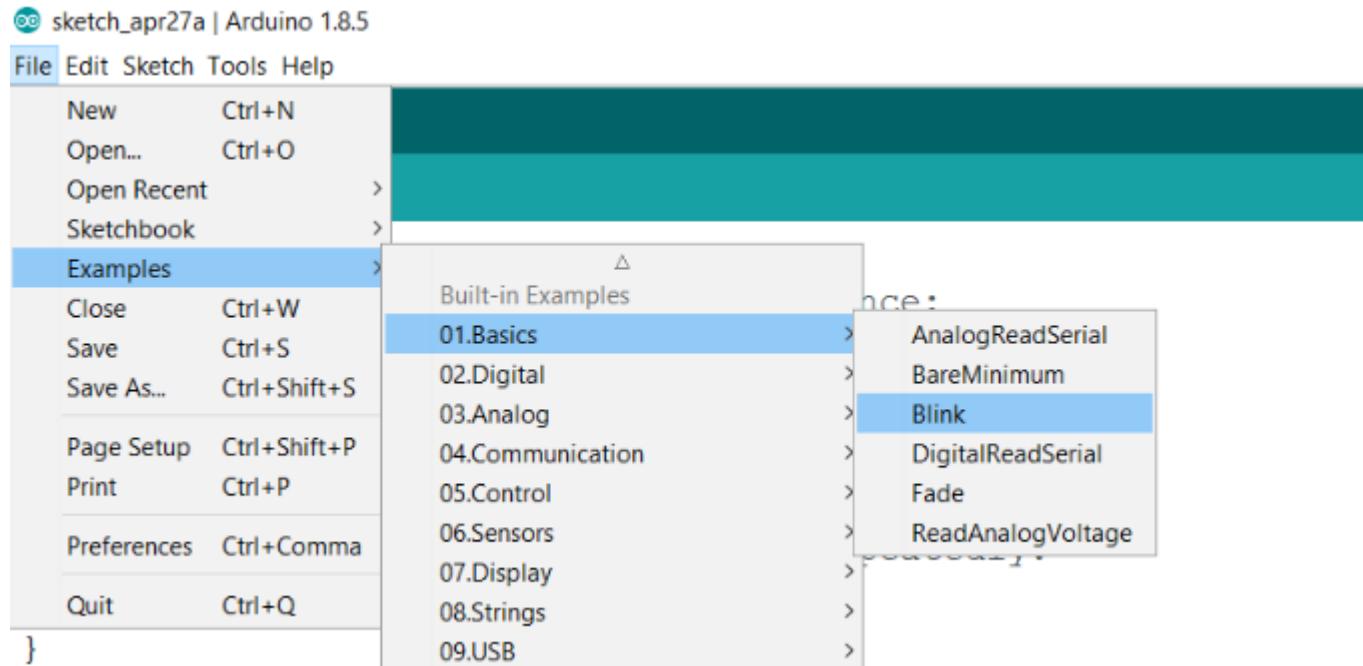
- Option 1: USB cable
- Option 2: External adapter



How to use example code?



- Step-2: Open the LED blink example sketch: **File > Sketchbook > Examples > led_blink**



Writing 1st sketch on Arduino IDE



- Now we are good to go for writing a sketch. Write a sketch to blink a LED using pin 13.

The screenshot shows the Arduino IDE interface with the title bar "sketch_oct03a | Arduino 1.0.5". The code editor contains the following sketch:

```
int led = 13;

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

void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

The status bar at the bottom indicates "10" on the left and "Arduino Uno on COM13" on the right.

Writing 1st sketch on Arduino IDE



- Now press the **verify** button to compile the sketch. If there is no any syntax error, you will be shown ‘Done Compiling’. If there exists any error, that error will be shown on the error console window.

```
sketch_oct03a | Arduino 1.0.5
File Edit Sketch Tools Help
Verify →
int led = 13;

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

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}

Done compiling.
C:\Users\ASUS\AppData\Local\Temp\build40407\9854
5540971358.tmp\sketch_oct03a.cpp.hex
Binary sketch size: 1,084 bytes (of a 32,457 byte maximum)

10 Arduino Uno on COM13
```

Verify → Error console window



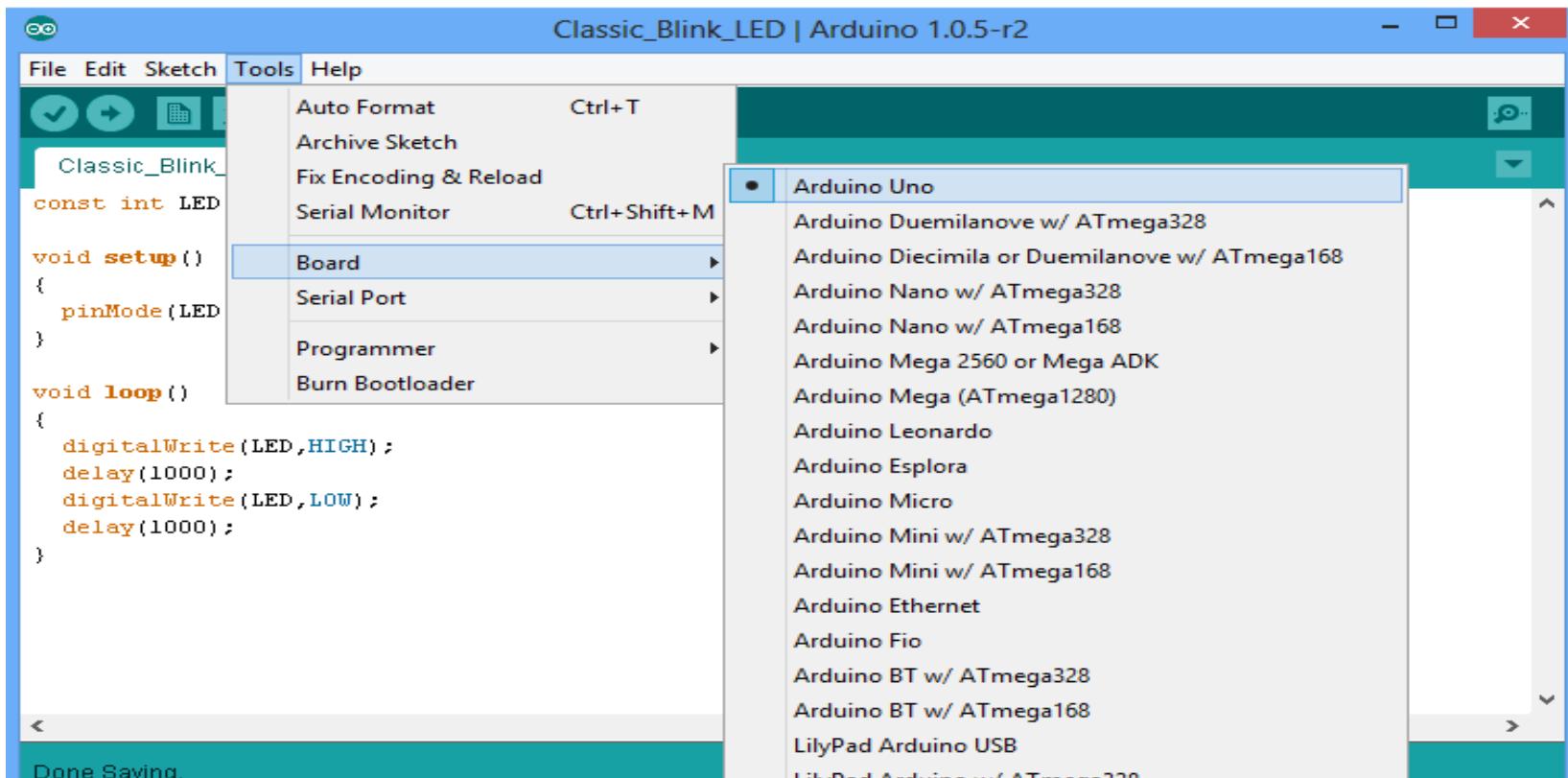
How to upload a code?

- First select the **Board**
- Then select the **Port** (in new IDE port is automatically selected if there only one board connected)

Upload a Code

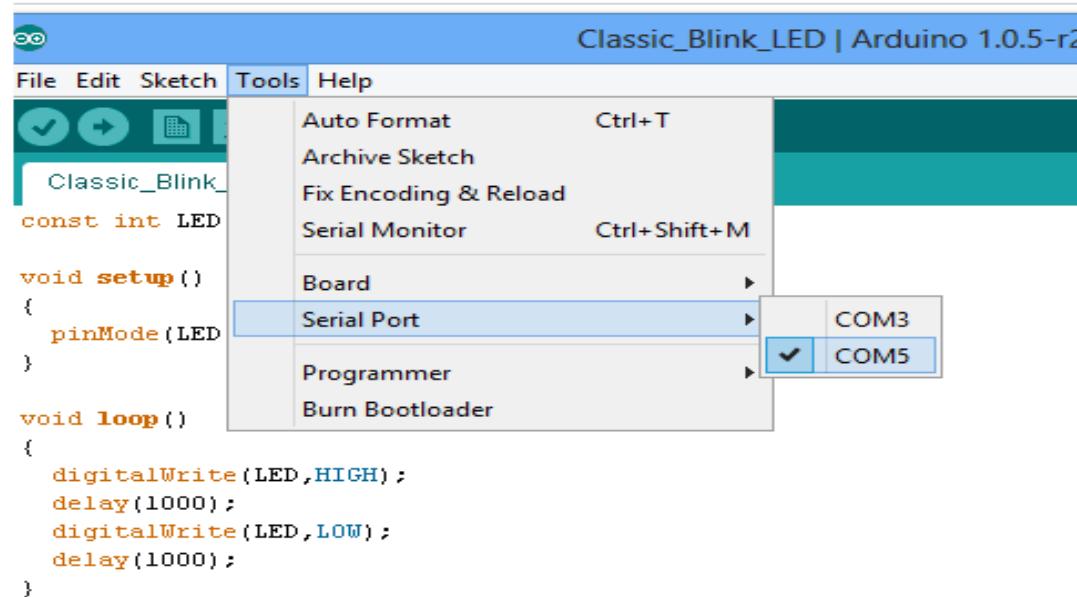


- Go to **Tools > Board** and select the appropriate arduino model on which the sketch is going to be uploaded. For our case, select Arduino Uno.



Upload a Code

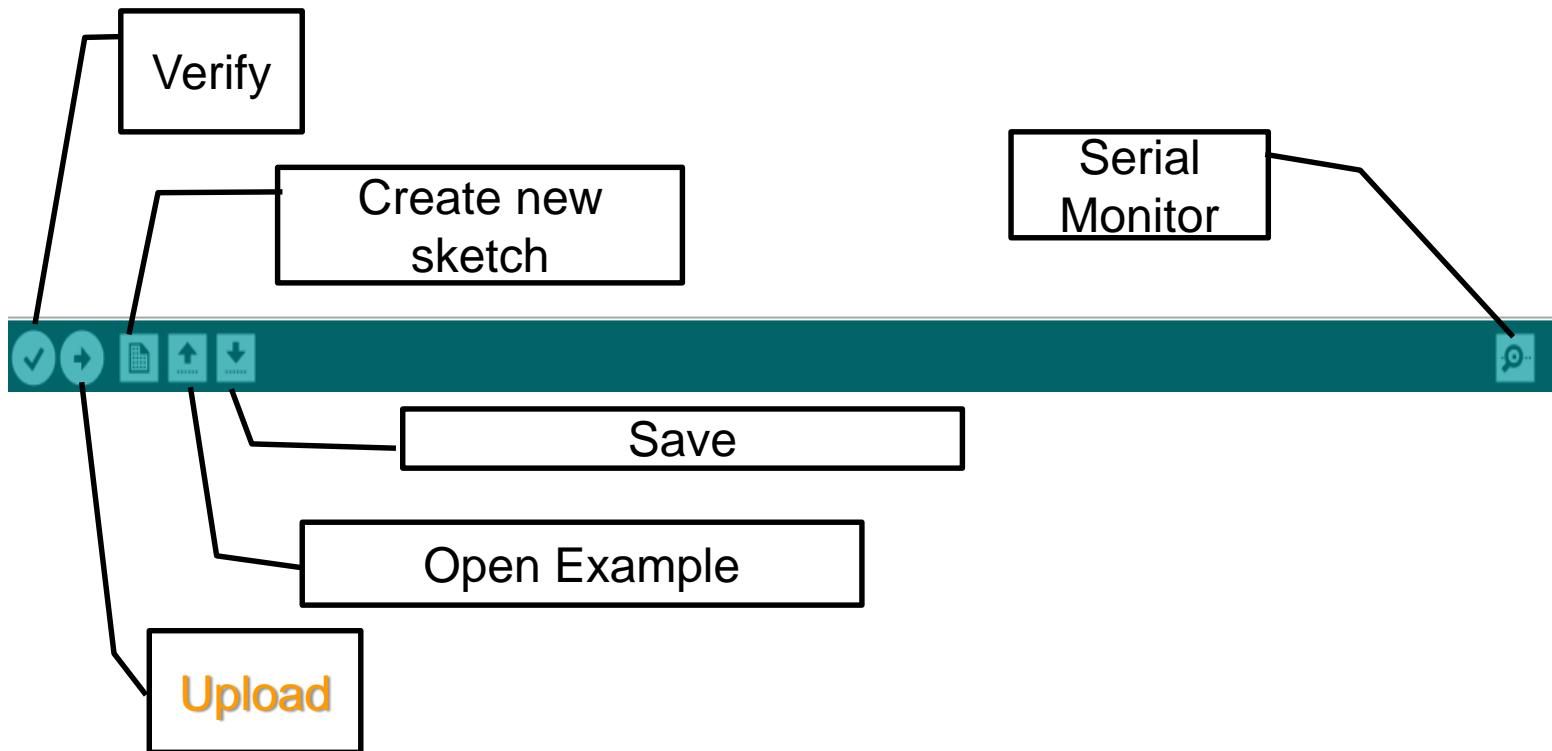
- Go to **Tools > Serial Port** and select the appropriate port on which the hardware Arduino Uno is connected to your PC/Laptop.
- Usually the highest number of port
- Port selection is not required for simulation.



How to upload a code?



- Step-4: Finally upload the code by clicking **upload** button





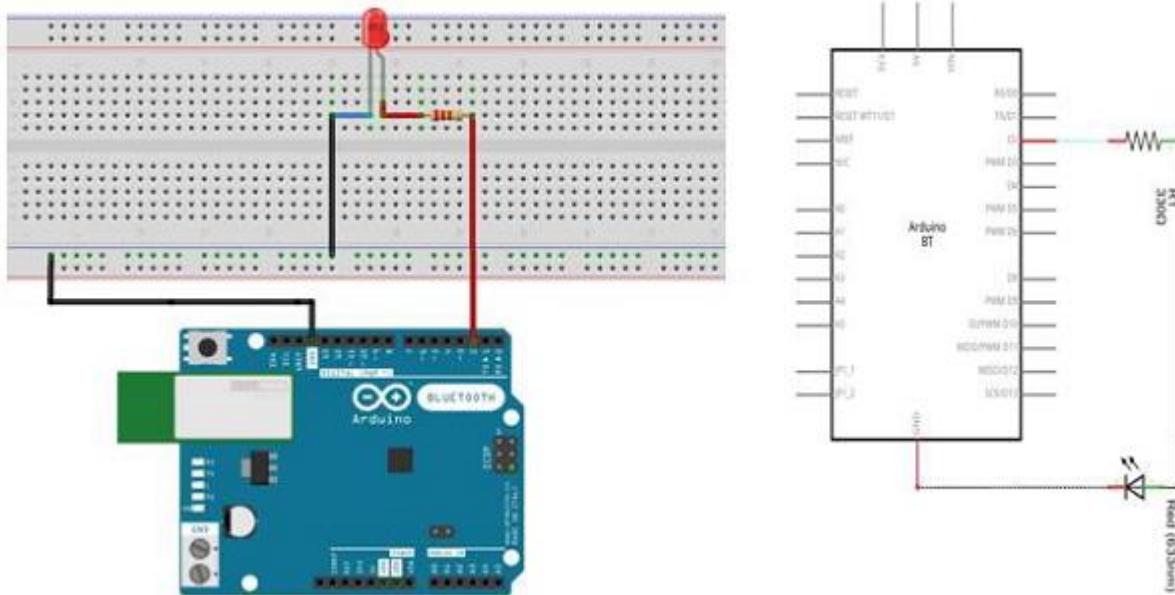
Outcome:

- Once the code is successfully uploaded, there is a built-in led with the pin# 13 which will be Turned ON.

Task



- Do the same task with external bread board and LED by following the connection diagram:



CSE 423

Introduction to Hardware for
Embedded Systems

Input-Analog Sensors

There are countless variety of variable resistors to measure all kinds of physical properties. A sensor or variable resistor changes in its resistance based on the detected level of the physical property and as a result affect the electricity flow . The decrease or increase in the electricity flow that passes through the variable resistor is what is read as a signal by Arduino board as an indicator of change in the sensed property:

Position Sensors/GPS

Distance Sensors

Motion Sensors/ Accelerometer

Tilt Sensors

Digital Compass

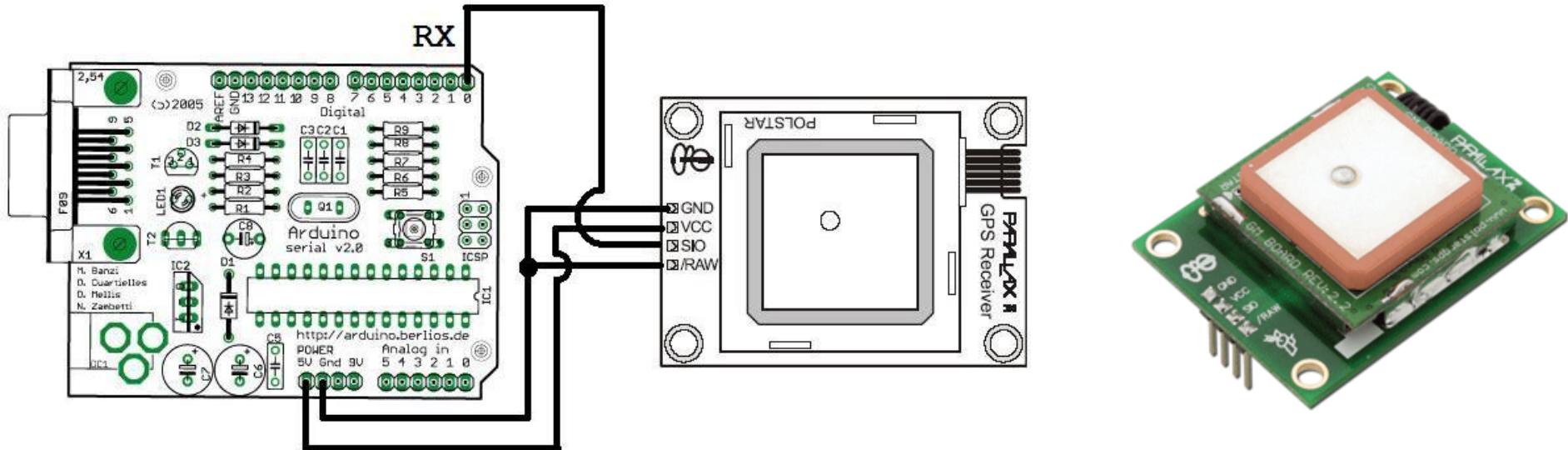
Bend Sensors

Input-Analog Sensors

- Pressure Sensors
- Tension Sensors
- Light Sensors
- Color Sensors
- Temperature Sensors
- Humidity Sensors
- Wind Sensor
- Gas and Chemical Sensors
- Skin Capacity Sensors
- Tactile sensors such as push button and Dials and Sliders
- Touch Sensors
- Magnetic Field Sensors
- RFID Tags and Readers/RFID read/write tags and Reader/Writers
- Motion state Mechanical Sensors

Input Devices-Location-GPS Sensor

Connecting a Parallax GPS module to the Arduino, and using Arduino code enables us to read information like date, time, location and satellites in view from the GPS Sensor





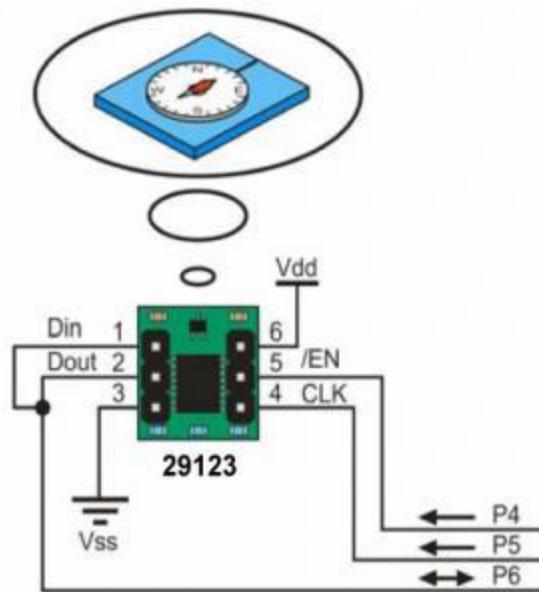
GPS Module and GPS Shield for Arduino

Adding GPS to your Arduino has never been easier. The EM406 plugs easily onto the shield, and you will be able to locate your exact position within a few meters. GPS also gives you amazingly accurate time! Spark fun website has a complete manual and example sketches for working with GPS modules

http://www.sparkfun.com/commerce/product_info.php?products_id=9898

Input Devices-Location-Digital Compass

Connecting a Parallax Hitachi HM55B Compass, you can get angle to north, as analog data.

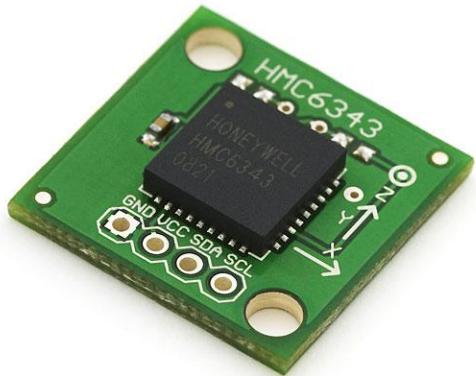




Hitachi H48C 3-Axis Accelerometer

The Hitachi H48C 3-Axis Accelerometer is an integrated module that can sense gravitational (g) force of $\pm 3g$ on three axes (X, Y, and Z). This sensor can be used for Tilt measurement, Multi-axis vibration measurement, Multi-axis movement/lack-of-movement measurements.

<http://www.parallax.com/Store/Sensors/AccelerationTilt/tabid/172/CategoryID/47>List/0/SortField/0/Level/a/ProductID/97/Default.aspx>



Compass Module with Tilt Compensation - HMC6343

The HMC6343 is a solid-state compass module with tilt compensation from Honeywell. What you get is a compass heading over I2C that stays the same even as you tilt the board. Solid-state (and many classic water based) compasses fail badly when not held flat. The tilt compensated HMC6343 is crucial for those real-world compass applications.

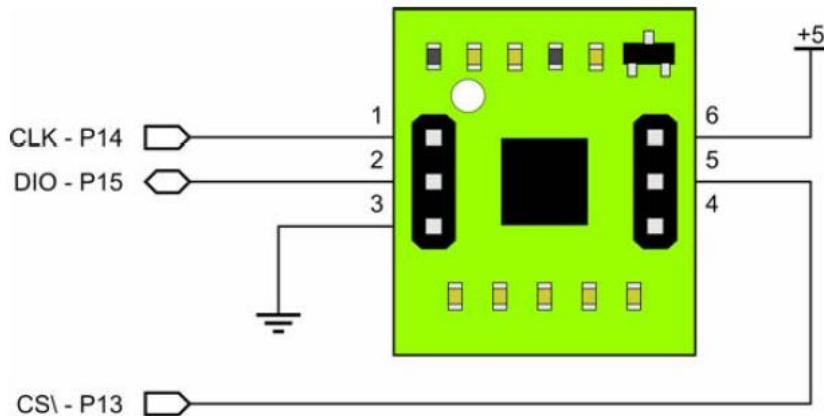
http://www.sparkfun.com/commerce/product_info.php?products_id=8656

<http://wiring.org.co/learning/libraries/hmc6343sparkfun.html>

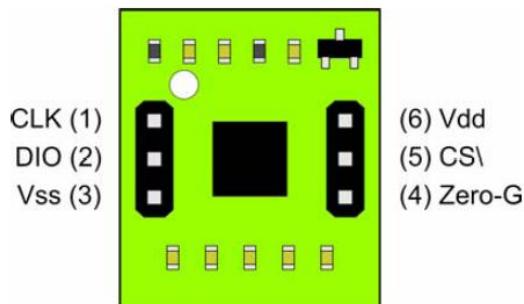
Input Devices-Orientation/Motion-Accelerometer

Parallax Hitachi H48C Tri-Axis Accelerometer is an integrated module that can sense gravitational (g) force of $\pm 3g$ on three axes (X, Y, and Z). So it helps you detect Movement, Speed and using time factor the distance covered in a motion in three directions

Figure 1. H48C Connections



- (1) CLK Synchronous clock input
- (2) DIO Bi-directional data to/from host
- (3) Vss Power supply ground (0v)
- (4) Zero-G "Free-fall" output; active-high
- (5) CS\ Chip select input; active-low
- (6) Vdd +5vdc



Input Devices-Orientation- Tilt Sensor

The Parallax Memsic 2125 is a low cost, dual-axis thermal accelerometer capable of measuring tilt, acceleration, rotation, and vibration with a range of ± 3 g.

Key Features of the Memsic 2125:

Measure 0 to ± 3 g on either axis

Fully temperature compensated over 0° to 70° C range

Simple, pulse output of g-force for X and Y axis

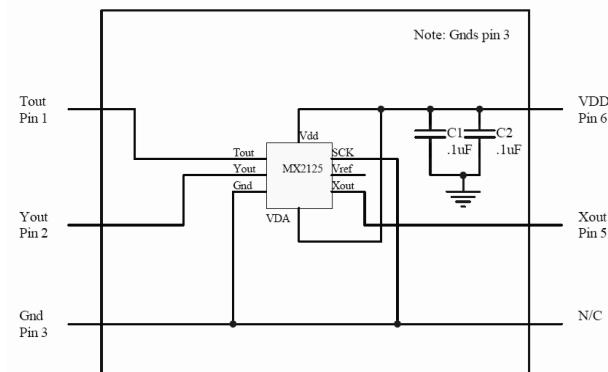
Analog output of temperature (TOut pin)

Low current 3.3 or 5V operation: less than 4 mA at 5 vdc

Dual-axis tilt sensing

Single-axis rotational position sensing

Movement/Lack-of-movement sensing

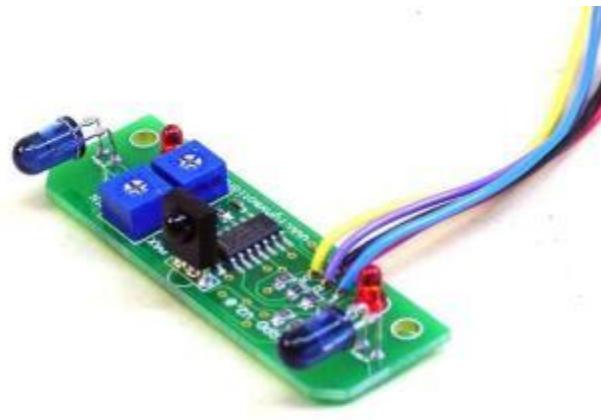


Input Devices-Distance-Range Finder

The PING range finder is an ultrasound sensor from Parallax able of detecting objects up to a 3 meter distance. The sensor comes with 3 pins, two are dedicated to power and ground, while the third one is used both as input and output.



Input Devices-Distance-IR Sensor /IR LED



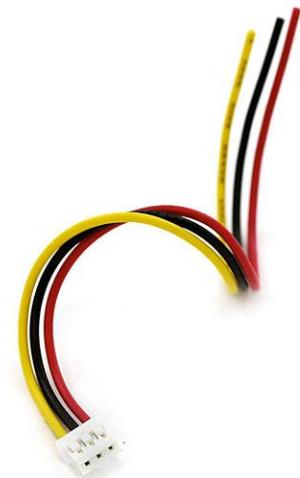
To make a simple IR distance sensor You can use a Panasonic pna4602m IR sensor and an IR led.

Hardware Requirements

Panasonic pna4602m IR sensor (1)

IR led (1)

220 ohm resistor (2)



Infrared Proximity Sensor Short Range - Sharp GP2D120XJ00F

Infrared proximity sensor made by Sharp. Part # GP2D120XJ00F has an analog output that varies from 3.1V at 3cm to 0.3V at 40cm with a supply voltage between 4.5 and 5.5VDC. The sensor has a Japanese Solderless Terminal (JST) Connector. We recommend purchasing the related pigtail below or soldering wires directly to the back of the module.

Infrared Proximity Sensor Long Range - Sharp GP2Y0A02YK0F

Infrared proximity sensor made by Sharp. Part # GP2Y0A02YK0F has an analog output that varies from 2.8V at 15cm to 0.4V at 150cm with a supply voltage between 4.5 and 5.5VDC. The sensor has a Japanese Solderless Terminal (JST) Connector. We recommend purchasing the related pigtail below or soldering wires directly to the back of the module. This sensor is great for sensing objects up to 5 feet away!

Infrared Proximity Sensor - Sharp GP2Y0A21YK

Infrared proximity sensor made by Sharp. Part # GP2Y0A21YK has an analog output that varies from 3.1V at 10cm to 0.4V at 80cm. The sensor has a Japanese Solderless Terminal (JST) Connector. We recommend purchasing the related pigtail below or soldering wires directly to the back of the module.

Infrared Sensor Jumper Wire - 3-Pin JST

Three pin JST connector with red, black, and yellow colors. 5 inch wire outs. This cable comes fully assembled as shown and connects directly to many different Sharp sensors. Quick and easy, this cable will save you 15-20 minutes over making your own wiring harness.

http://www.sparkfun.com/commerce/product_info.php?products_id=8959

http://www.sparkfun.com/commerce/product_info.php?products_id=8958

http://www.sparkfun.com/commerce/product_info.php?products_id=242

http://www.sparkfun.com/commerce/product_info.php?products_id=8733



Ultrasonic Range Finder - XL-Maxsonar EZ0-4

The XL series of MaxSonars are a super high-performance version of the easy-to-use sonar range finder from Maxbotix. The XL series of this sensor features higher resolution, longer range, higher power output and better calibration when compared to the LV version. We are extremely pleased with the size, quality, and ease of use of this little range finder. The sensor provides very accurate readings from 0 to 765cm (0 to 25.1ft) with 1cm resolution. This sensor can be powered with anywhere between 3.3 and 5VDC. Range information can be gathered through one of three methods - analog, serial, or PWM - all of which are active at the same time. The analog output will produce a voltage proportional to the measured distance, with a sensitivity of $(V_{cc}/1024)V/cm$. The serial interface is simple and formatted to RS-232, with voltages ranging from 0 to V_{cc} and terminal settings of 9600-8-N-1. Finally, the PWM pin outputs a pulse-width representation of the range with a scale factor of 58us/cm. The Maxsonar-XL series is offered in the EZ0, EZ1, EZ2, EZ3, and EZ4 versions, each with progressively narrower beam angles allowing the sensor to match the application. Please see beam width explanation below.

http://www.sparkfun.com/commerce/product_info.php?products_id=9491

http://www.sparkfun.com/commerce/product_info.php?products_id=9492

http://www.sparkfun.com/commerce/product_info.php?products_id=9493

http://www.sparkfun.com/commerce/product_info.php?products_id=9494

http://www.sparkfun.com/commerce/product_info.php?products_id=9495

<http://www.arduino.cc/playground/Main/MaxSonar>

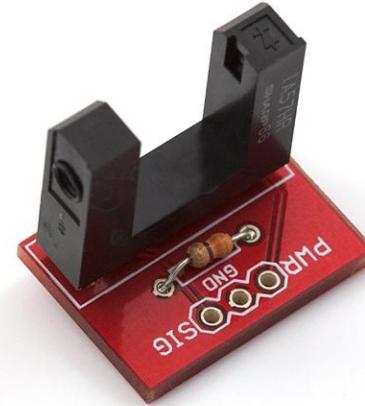
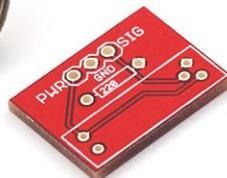


Photo Interrupter GP1A57HRJ00F

This sensor is composed of an infrared emitter on one upright and a shielded infrared detector on the other. By emitting a beam of infrared light from one upright to the other, the sensor can detect when an object passes between the uprights, breaking the beam. Used for many applications including optical limit switches, pellet dispensing, general object detection, etc. Gap width = 10mm . A breakout board is also available. This sensor is best for detecting boundary passage and presence in a particular range.

http://www.sparkfun.com/commerce/product_info.php?products_id=9299

http://www.sparkfun.com/commerce/product_info.php?products_id=9322



IR Reciever and Transmitters from Parallax

Infrared receiver with 38 kHz carrier frequency, for use with our IR Transmitter Assembly Kit. The IR Transmitter Kit consists of: IR LED (#350-00003), LED Standoff (#350-90000), and LED Light Shield (#350-90001).

IR Reciever and Transmitter from Spark Fun

Sparkfun IR LED is a very simple, clear infrared LED. These devices operate between 940-950nm and work well for generic IR systems including remote control and touch-less object sensing. 1.5VDC forward voltage and 50mA max forward current. Spark Fun IR Reciever Breakout is a very small infrared receiver. This receiver has all the filtering and 38kHz demodulation built into the unit. Simply point a IR remote at the receiver, hit a button, and you'll see a stream of 1s and 0s out of the data pin.

<http://www.parallax.com/Store/Sensors/ColorLight/tlid/175/CategoryID/50>List/0/SortField/0/Level/a/ProductID/177/Default.aspx>

<http://www.parallax.com/StoreSearchResults/tlid/768>List/0/SortField/4/ProductID/178/Default.aspx?txtSearch=IR+Transmitter+Assembly+Kit>

http://www.sparkfun.com/commerce/product_info.php?products_id=9349

http://www.sparkfun.com/commerce/product_info.php?products_id=8554



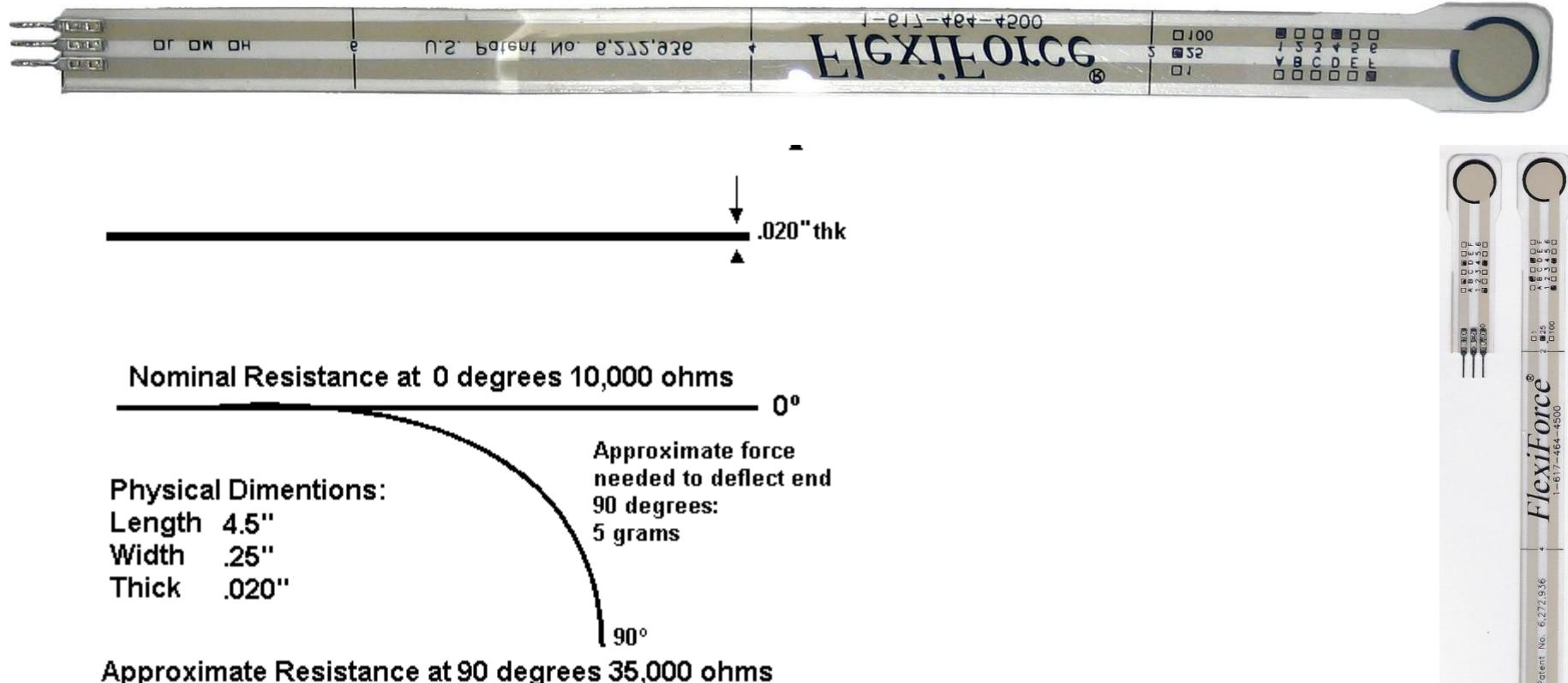
PIR Motion Detector

This is a simple to use motion sensor. Power it up and wait 1-2 seconds for the sensor to get a snapshot of the still room. If anything moves after that period, the 'alarm' pin will go low.

http://www.sparkfun.com/commerce/product_info.php?products_id=8630

<http://itp.nyu.edu/physcomp/sensors/Reports/PIRMotionSensor>

Input Devices-Bend/Pressure Sensor



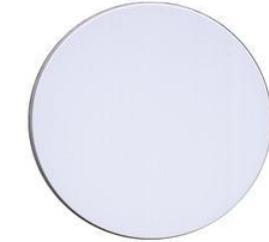
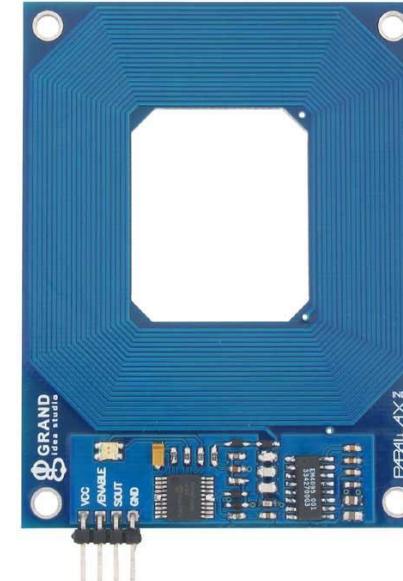
In a bend sensor, the bending angle is proportional to its resistance.

In Pressure sensor the amount of force on the button area is detected by sensor

Input Devices-Tagging-RFID Reader

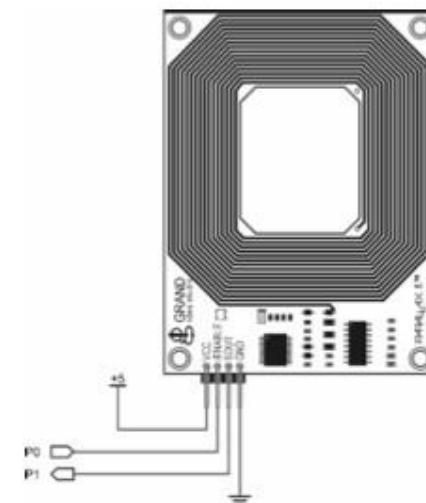
The Parallax Radio Frequency Identification (RFID) Reader Module is a solution to read passive RFID transponder tags from up to 4 inches away. The RFID Reader Module can be used in a wide variety of hobbyist and commercial applications, including access control, automatic identification, robotics, navigation, inventory tracking, and payment systems. It requires single +5VDC supply and has a Bi-color LED for visual indication of activity.

The reader is a sensor that is provoked by RFID tags, each passive RFID tag has a unique identification number that is read by the reader. Parallax RFID Disc Sticker is made of PVC and has adhesive on one side. This tag can be glued on flat and clean surfaces. It is 25mm in diameter, and 1.0mm in thickness, and is designed to work with RFID Reader. Each card is preprogrammed with a unique identification number.



Pin	Pin Name	Type	Function
1	VCC	P	System power, +5V DC input.
2	/ENABLE	I	Module enable pin. Active LOW digital input. Bring this pin LOW to enable the RFID reader and activate the antenna.
3	SOUT	O	Serial Out. TTL-level interface, 2400bps, 8 data bits, no parity, 1 stop bit.
4	GND	G	System ground. Connect to power supply's ground (GND) terminal.

Note: Type: I = Input, O = Output, P = Power, G = Ground





RFID Serial Reader Module

This RFID Reader Module (Serial version only) comes with two 54x85mm Rectangle Tags. Designed in cooperation with Grand Idea Studio, the Parallax Radio Frequency Identification (RFID) Reader Module is the first low-cost solution to read passive RFID transponder tags. The RFID Reader Module can be used in a wide variety of hobbyist and commercial applications, including access control, automatic identification, robotics, navigation, inventory tracking, payment systems, and car immobilization. There are a variety of transponder tags that come in different packages. Each tag has a specific range that is within 10% of the given distance for each type of tag. The reason for the 10% is due to environmental conditions and RFID modules.

<http://www.parallax.com/StoreSearchResults/tabid/768/txtSearch/RFID/List/0/SortField/4/ProductID/114/Default.aspx>

<http://www.parallax.com/StoreSearchResults/tabid/768/txtSearch/RFID/List/0/SortField/4/ProductID/115/Default.aspx>

<http://www.parallax.com/StoreSearchResults/tabid/768/txtSearch/RFID/List/0/SortField/4/ProductID/116/Default.aspx>

<http://www.parallax.com/Store/Accessories/CommunicationRF/tabid/161/ProductID/688>List/0/Default.aspx?SortField=ProductName,ProductName>



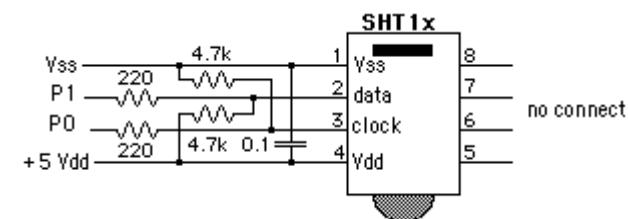
RFID Serial Read/Write Module

Designed in cooperation with Grand Idea Studio (www.grandideastudio.com), the Parallax Radio Frequency Identification (RFID) Read/Write Module provides a low-cost solution to read and write passive RFID transponder tags up to 3 inches away. The RFID transponder tags provide a unique serial number and can store up to 116 bytes of user data, which can be password protected to allow only authorized access. The RFID Read/Write Module can be used in a wide variety of hobbyist and commercial applications, including access control, user identification, robotics navigation, inventory tracking, payment systems, car immobilization, and manufacturing automation. It is Read/Write compatible only with the RFID R/W 54mm x 85mm Rectangle Tag. It is Read compatible with all RFID tags sold by Parallax.

<http://www.parallax.com/Store/Accessories/CommunicationRF/tabid/161/CategoryID/36/List/0/SortField/0/catpageindex/2/Level/a/ProductID/688/Default.aspx>
<http://www.parallax.com/Store/Accessories/CommunicationRF/tabid/161/ProductID/689>List/0/Default.aspx?SortField=ProductName,ProductName>

Input Devices-Environment-Humidity/Temperature Sensor

Parallax Sensirion Temperature/Humidity Sensor is a smart sensor for both humidity and temperature. Humidity is notoriously difficult to measure. If you're interested in the details see Tracy Allen's web site (EME Systems) for his discussion. The Sensirion SHT1x addresses many of these issues head on., and All that Arduino has to do is read out the humidity and temperature values through the two-wire digital serial interface. The only math required is a simple scale and offset. The SHT1x is factory calibrated so that it returns temperature with a resolution of 0.01 degrees Celsius and relative humidity with a resolution of 0.03 percent. The accuracy is better than most other sensors too.





Gas Sensors

LPG Gas Sensor - MQ-6 is a simple-to-use liquefied petroleum gas (LPG) sensor, suitable for sensing LPG (composed of mostly propane and butane) concentrations in the air. The MQ-6 can detect gas concentrations anywhere from 200 to 10000ppm with a very high sensitivity and fast response.

Carbon Monoxide Sensor - MQ-7 is a simple-to-use Carbon Monoxide (CO) sensor, suitable for sensing CO concentrations in the air. The MQ-7 can detect CO concentrations anywhere from 20 to 2000ppm with a very high sensitivity and fast response.

Alcohol Gas Sensor - MQ-3 is suitable for detecting alcohol concentration on your breath, just like your common breathalyzer. It has a high sensitivity and fast response time. Sensor provides an analog resistive output based on alcohol concentration.

Methane CNG Gas Sensor - MQ-4 is a simple-to-use compressed natural gas (CNG) sensor, suitable for sensing natural gas (composed of mostly Methane [CH4]) concentrations in the air. The MQ-4 can detect natural gas concentrations anywhere from 200 to 10000ppm with a very high sensitivity and fast response.

http://www.sparkfun.com/commerce/product_info.php?products_id=9405

http://www.sparkfun.com/commerce/product_info.php?products_id=9404

http://www.sparkfun.com/commerce/product_info.php?products_id=9403

http://www.sparkfun.com/commerce/product_info.php?products_id=8880

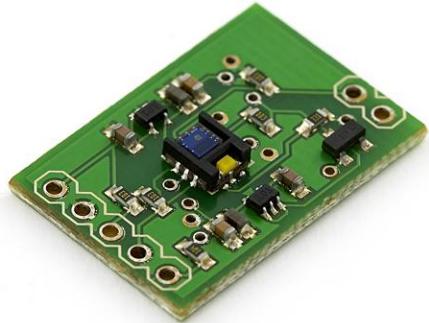


INSPEED Vortex Wind Sensor

Rugged wind sensor handles speeds from 5 to over 125 mph. Reed switch/magnet provides one pulse per rotation.

http://www.inspeed.com/anemometers/Vortex_Wind_Sensor.asp

<http://community.pachube.com/arduino/anemometer>

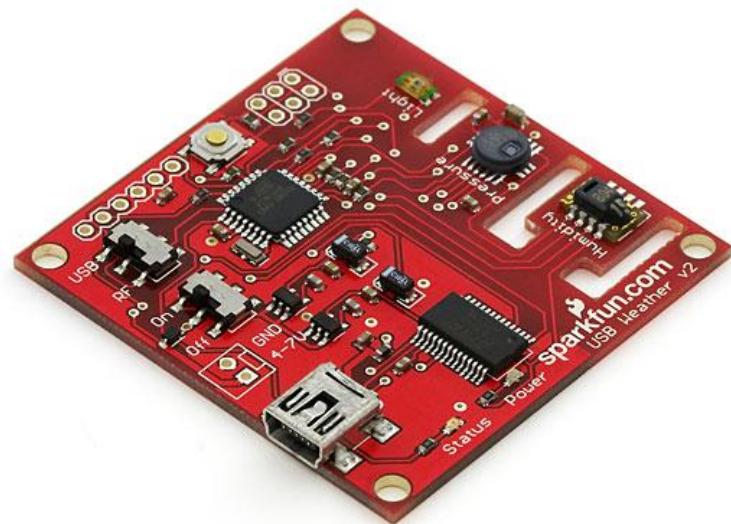


Light Color Sensor Evaluation Board

The ADJD-S371 is a great little sensor. This evaluation board provides all the necessary support circuitry to actually play with the sensor! It is capable of sensing light color that is reflected off surfaces.

http://www.sparkfun.com/commerce/product_info.php?products_id=8663

<http://interactive-matter.org/2008/08/tinkering-with-adjd-s371-q999/>



USB Weather Board

We take the sensitive SCP1000 barometric pressure sensor, add the TEMT6000 ambient light sensor, match it with a sensitive SHT15 humidity sensor, and we give you weather over USB which allows you to immediately tell what the current pressure, humidity, ambient light level, and temperature is. Graphed over time you can watch weather fronts move in and the rain come down. Serial output is a single visible ASCII string at 9600bps. There is a footprint and switch for 'RF'. This unit can be powered from our large solar cell and data can be transmitted via our BlueSMiRF wireless modem! All you need now is a greenhouse to monitor.

http://www.sparkfun.com/commerce/product_info.php?products_id=9800

<http://wiring.org.co/learning/libraries/serialweather.html>



Light Intensity to Frequency IC

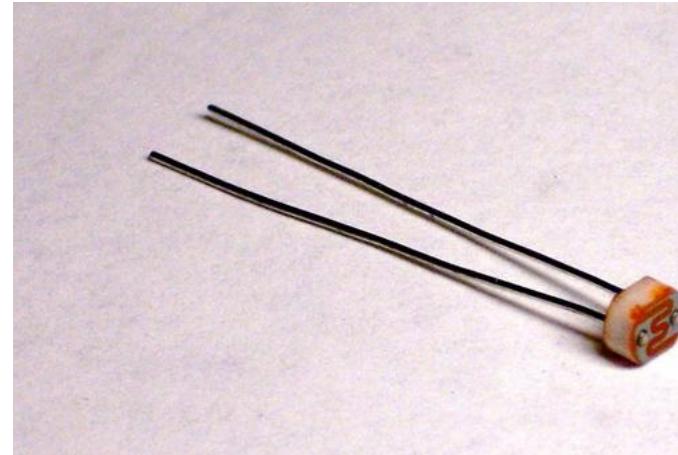
a light sensing circuit wrapped up in a clear plastic casing. This neat little device will convert irradiance (the light energy on the surface of the sensor) into frequency. Working with a simple input concept like a frequency means that we won't have to build any extra circuitry to get the full range of information from the circuit, and having an accurate measure of radiance means that we'll be able to convert easily over to illuminance, which is how the light looks to us.

http://www.sparkfun.com/commerce/product_info.php?products_id=8940

<http://roamingdrone.wordpress.com/2008/11/13/arduino-and-the-taos-tsl230r-light-sensor-getting-started/>

Input Devices-Environment-Light Sensor

A light sensor is a variable resistor that detects levels of light intensity. It is an analog sensor and is the one that we are going to use in this class to learn conceptual framework and actual techniques of using sensors to sense physical properties of the environment



Input Devices-Environment-Color Sensor

With Parallax Color sensor you can detect colors in the environment





Push button Switch

Push button switches are perfect as tactile sensors for detecting whether a part is clicked or pushed.

http://www.sparkfun.com/commerce/product_info.php?products_id=97

http://www.sparkfun.com/commerce/product_info.php?products_id=9190



Five way Tactile Switch

It is basically five push buttons packaged as one interface. Great for detection of push and its orientation in five directions with one sensor.

http://www.sparkfun.com/commerce/product_info.php?products_id=10063



Linear Potentiometers

Linear potentiometers are variable resistors that change resistance based on the magnitude of linear disposition of their handle within a limited physical range. They can be used for detecting change in position of kinetic parts within a spatial composition within a defined limited range.

http://www.sparkfun.com/commerce/product_info.php?products_id=9119

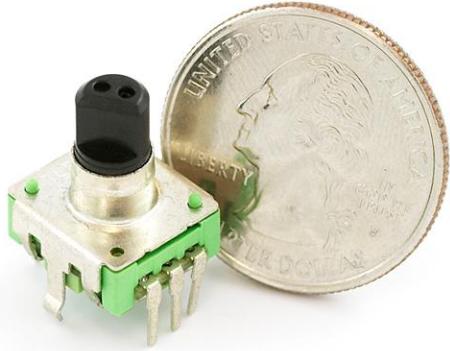


Logarithmic or Linear Rotary Potentiometers

A rotary potentiometer is an angular measuring device with limited rotation. Turning the pot changes the resistance of the potentiometer. The resistance changes. Connect VCC to an outer pin, GND to the other, and the center pin will have a voltage that varies from 0 to VCC depending on the rotation of the pot. Rotary potentiometers come in linear or logarithmic variations. In linear rotary potentiometers the angle of rotation has a linear relation to how the resistance of the potentiometer changes. In logarithmic potentiometers this relation is not linear and changes exponentially.

http://www.sparkfun.com/commerce/product_info.php?products_id=9940

http://www.sparkfun.com/commerce/product_info.php?products_id=9939



Rotary Encoder

A rotary or "shaft" encoder is an angular measuring device. It is used to precisely measure rotation of motors or to create wheel controllers (knobs) that can turn infinitely (with no end stop like a potentiometer has). Some of them are also equipped with a pushbutton when you press on the axis (like the ones used for navigation on many music controllers). They come in all kinds of resolutions, from maybe 12 to at least 1024 steps per revolution. This is a 12-step rotary encoder with a nice 'clicking' feel. It's breadboard friendly, and has a pretty handy select switch (by pushing in on the knob). The encoder is different from a potentiometer in that an encoder has full rotation without limits. You can tell how much and in which direction the encoder has been turned. Incorporating the temporal aspect you can also calculate speed of rotation based on how long it takes the rotary Encoder to move from one step to the next.

http://www.sparkfun.com/commerce/product_info.php?products_id=9117#

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



Thumb Joystick

It is basically a combination of two rotary potentiometers to detect direction in two axes perpendicular to each other and a push button to detect push action. The result is a low-tech joystick input device.

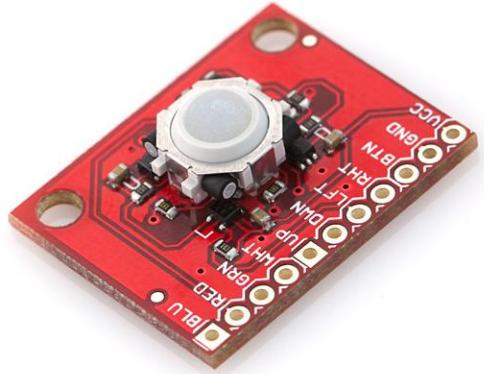
http://www.sparkfun.com/commerce/product_info.php?products_id=9032



Thumb Slide Joystick

This is another interesting Joystick that allows to send information regarding direction in x,y directions. It has an interesting 'slide' feeling

http://www.sparkfun.com/commerce/product_info.php?products_id=9032



Blackberry Trackballer Breakout

This is another easy to use tactile interface. The four spindles on the trackball have a tiny circular magnet at the end; each of these are paired with an SMD hall effect sensor, which are used to measure up, down, left and right movements of the trackball. An SMD momentary switch is placed under the trackball to give you a select switch. The BTN line will be pulled low when the switch is pressed. Also included on the Trackballer are 4 LEDs: red, blue, green and white. These can be powered to light the clear trackball up any color you can imagine. Regulated, 2.5-5.25VDC power must be provided to power the Hall sensors. The hall-effect sensors and trackball combo are surprisingly sensitive. A slight roll of the trackball creates multiple high/low transitions on the four axis pins, easily picked up by any microcontroller. A 360° rotation of the trackball, along a single axis, will result in about 9 high/low transitions.

http://www.sparkfun.com/commerce/product_info.php?products_id=9320



Reed Switch and Hall effect Sensor for Sensing Magnetic Field

When a reed switch is exposed to a magnetic field, the two ferrous materials inside the switch pull together and the switch closes. When the magnetic field is removed, the reeds separate and the switch opens. This makes for a great non-contact switch. This switch can carry up to 1.2A. In a hall effect sensor, holding a magnet near the sensor will cause the output pin to toggle. This makes for a robust presence sensor. A reed sensor also works nicely, but can be limited by the glass encapsulation and size. A hall effect sensor is much smaller, but can handle less current than a reed switch.

http://www.sparkfun.com/commerce/product_info.php?products_id=8642

http://www.sparkfun.com/commerce/product_info.php?products_id=9312



Scientific 2" Flexible Stretch Sensor

Infrared receiver with 38 kHz carrier frequency, for use with our IR Transmitter Assembly Kit. The IR Transmitter Kit consists of: IR LED (#350-00003), LED Standoff (#350-90000), and LED Light Shield (#350-90001).

IR Reciever and Transmitter from Spark Fun

Sparkfun IR LED is a very simple, clear infrared LED. These devices operate between 940-950nm and work well for generic IR systems including remote control and touch-less object sensing. 1.5VDC forward voltage and 50mA max forward current. Spark Fun IR Reciever Breakout is a very small infrared receiver. This receiver has all the filtering and 38kHz demodulation built into the unit. Simply point a IR remote at the receiver, hit a button, and you'll see a stream of 1s and 0s out of the data pin.

<http://www.robotshop.com/images-scientific-2inch-stretch-sensor-2.html>

Mechanical/Electrical Circuits

What is Electricity?

It is the flow of electrons in a medium



Current (I) is the number of electrons that pass through the medium per second

$$1 \text{ A (Ampere)} = 6.25 \times 10^{18} \text{ e/sec}$$

Voltage

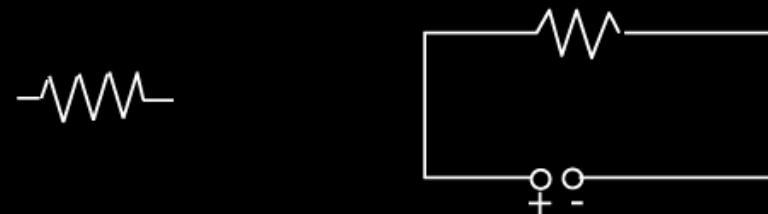
Is the potential to move electrons



Voltage is measured in V (Volts)

Resistor

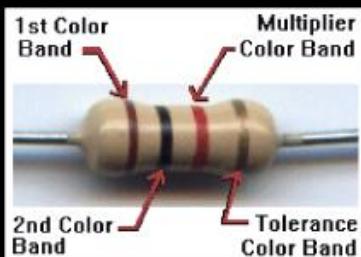
A device that slows down the flow of electrons



Resistance is measured in Ohms (Ω)

Resistor Color Coding

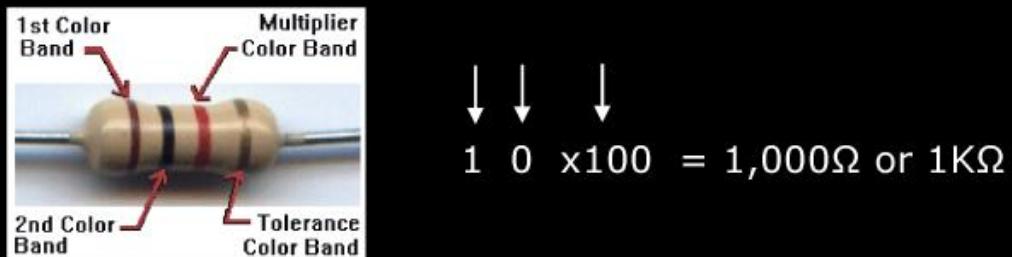
To identify the resistance we use a code system: we position them so that the gold strip (or band) is on the right side and then we measure:



Color	First Strip	Second Strip	Third Strip
Black	0	0	x1
Brown	1	1	x10
Red	2	2	x100
Orange	3	3	x1,000
Yellow	4	4	x10,000
Green	5	5	x100,000
Blue	6	6	x1,000,000
Purple	7	7	
Gray	8	8	
White	9	9	

Resistor Color Coding

To identify the resistance we use a code system: we position them so that the gold strip (or band) is on the right side and then we measure:



Color	First Strip	Second Strip	Third Strip
Black	0	0	x1
Brown	1	1	x10
Red	2	2	x100
Orange	3	3	x1,000
Yellow	4	4	x10,000
Green	5	5	x100,000
Blue	6	6	x1,000,000
Purple	7	7	
Gray	8	8	
White	9	9	

Scale

In practical breadboard applications current is measured in mA and resistance in Ohms, K Ohms and M Ohms.

1 Giga	= 1,000,000,000 (billion)
1 Mega	= 1,000,000 (million)
1 Kilo	= 1,000
1 milli	= 1/1,000
1 micro	= 1/1,000,000 (millionth)
1 pico	= 1/1,000,000,000 (billionth)

Ohm's Law

Voltage = Current x Resistance

$$V = I \times R \quad \text{or} \quad I = V/R \quad \text{or} \quad R = V/I$$

V I R

Ohm's Law

Voltage = Current x Resistance

$$V = I \times R \quad \text{or} \quad I = V/R \quad \text{or} \quad R = V/I$$

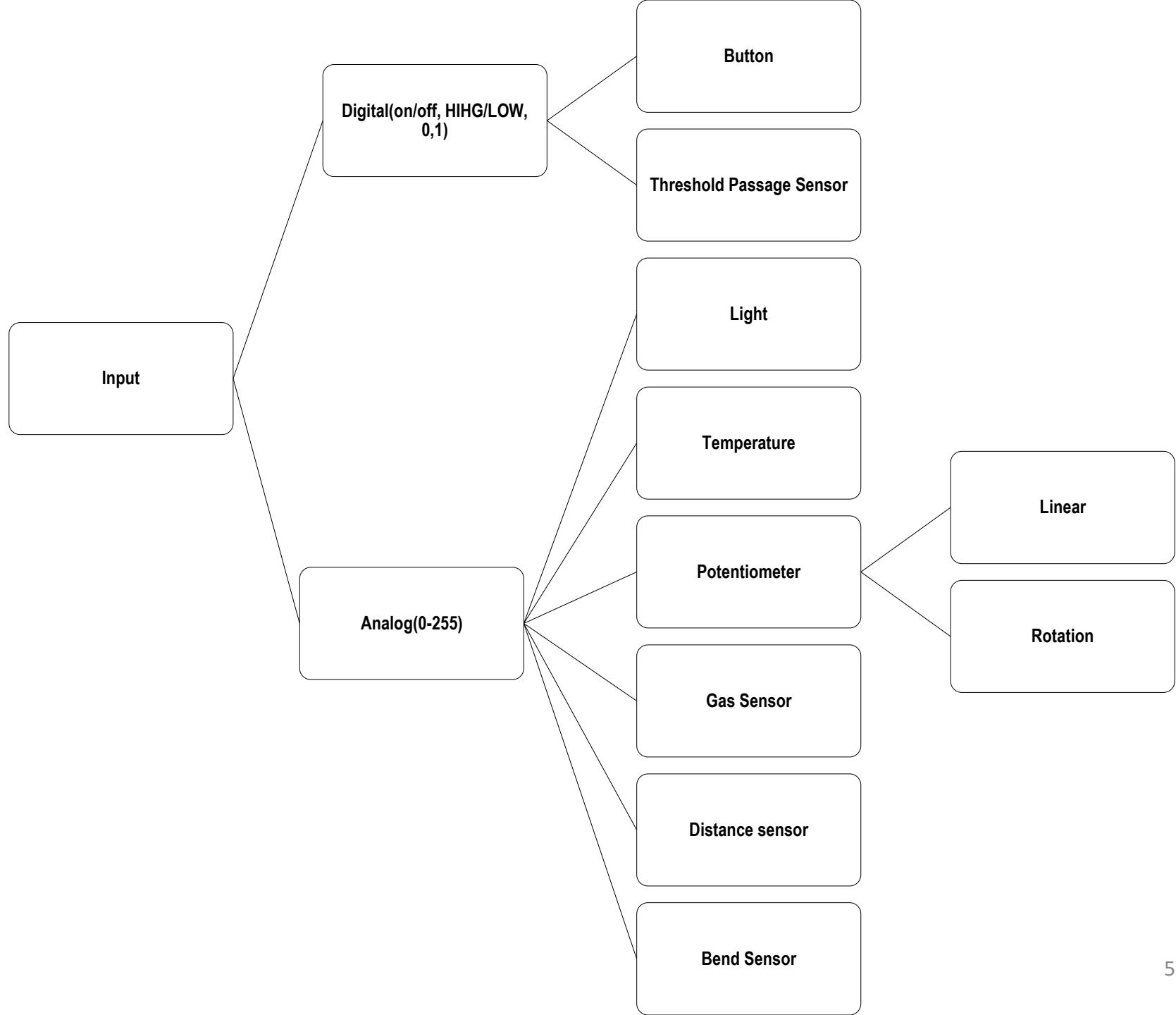


Ohm's Law

Voltage = Current x Resistance

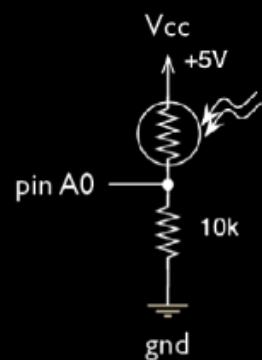
$$V = I \times R \quad \text{or} \quad I = V/R \quad \text{or} \quad R = V/I$$





Photocell

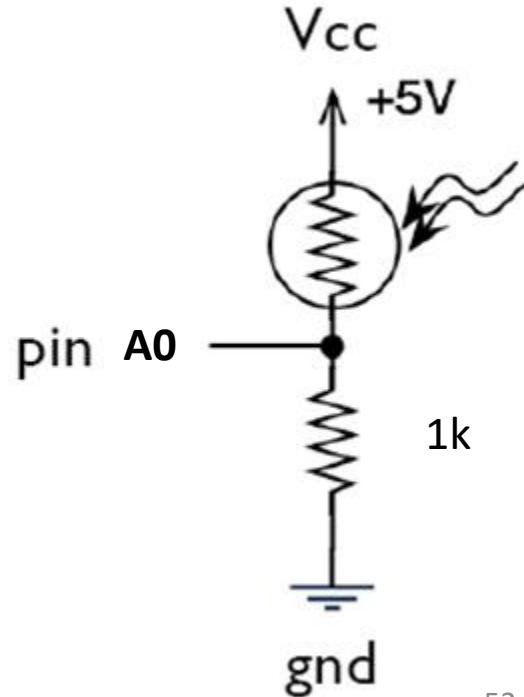
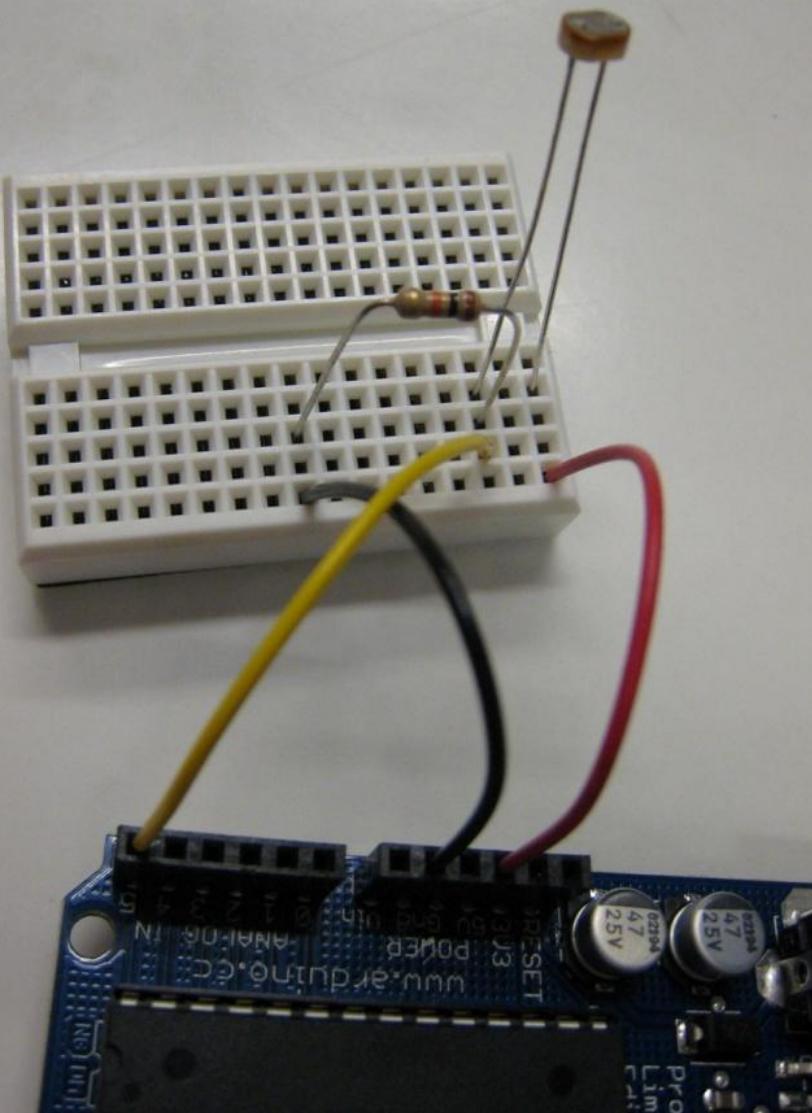
A photocell is a variable resistor (potentiometer)

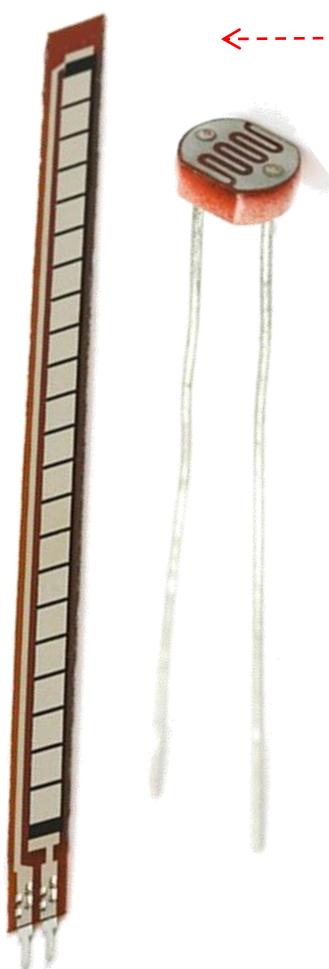


Arduino- Analog Input - Photocell

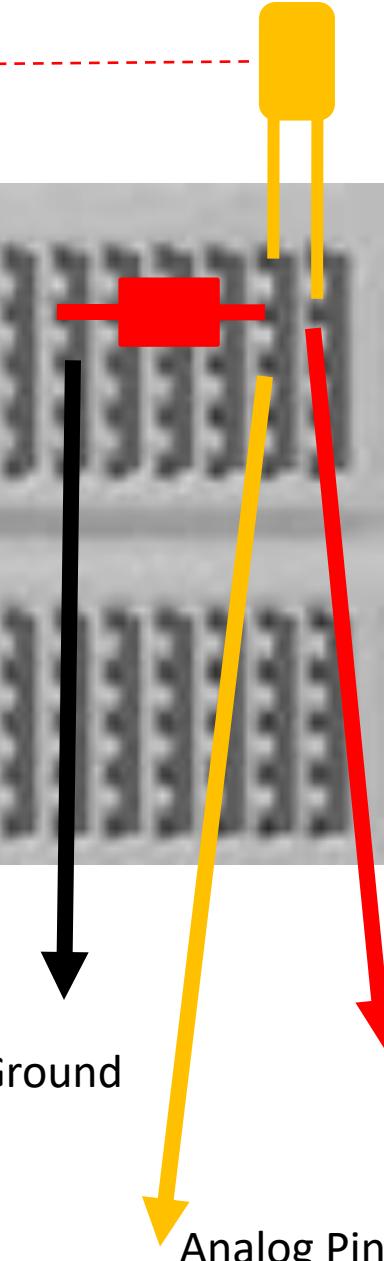
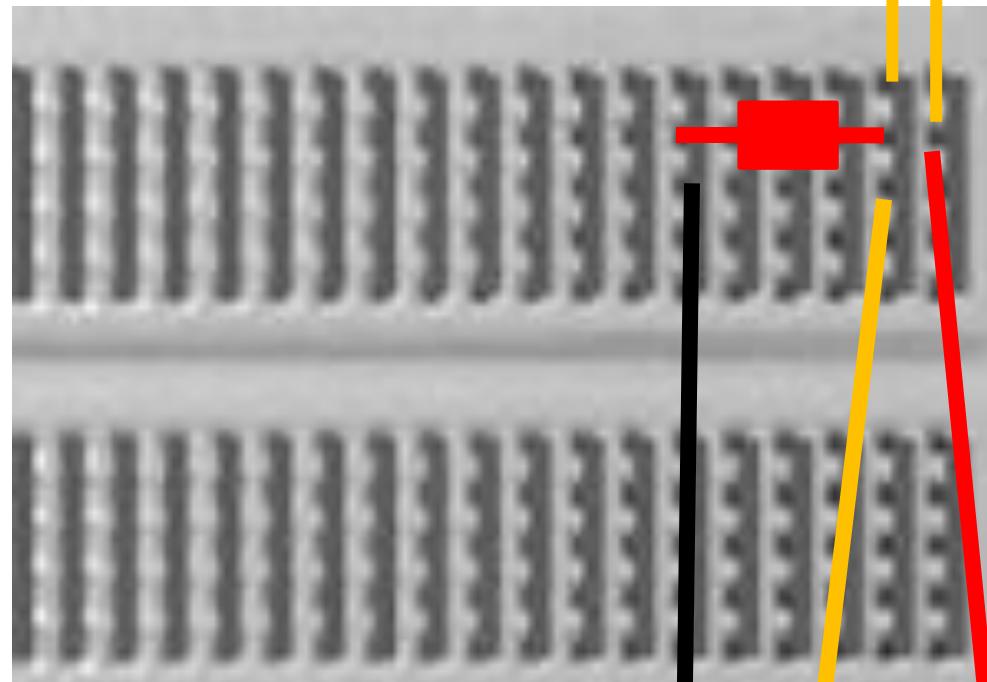
Connect the photocell in the following configuration (in accordance to the diagram):

As you know A photocell detects the level of light intensity. Light intensity can be at many levels, so it is an analog data type, as a result the photocell should be connected to one of the analog pins.





Two-legged sensor

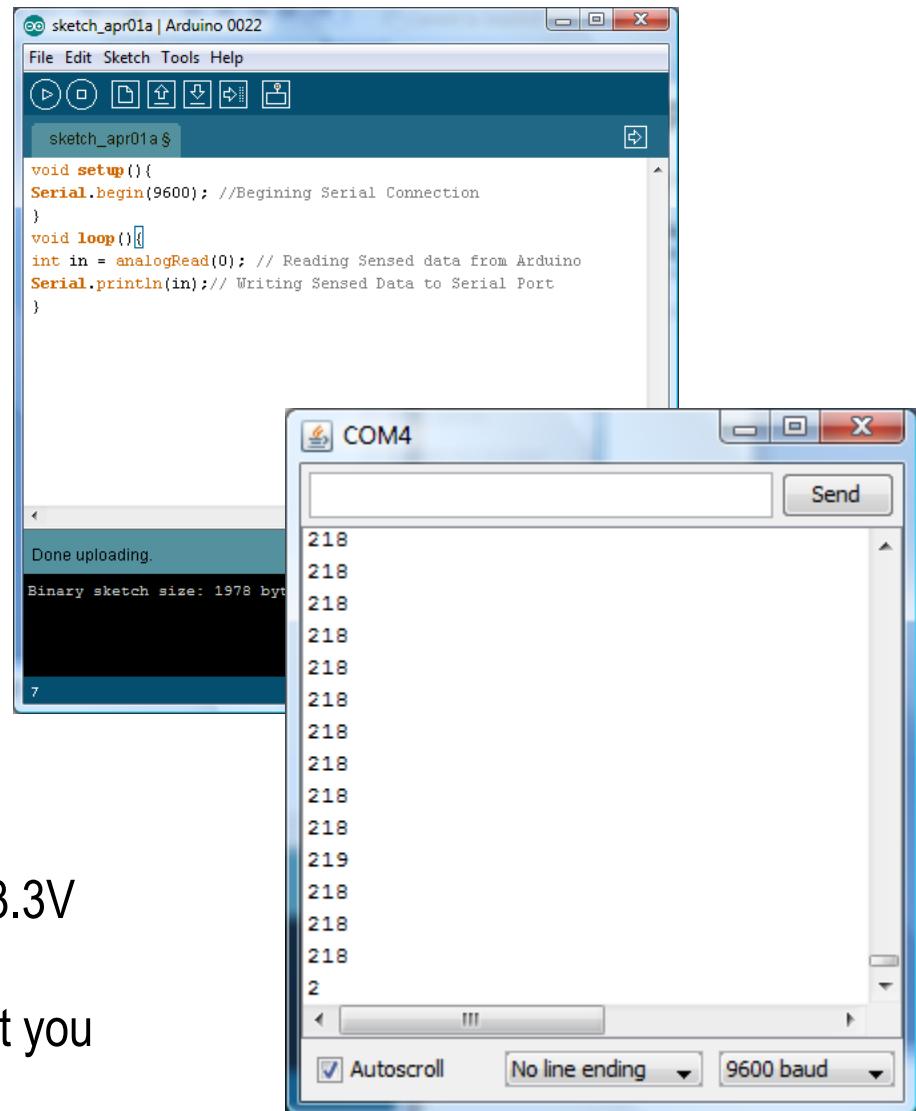


**The bend sensor, another two-legged sensor, is connected to Arduino like the photocell as well

Connecting Photocell and other two-legged sensors

Arduino- Analog Input - Photocell

```
void setup(){  
Serial.begin(9600); //Beginning Serial Connection  
}  
void loop(){  
int in = analogRead(0); // Reading Sensed data from Arduino  
Serial.println(in); // Writing Sensed Data to Serial Port  
}
```



1. While running you can see the values of the photocell by pressing the Serial Monitor button
2. Depending on powering the sensor with 5V or 3.3V you can change the sensitivity of the sensor
3. Depending on the magnitude of the resistor that you use you can change the sensitivity of the sensor



Output

```
main:  
  Send Data  
  Goto main
```



Input

```
main:  
  Get Data  
  Goto main
```



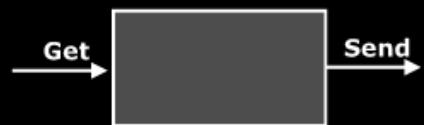
Output

```
main:  
  Send Data  
  Goto main
```



Input

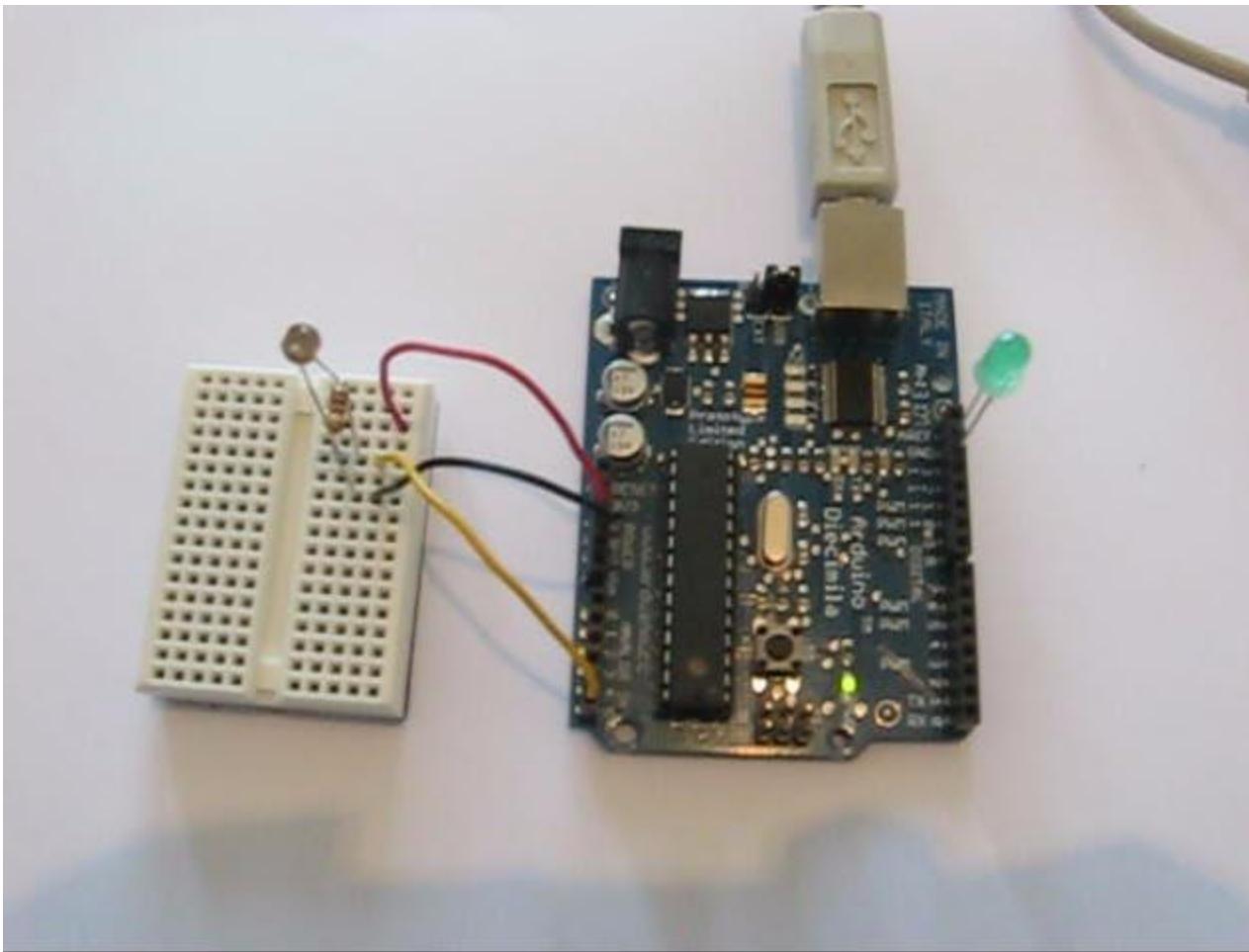
```
main:  
  Get Data  
  Goto main
```



Responsive

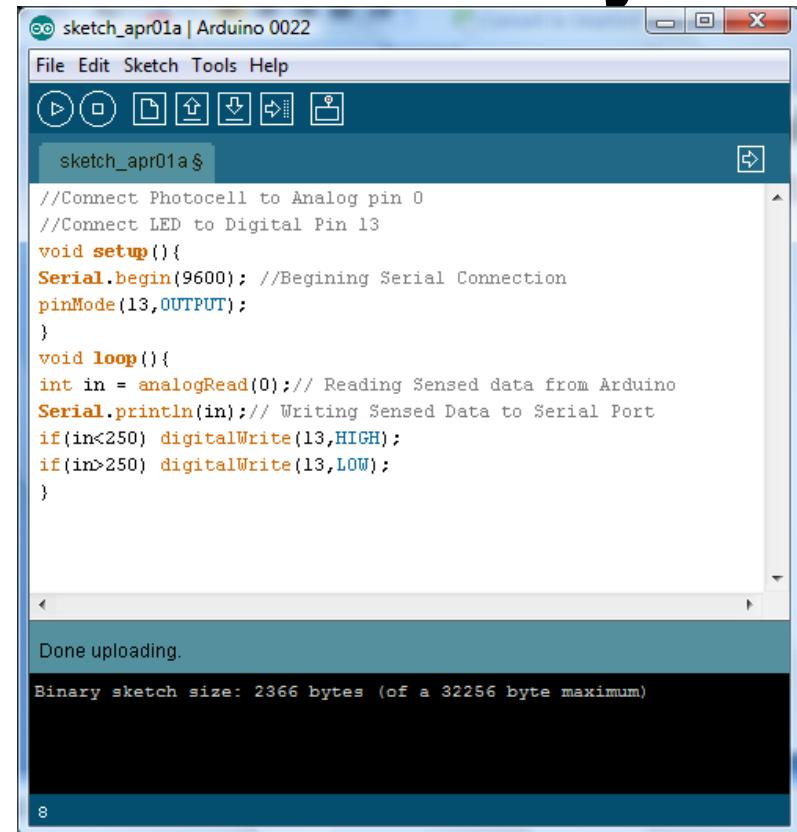
```
main:  
  Get Data  
  if condition then  
    Send Data  
  Goto main
```

Arduino- Analog Input/Digital Output- Your First Interactive System



Arduino- Analog Input/Digital Output- Your First Interactive System

```
//Connect Photocell to Analog pin 0
//Connect LED to Digital Pin 13
void setup(){
Serial.begin(9600); //Beginning Serial Connection
pinMode(13,OUTPUT);
}
void loop(){
int in = analogRead(0); // Reading Sensed data from Arduino
Serial.println(in); // Writing Sensed Data to Serial Port
if(in<250) digitalWrite(13,HIGH);
if(in>250) digitalWrite(13,LOW);
}
```

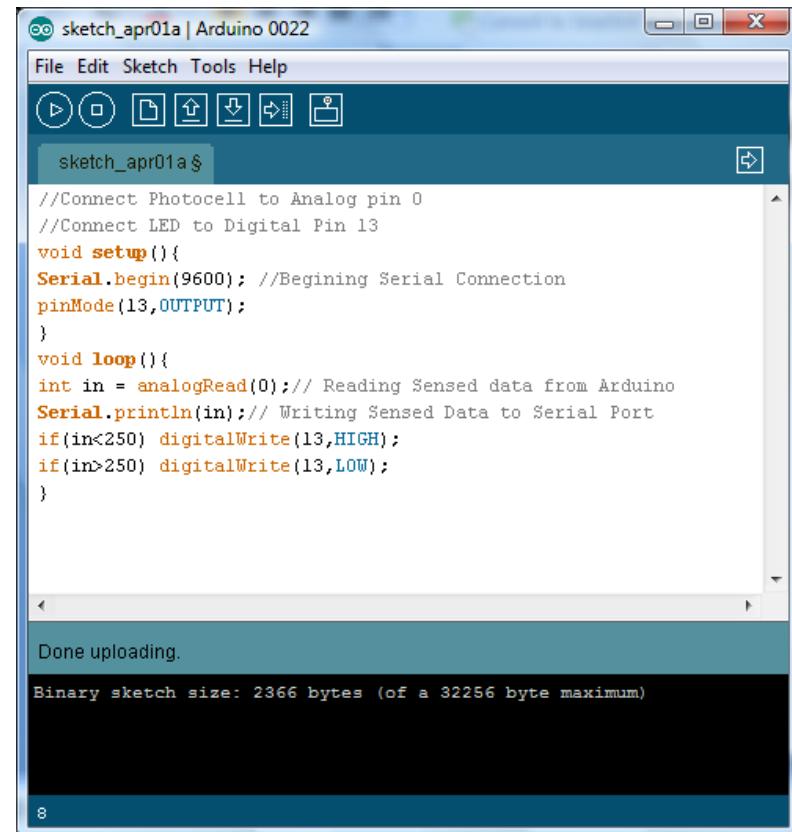


Depending on lighting condition of the environment you may need to change the critical threshold – 250 at this point- to correspond to the lighting condition

Arduino- Analog Input/Digital Output- Your First Interactive System

Here in the two IF statements we are introducing a condition for a threshold, stating that if the read light intensity is less than the specified threshold(In this case 250) then it means that the environment is dark So we are asking the system to turn the light on. And if the light intensity is above the specified threshold it means that the environment has enough light already so we are asking the system to turn the light off.

The question always is how to specify this threshold. For now run the code once and monitor the sensed values in the Serial port console to see what is the maximum and minimum and then choose the threshold approximately midway.



The screenshot shows the Arduino IDE interface with a sketch titled "sketch_apr01a". The code is as follows:

```
//Connect Photocell to Analog pin 0
//Connect LED to Digital Pin 13
void setup(){
  Serial.begin(9600); //Beginning Serial Connection
  pinMode(13,OUTPUT);
}
void loop(){
  int in = analogRead(0);// Reading Sensed data from Arduino
  Serial.println(in);// Writing Sensed Data to Serial Port
  if(in<250) digitalWrite(13,HIGH);
  if(in>250) digitalWrite(13,LOW);
}
```

The status bar at the bottom indicates "Done uploading." and "Binary sketch size: 2366 bytes (of a 32256 byte maximum)".

Arduino- Analog Input/Digital Output- System Calibration

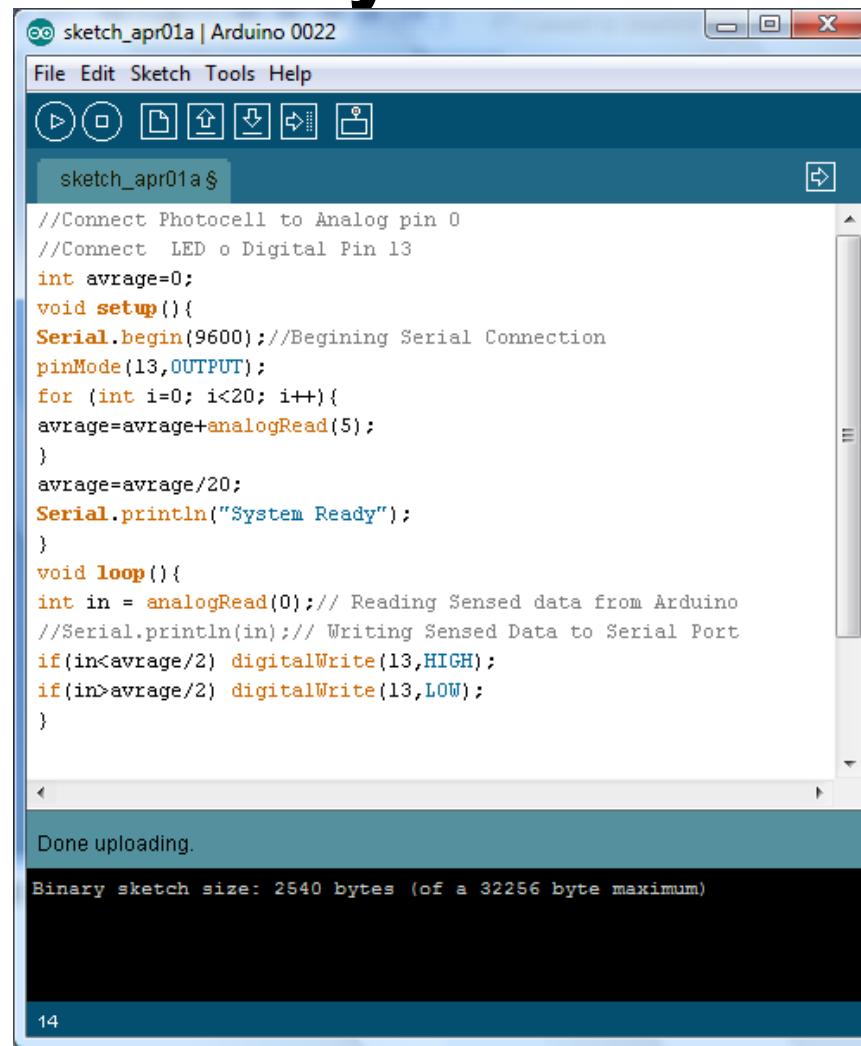
Most of the time we choose the decisive threshold of the system based on the status of the environmental properties. Changing the environment of the system results in shifts in maximum and minimum of the range of the sensed data.

As a result when changing the environment of the system we need to calibrate the system and reset the threshold based on the new range of the sensed data.

The other option is to write the code in a way that the system calibrates itself automatically.

Arduino- Analog Input/Digital Output- System Calibration

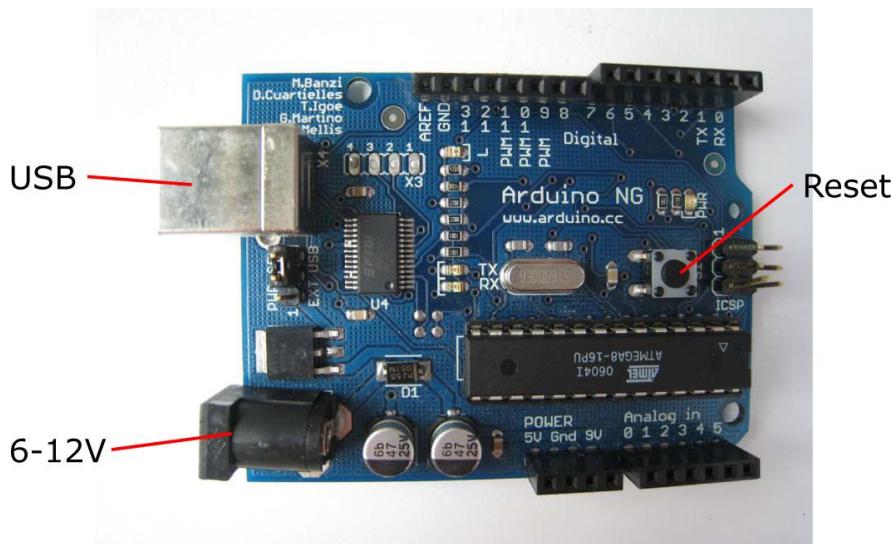
```
//Connect Photocell to Analog pin 0
//Connect LED o Digital Pin 13
int avrage=0;
void setup(){
Serial.begin(9600);//Beginning Serial Connection
pinMode(13,OUTPUT);
for (int i=0; i<20; i++){
avrage=avrage+analogRead(0);
}
avrage=avrage/20;
Serial.println("System Ready");
}
void loop(){
int in = analogRead(0);// Reading Sensed data from Arduino
//Serial.println(in);// Writing Sensed Data to Serial Port
if(in<avrage/2) digitalWrite(13,HIGH);
if(in>avrage/2) digitalWrite(13,LOW);
}
```



Arduino- Analog Input/Digital Output- System Calibration

This way you do not need to calibrate the system manually. You can design and implement your responsive system off-site, install it on site and using the reset button on the board reset the system to calibrate to the new environment.

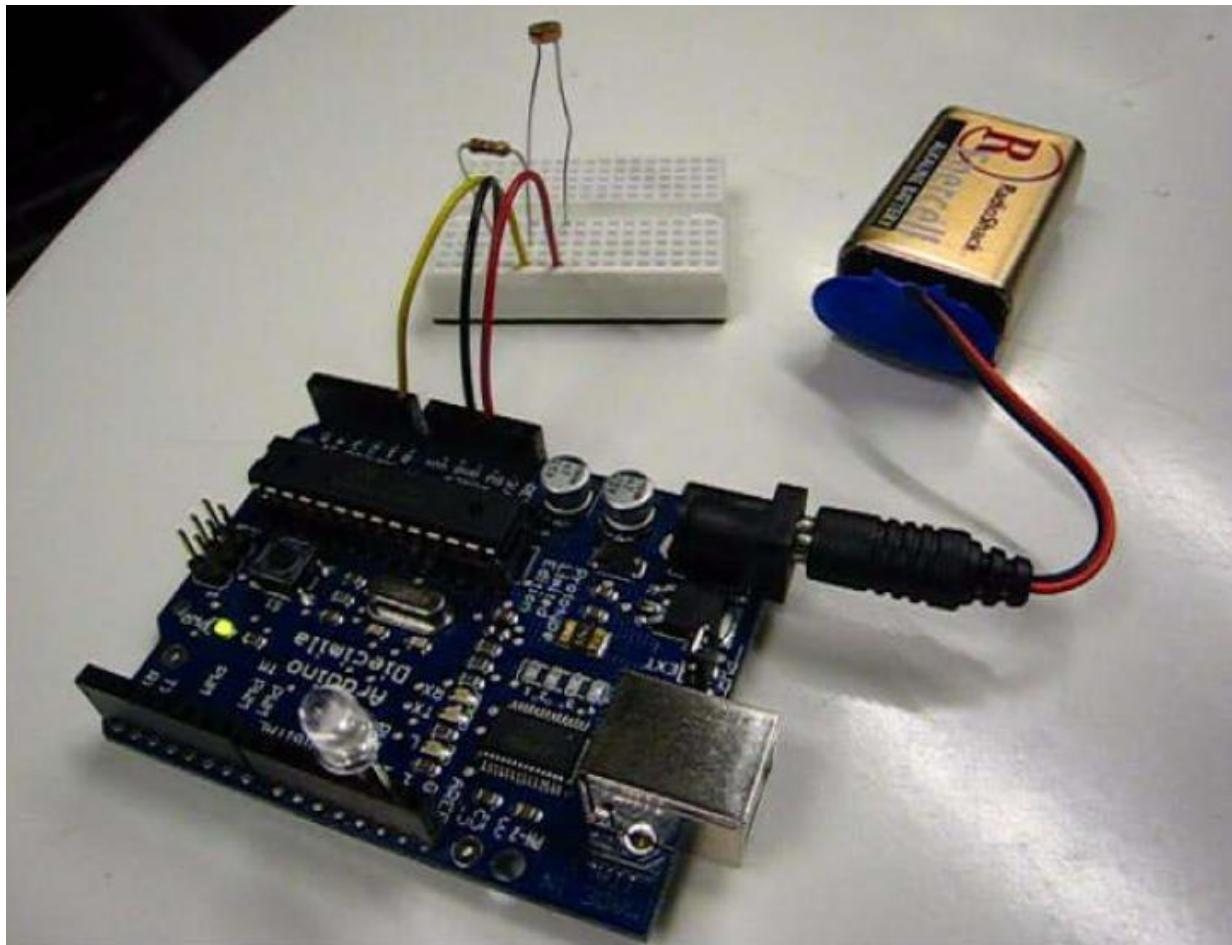
Holding down the reset button for five seconds will do the job.



The screenshot shows the Arduino IDE interface. The top window displays the code for 'S18_AnalogInput_PhotocellControlLED_PhotocellCalibration'. The code reads analog data from pin 5, calculates an average over 20 samples, and then prints 'System Ready' to the serial monitor. The bottom window shows the serial monitor with the text 'System Ready' displayed at 9600 baud.

```
//Connect Photocell to Analog pin 5
//Connect Photocell to Digital Pin 13
int avrage=0;
void setup(){
Serial.begin(9600);//Begining Serial Connection
pinMode(13,OUTPUT);
for (int i=0; i<20; i++){
avrage=avrage+analogRead(5);
}
avrage=avrage/20;
Serial.println("System Ready");
}
void loop(){
int in = analogRead(5);// Reading Sensed data from Arduino
//Serial.println(in); // Writing Sensed Data to Serial Port
if(in<avrage/2) digitalWrite(13,HIGH);
if(in>avrage/2) digitalWrite(13,LOW);
}
```

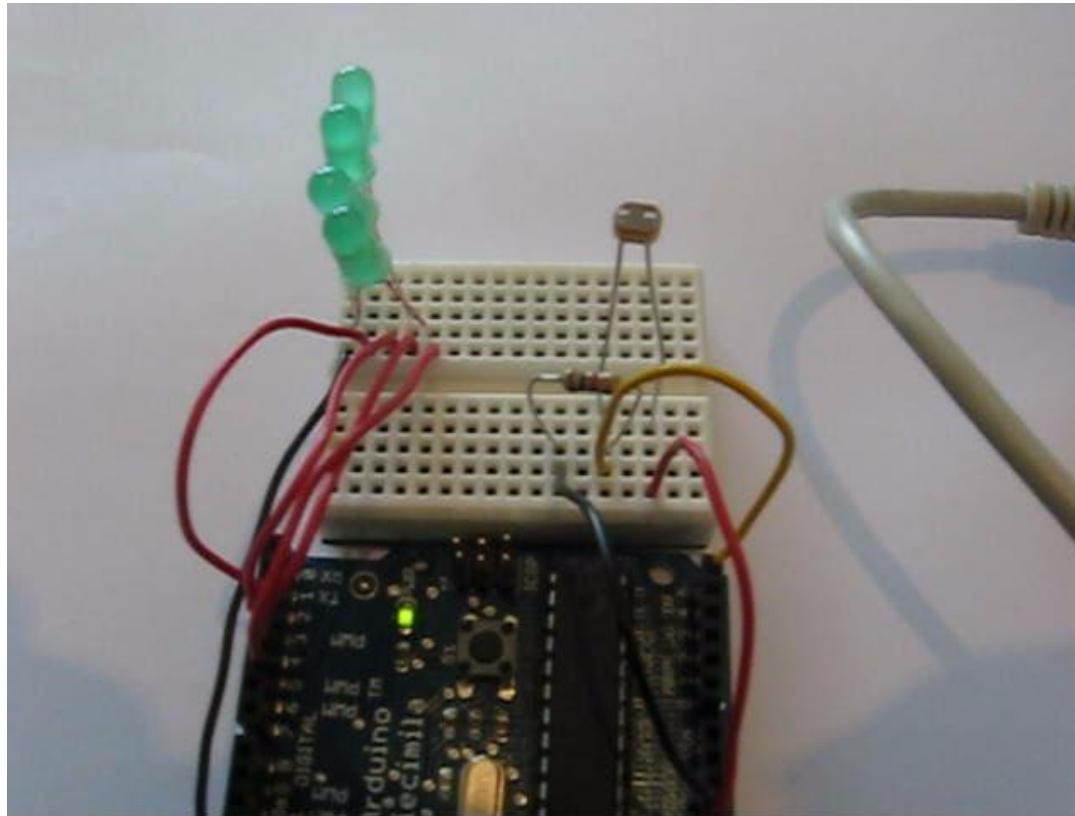
Arduino- Analog Input/Digital Output- System Calibration



Using Automatic Self-Calibration techniques you can detach the system from the computer and install it as an stand alone piece in the environment

Arduino- Analog Input/Digital Output- Using Multiple Actuators

We can control multiple Actuators based on sensed physical property from one sensor, defining multiple threshold beyond which different signals are sent to different sensors resulting in more than one conditional situation

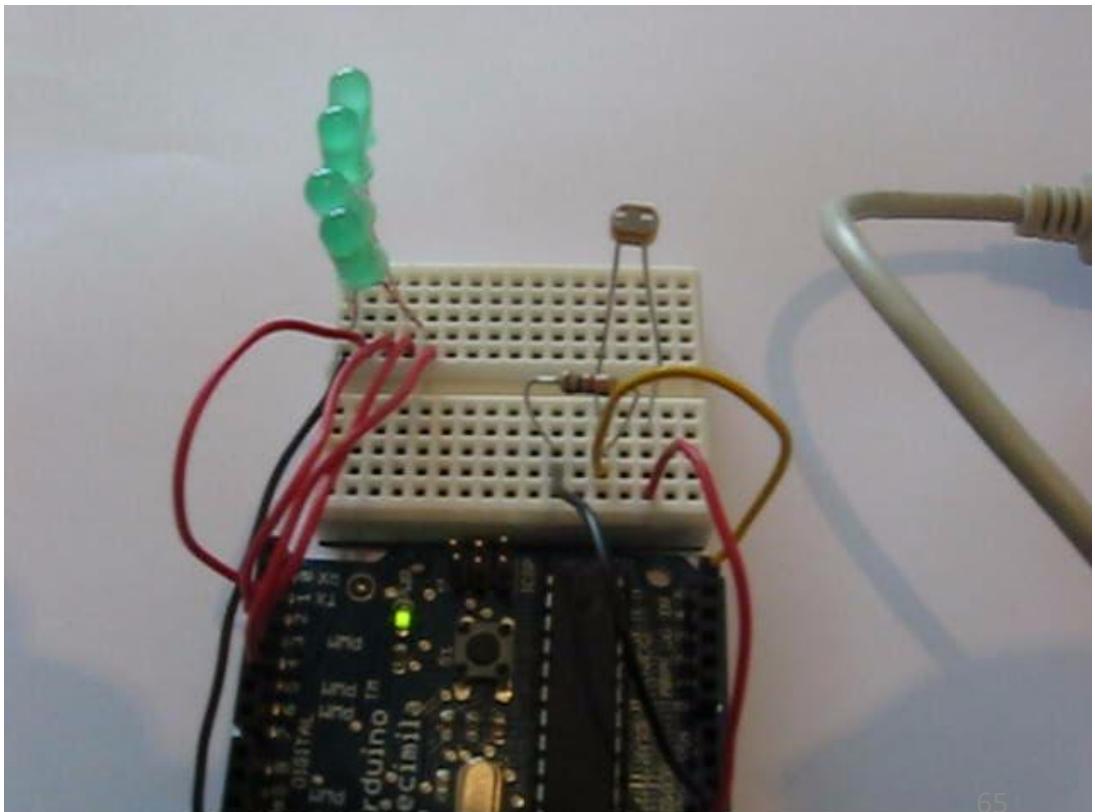


```

//Connect Photocell to Analog pin 0
//Connect LEDs to Digital Pin 2,3,4,5
int avrage=0;
void setup(){
Serial.begin(9600);//Beginning Serial Connection
pinMode(2,OUTPUT);
pinMode(3,OUTPUT);
pinMode(4,OUTPUT);
pinMode(5,OUTPUT);
for (int i=0; i<20; i++){
avrage=avrage+analogRead(0);
}
avrage=avrage/20;
Serial.println("System Ready");
Serial.println(avrage);
}
void loop(){
int in = analogRead(0);// Reading Sensed data from Arduino
//Serial.println(in);// Writing Sensed Data to Serial Port
if(in<avrage/5){
digitalWrite(2,HIGH);
digitalWrite(3,HIGH);
digitalWrite(4,HIGH);
digitalWrite(5,HIGH);
}
if((avrage/5<in)&&(in<avrage/4)){
digitalWrite(2,LOW);
digitalWrite(3,HIGH);
digitalWrite(4,HIGH);
digitalWrite(5,HIGH);
}
if((avrage/4<in)&&(in<avrage/3)){
digitalWrite(2,LOW);
digitalWrite(3,LOW);
digitalWrite(4,HIGH);
digitalWrite(5,HIGH);
}
if((avrage/3<in)&&(in<avrage/2)){
digitalWrite(2,LOW);
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,HIGH);
}
if((avrage/2<in)&&(in<avrage)){
digitalWrite(2,LOW);
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,LOW);
}
}

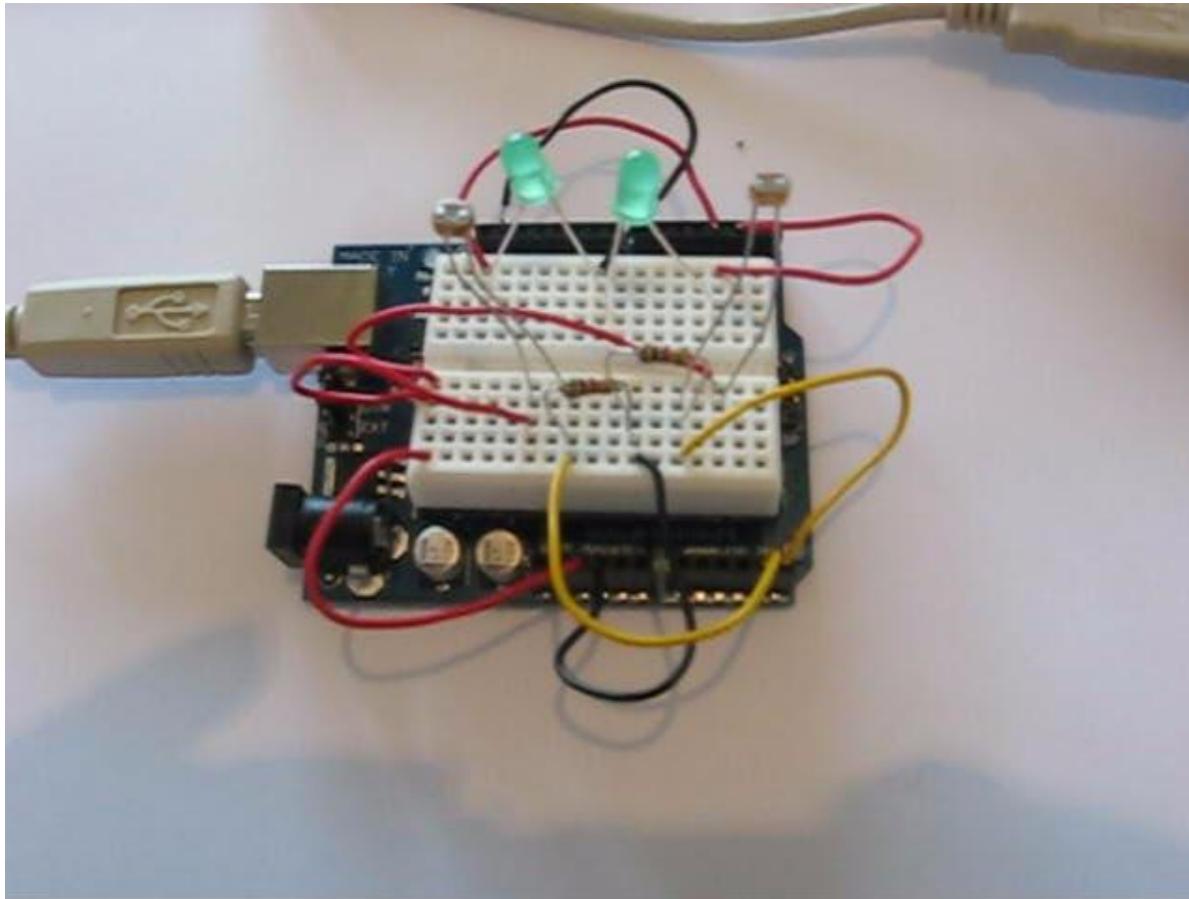
```

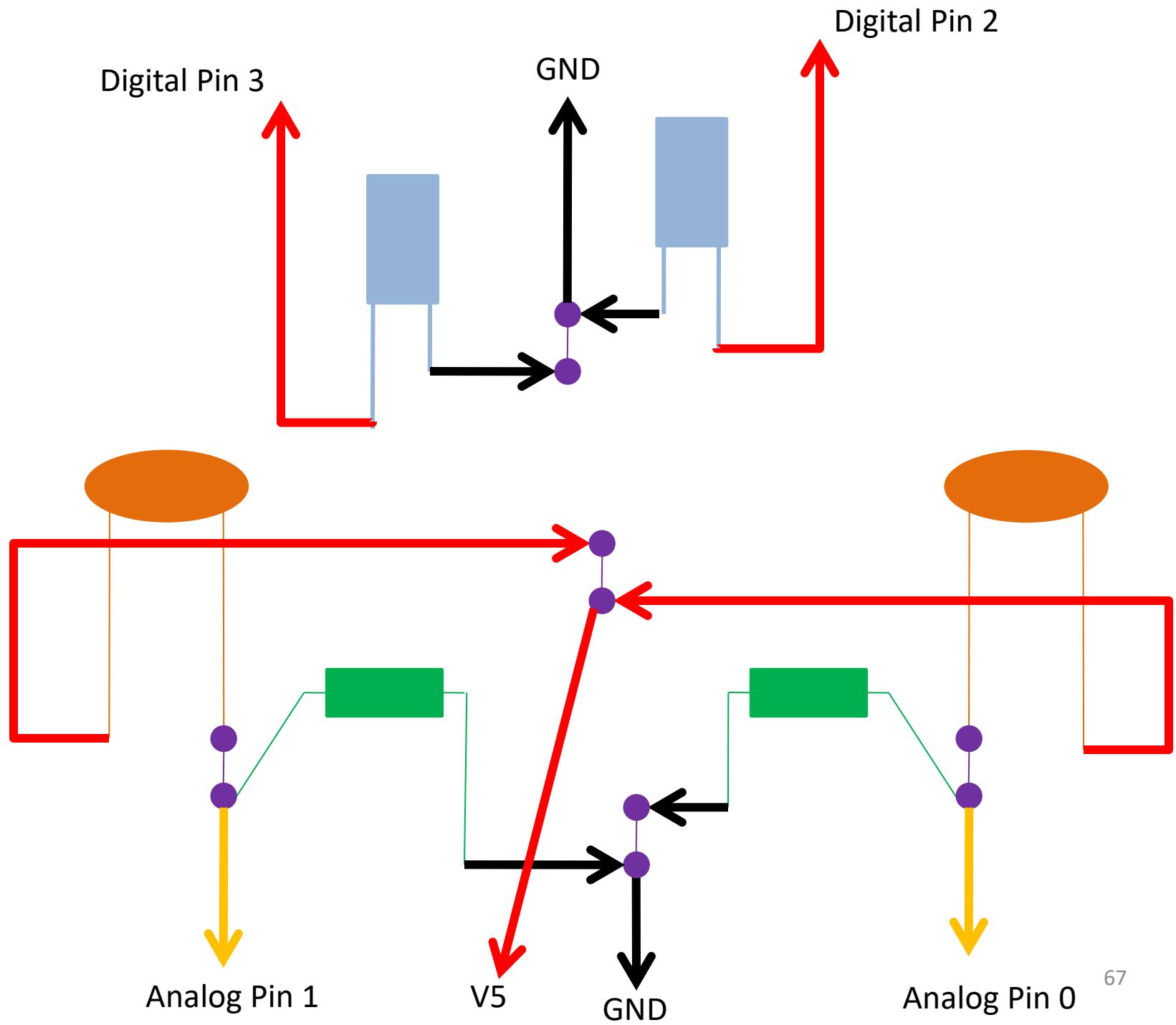
Arduino- Analog Input/Digital Output- Using Multiple Actuators



Arduino- Analog Input/Digital Output- Using Multiple Sensors & Multiple Actuators

We can have multiple sensors. In this Scenario we have two sensors and each sensor is controlling one LED



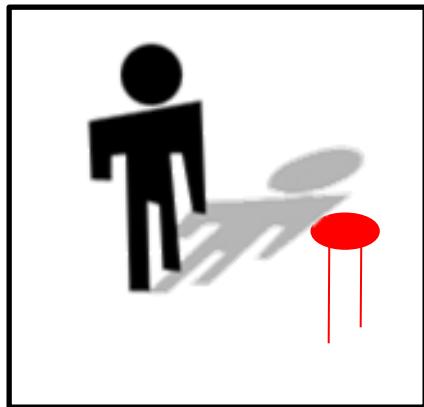


Arduino- Analog Input/Digital Output- Using Multiple Sensors & Multiple Actuators

```
//Sensor A is Connected to Analog pin 0 and is controlling the LED connected to Digital Pin 2
//Sensor B is Connected to Analog pin 1 and is controlling the LED connected to Digital Pin 3
int averageA;
int averageB;
void setup(){
    Serial.begin(9600);
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    for(int i=0;i<10;i++) //Calibrate the first sensor
        averageA+=analogRead(0); //Calibrate the first sensor
    averageA/=10; //Calibrate the first sensor
    for(int i=0;i<10;i++) //Calibrate the second sensor
        averageB+=analogRead(1); //Calibrate the second sensor
    averageB/=10; //Calibrate the second sensor
    Serial.println("System Ready"); //System let us know that the calibration is done
}
void loop(){
    int A = analogRead(0);
    int B = analogRead(1);
    if (A<averageA/2)
        digitalWrite(2,HIGH);
    else
        digitalWrite(2,LOW);
    if (B<averageB/2)
        digitalWrite(3,HIGH);
    else
        digitalWrite(3,LOW);
}
```

Detecting Presence of a subject using one photocell

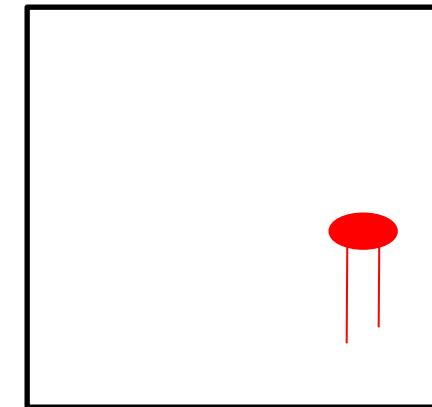
1. Using one Photo sensor you can detect if a subject is near or at a specific point in the space by detecting the casted shadow on a photo sensor



A shadow is casted on this spot



Somebody/Something is here



No shadow is casted on this spot

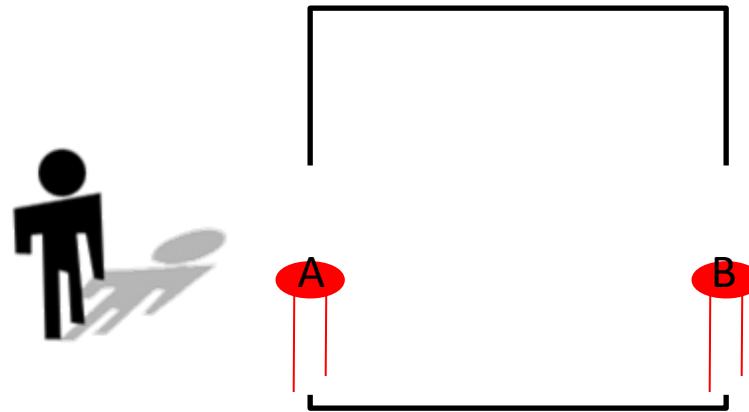


Nobody/Nothing is here

1. Read the amount of Light
2. If the amount of light is less than the threshold it means that a shadow is casted
3. If Something/Someone is casting a shadow at this area it means that He/She/It is there
4. React to this knowledge
5. Go to 1

Keeping track of entering and exiting of a subject to and from a space using two photo sensors

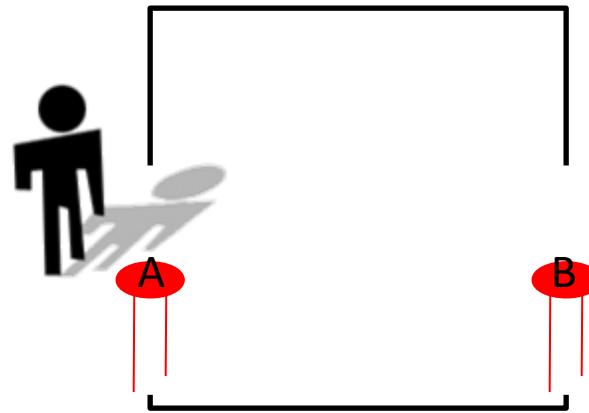
1. Using Two Photo sensors you can detect if a subject has entered a space and is inside or has left the space and is outside. You can also detect from which point the subject has entered or left



At first the User is outside and none of the sensors are detecting a shadow

Keeping track of entering and exiting of a subject to and from a space using two photo sensors

1. Using Two Photo sensor you can detect if a subject has intered a space and inside or has left the space and is outside. You can also detect from which point the subject has entered or left



At first the User is outside and none of the sensors are detecting a shadow



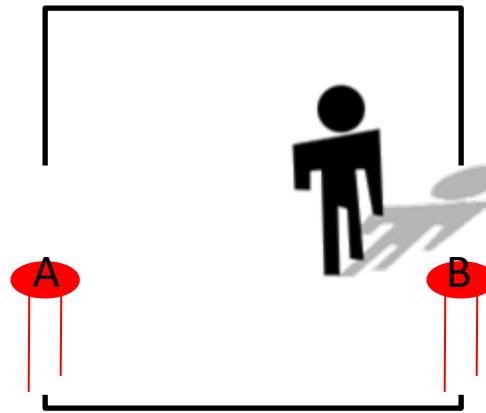
Shadow is casted at point A



The user was outside and point A is passed >>>> The user is entering the space from point A

Keeping track of entering and exiting of a subject to and from a space using two photo sensors

1. Using Two Photo sensor you can detect if a subject has intered a space and inside or has left the space and is outside. You can also detect from which point the subject has entered or left



At first the User is outside and none of the sensors are detecting a shadow



Shadow is casted at point A



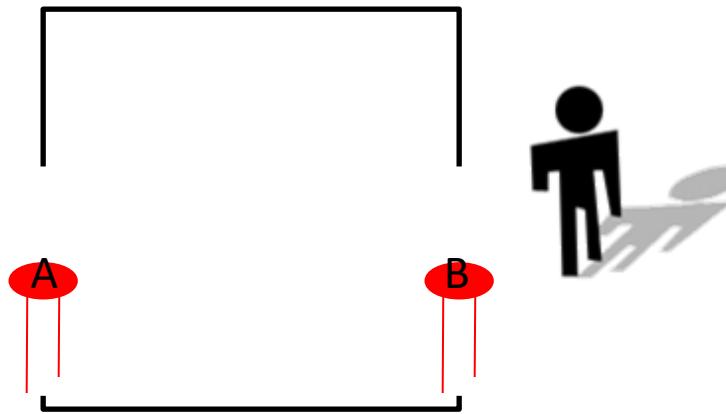
The user was outside and point A is passed >>>> The user is entering the space from point A



Shadow is casted at point B

Keeping track of entering and exiting of a subject to and from a space using two photo sensors

1. Using Two Photo sensor you can detect if a subject has intered a space and inside or has left the space and is outside. You can also detect from which point the subject has entered or left



At first the User is outside and none of the sensors are detecting a shadow



Shadow is casted at point A



The user was outside and point A is passed >>>> The user is entering the space from point A

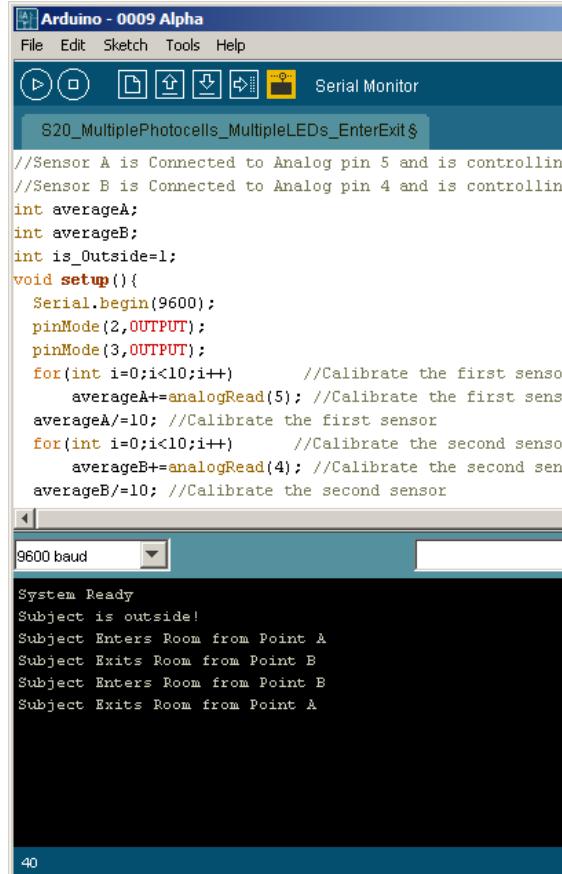


Shadow is casted at point B



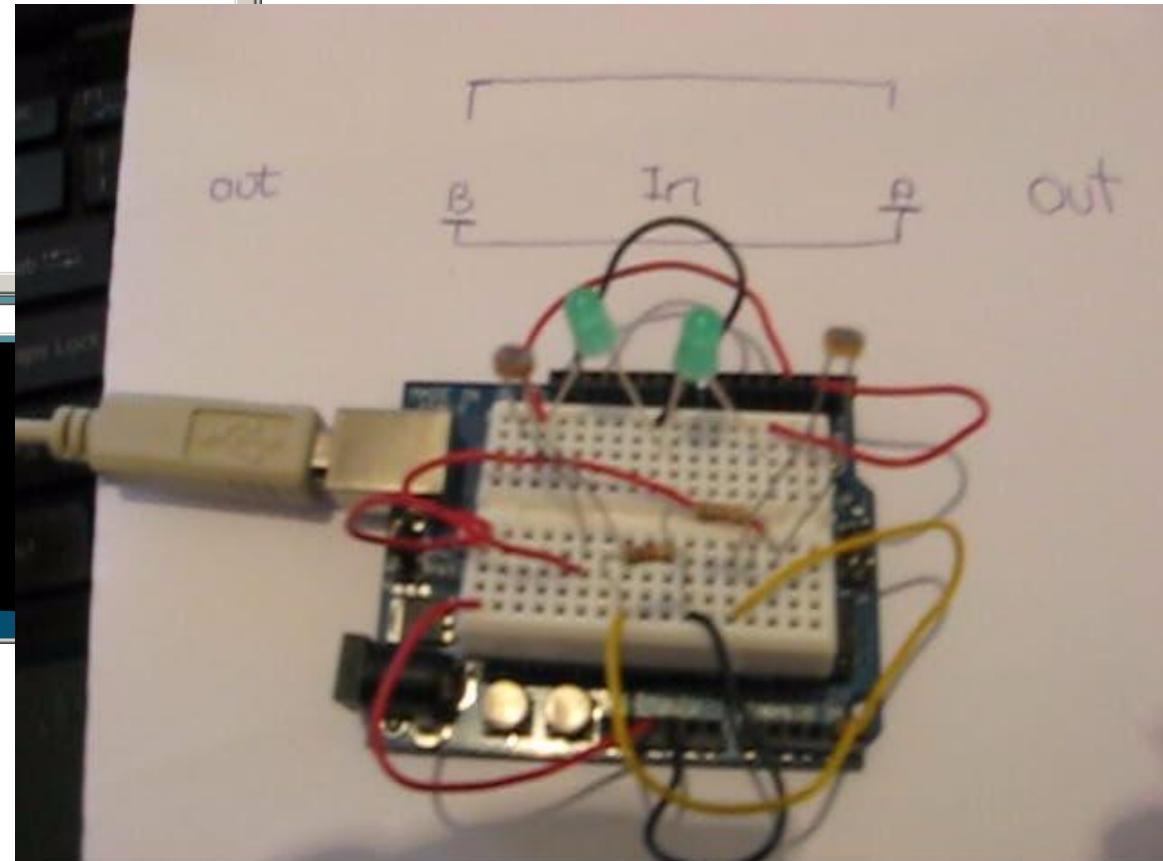
The user was inside and point B is passed>>>> The user is exiting the space from point B

Arduino-Detect Entrance/Exit with two photocells



The image shows the Arduino IDE interface. The top window is titled "Arduino - 0009 Alpha" and contains the code for the project. The bottom window is the "Serial Monitor" showing the program's output.

```
//Sensor A is Connected to Analog pin 5 and is controlling the LED connected to Digit:  
//Sensor B is Connected to Analog pin 4 and is controlling the LED connected to Digit:  
  
int averageA;  
int averageB;  
int is_Outside=1;  
  
void setup(){  
  Serial.begin(9600);  
  pinMode(2,OUTPUT);  
  pinMode(3,OUTPUT);  
  for(int i=0;i<10;i++)      //Calibrate the first sensor  
    averageA+=analogRead(5); //Calibrate the first sensor  
  averageA/=10; //Calibrate the second sensor  
  for(int i=0;i<10;i++)      //Calibrate the second sensor  
    averageB+=analogRead(4); //Calibrate the second sensor  
  averageB/=10; //Calibrate the second sensor  
}  
  
9600 baud  
  
System Ready  
Subject is outside!  
Subject Enters Room from Point A  
Subject Exits Room from Point B  
Subject Enters Room from Point B  
Subject Exits Room from Point A
```



Arduino- Detect Entrance/Exit with two photocells

```
//Sensor A is Connected to Analog pin 0 and is controlling the LED connected to Digital Pin 2
//Sensor B is Connected to Analog pin 1 and is controlling the LED connected to Digital Pin 3
int averageA;
int averageB;
int is_Outside=1;
void setup(){
    Serial.begin(9600);
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    for(int i=0;i<10;i++) //Calibrate the first sensor
        averageA+=analogRead(0); //Calibrate the first sensor
    averageA/=10; //Calibrate the first sensor
    for(int i=0;i<10;i++) //Calibrate the second sensor
        averageB+=analogRead(1); //Calibrate the second sensor
    averageB/=10; //Calibrate the second sensor
    Serial.println("System Ready"); //System let us know that the calibration is done
    Serial.println(averageA);
    Serial.println(averageB);
}
void loop(){
    int A = analogRead(0);
    int B = analogRead(1);
    if (A<averageA/2){//Point A is passed
        digitalWrite(2,HIGH);
        if(is_Outside==1)Serial.println("Subject Enters Room from Point A");
        if(is_Outside==-1)Serial.println("Subject Exits Room from Point A");
        is_Outside=-is_Outside;
        delay(1000);// With delay system avoids multiple reads from one point pass
    }else
        digitalWrite(2,LOW);
    if (B<averageB/2){///Point B is passed
        digitalWrite(3,HIGH);
        if(is_Outside==1)Serial.println("Subject Enters Room from Point B");
        if(is_Outside==-1)Serial.println("Subject Exits Room from Point B");
        is_Outside=-is_Outside;
        delay(1000);//With delay system avoids multiple reads from one point pass
    }else
        digitalWrite(3,LOW);
}
```

Arduino-Detecting Movement Direction

Arduino - 0009 Alpha

File Edit Sketch Tools Help

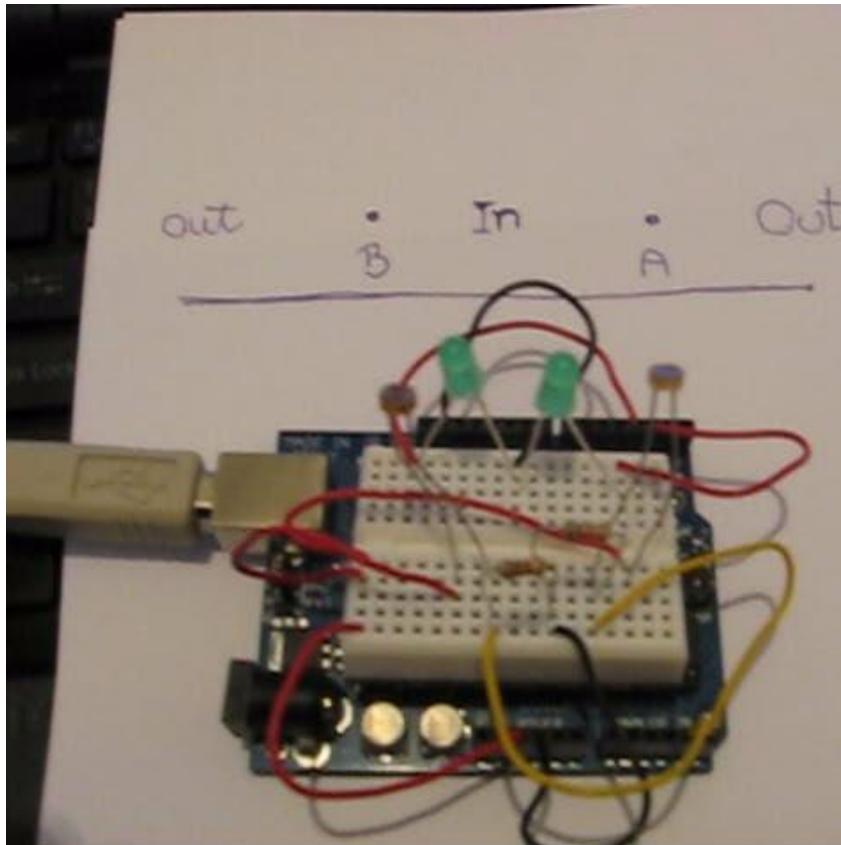
S21_MultiplePhotocells_MultipleLEDs_Direction

```
if (A<averageA/2){//Point A is passed
  digitalWrite(2,HIGH);
  if(is_Outside==1)Serial.println("Subject is moving from Right to Left");
  if(is_Outside== -1)Serial.println("Subject is moving from Left to Right");
  is_Outside=-is_Outside;
  delay(1000); //With delay system avoids multiple reads from one point pass
}else
  digitalWrite(2,LOW);
if (B<averageB/2){///Point B is passed
  digitalWrite(3,HIGH);
  if(is_Outside==1)Serial.println("Subject is moving from Left to Right");
  if(is_Outside== -1)Serial.println("Subject is moving from Right to Left");
  is_Outside=-is_Outside;
  delay(1000); //With delay system avoids multiple reads from one point pass
}else
  digitalWrite(3,LOW);
}

9600 baud
```

Send

System Ready
Subject is outside!
Subject is moving from Right to Left
Subject is moving from Right to Left
Subject is moving from Left to Right
Subject is moving from Left to Right



Arduino-

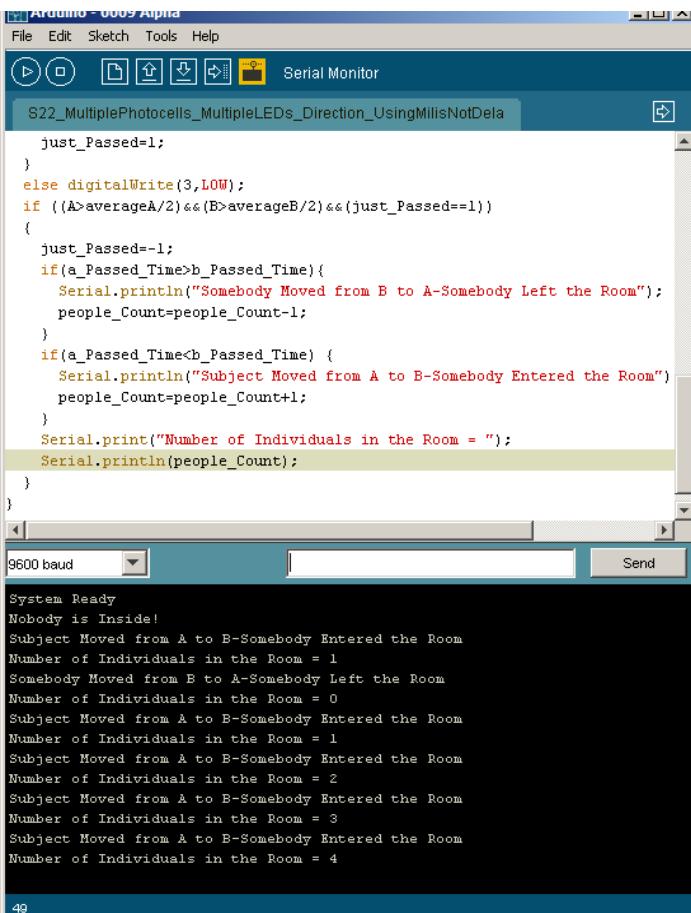
Detecting Movement Direction

```
//Sensor A is Connected to Analog pin 0 and is controlling the LED connected to Digital Pin 2
//Sensor B is Connected to Analog pin 41and is controlling the LED connected to Digital Pin 3
int averageA;
int averageB;
int is_Outside=1;
void setup(){
    Serial.begin(9600);
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    for(int i=0;i<10;i++) //Calibrate the first sensor
        averageA+=analogRead(0); //Calibrate the first sensor
    averageA/=10; //Calibrate the first sensor
    for(int i=0;i<10;i++) //Calibrate the second sensor
        averageB+=analogRead(1); //Calibrate the second sensor
    averageB/=10; //Calibrate the second sensor
    Serial.println("System Ready"); //System let us know that the calibration is done
    Serial.println("Subject is outside!");
}
void loop(){
    int A = analogRead(0);
    int B = analogRead(1);
    if (A<averageA/2){//Point A is passed
        digitalWrite(2,HIGH);
        if(is_Outside==1)Serial.println("Subject is moving from Right to Left");
        if(is_Outside==-1)Serial.println("Subject is moving from Left to Right");
        is_Outside=-is_Outside;
        delay(1000);// With delay system avoids multiple reads from one point pass
    }else
        digitalWrite(2,LOW);
    if (B<averageB/2){///Point B is passed
        digitalWrite(3,HIGH);
        if(is_Outside==1)Serial.println("Subject is moving from Left to Right");
        if(is_Outside==-1)Serial.println("Subject is moving from Right to Left");
        is_Outside=-is_Outside;
        delay(1000);//With delay system avoids multiple reads from one point pass
    }else
        digitalWrite(3,LOW);
}
```

Same Physical Setting, Same Code Different Interpretation of Sensed Data

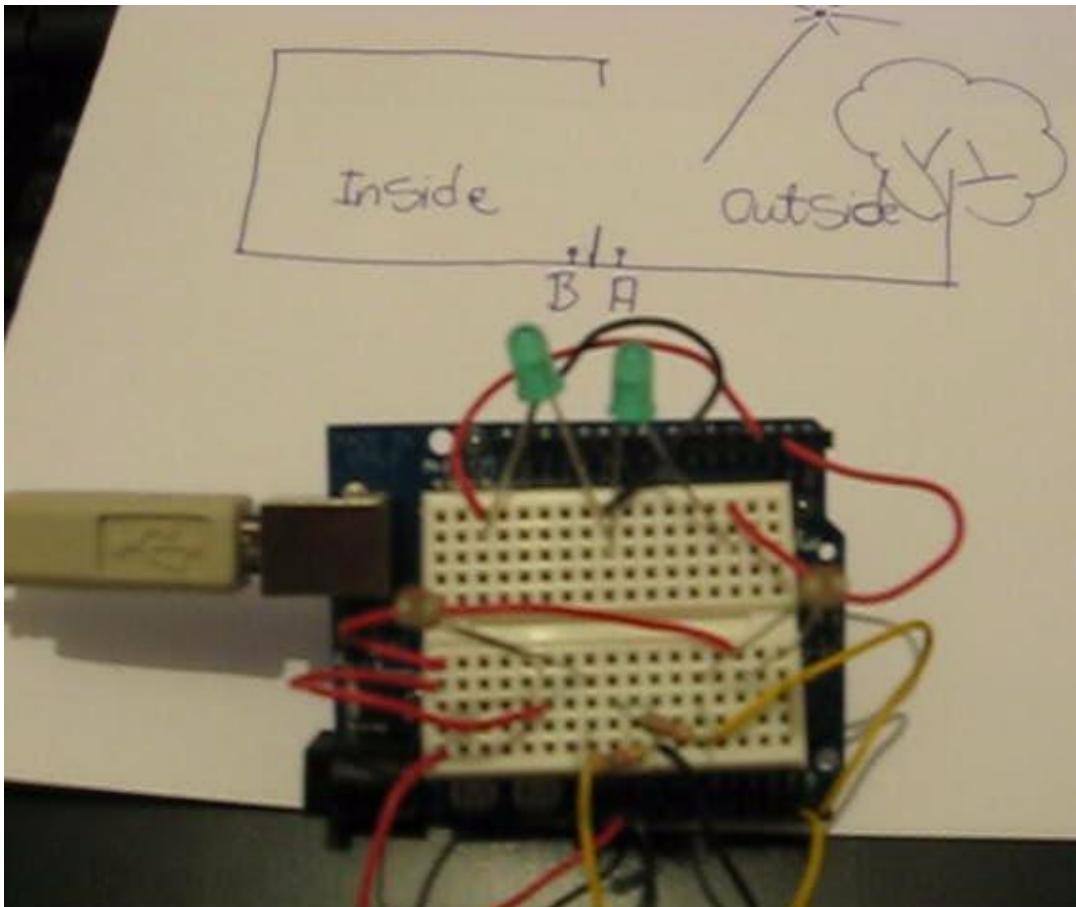
In the last two exercises we had the same physical setting, same arrangement of sensors and circuits and we were sensing the same physical properties, the only difference was how we were interpreting the data.

Detecting Entrance and Movement Direction from Single Node



The image shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, Help, and a Serial Monitor icon. The sketch window contains C++ code for an Arduino project named "S22_MultiplePhotocells_MultipleLEDs_Direction_UsingMillisNotDelay". The code uses two photocells (A and B) connected to digital pins 3 and 2 respectively. It tracks the number of individuals in the room and prints messages to the Serial Monitor indicating movement between areas labeled "Inside" and "Outside". The bottom window shows the Serial Monitor output at 9600 baud, displaying the printed messages.

```
File Edit Sketch Tools Help
Serial Monitor
S22_MultiplePhotocells_MultipleLEDs_Direction_UsingMillisNotDelay
just_Passed=1;
}
else digitalWrite(3,LOW);
if ((A>averageA/2)&&(B>averageB/2)&&(just_Passed==1))
{
just_Passed=-1;
if(a_Passed_Time>b_Passed_Time){
  Serial.println("Somebody Moved from B to A-Somebody Left the Room");
  people_Count=people_Count-1;
}
if(a_Passed_Time<b_Passed_Time) {
  Serial.println("Subject Moved from A to B-Somebody Entered the Room");
  people_Count=people_Count+1;
}
Serial.print("Number of Individuals in the Room = ");
Serial.println(people_Count);
}
}
System Ready
Nobody is Inside!
Subject Moved from A to B-Somebody Entered the Room
Number of Individuals in the Room = 1
Somebody Moved from B to A-Somebody Left the Room
Number of Individuals in the Room = 0
Subject Moved from A to B-Somebody Entered the Room
Number of Individuals in the Room = 1
Somebody Moved from A to B-Somebody Entered the Room
Number of Individuals in the Room = 2
Subject Moved from A to B-Somebody Entered the Room
Number of Individuals in the Room = 3
Subject Moved from A to B-Somebody Entered the Room
Number of Individuals in the Room = 4
```



Detecting Entrance and Movement Direction from Single Node

```
//Sensor A is Connected to Analog pin 0 and is controlling the LED connected to Digital Pin 2
//Sensor B is Connected to Analog pin 1 and is controlling the LED connected to Digital Pin 3
int averageA;
int averageB;
long a_Passed_Time;
long b_Passed_Time;
int just_Passed=-1;
int people_Count=0;
void setup(){
    Serial.begin(9600);
    pinMode(2,OUTPUT);
    pinMode(3,OUTPUT);
    for(int i=0;i<10;i++) //Calibrate the first sensor
        averageA+=analogRead(0); //Calibrate the first sensor
    averageA/=10; //Calibrate the first sensor
    for(int i=0;i<10;i++) //Calibrate the second sensor
        averageB+=analogRead(1); //Calibrate the second sensor
    averageB/=10; //Calibrate the second sensor
    Serial.println("System Ready"); //System let us know that the calibration is done
    Serial.println("Nobody is Inside!");
}
void loop(){
    int A = analogRead(0);
    int B = analogRead(1);
    if (A<averageA/2){//Point A is passed
        digitalWrite(2,HIGH);
        a_Passed_Time=millis();
        just_Passed=1;
    }
    else digitalWrite(2,LOW);
    if (B<averageB/2){///Point B is passed
        digitalWrite(3,HIGH);
        b_Passed_Time=millis();
        just_Passed=1;
    }
    else digitalWrite(3,LOW);
    if ((A>averageA/2)&&(B>averageB/2)&&(just_Passed==1))
    {
        just_Passed=-1;
        if(a_Passed_Time>b_Passed_Time){
            Serial.println("Somebody Moved from B to A-Somebody Left the Room");
            people_Count=people_Count-1;
        }
        if(a_Passed_Time<b_Passed_Time) {
            Serial.println("Subject Moved from A to B-Somebody Entered the Room");
            people_Count=people_Count+1;
        }
        Serial.print("Number of Individuals in the Room = ");
        Serial.println(people_Count);
    }
}
```

Detecting

Entrance and

Movement Direction

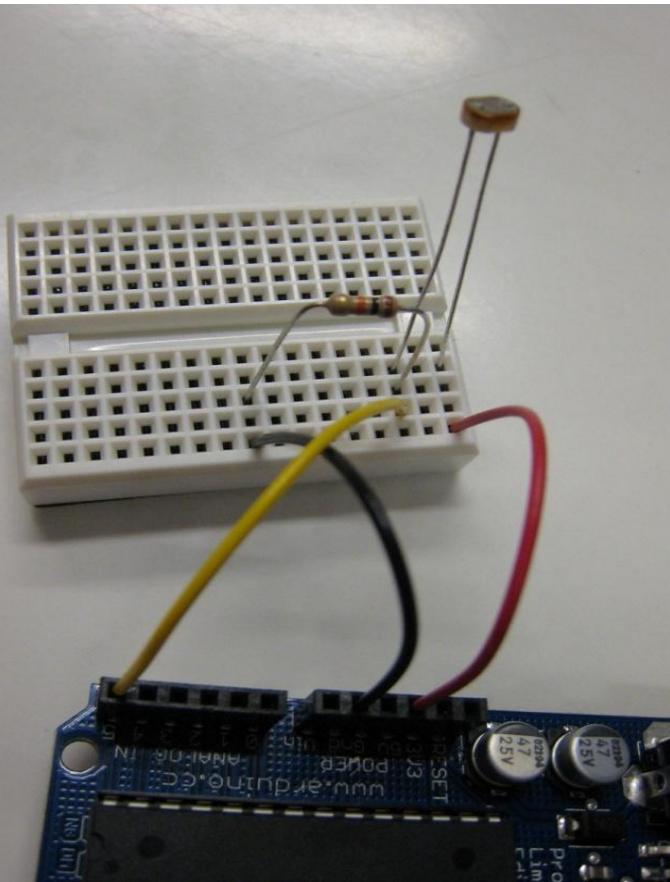
from Single Node

In this exercise every time that a photocell is covered the system also detects the time of the incident using millis() function. This function shows how much time has passed since the program is running. Once both photocells are covered and then are revealed, the system checks which photocell has been passed first. if A is passed first then it means that the motion is from A to B meaning somebody has entered the room, and if B has been passed first it means that the motion is from B to A meaning that somebody has entered the room. The system also Adjusts the number of individuals in the room accordingly each time that it detects an entrance or exit.

```
//Sensor A is Connected to Analog pin 0 and is controlling the LED connected to Digital Pin 2  
//Sensor B is Connected to Analog pin 1 and is controlling the LED connected to Digital Pin 3  
int averageA;  
int averageB;  
long a_Passed_Time;  
long b_Passed_Time;  
int just_Passed=-1;  
int people_Count=0;  
void setup()  
{  
    Serial.begin(9600);  
    pinMode(2,OUTPUT);  
    pinMode(3,OUTPUT);  
    for(int i=0;i<10;i++) //Calibrate the first sensor  
        averageA+=analogRead(0); //Calibrate the first sensor  
    averageA/=10; //Calibrate the first sensor  
    for(int i=0;i<10;i++) //Calibrate the second sensor  
        averageB+=analogRead(1); //Calibrate the second sensor  
    averageB/=10; //Calibrate the second sensor  
    Serial.println("System Ready"); //System let us know that the calibration is done  
    Serial.println("Nobody is Inside!");  
}  
void loop()  
{  
    int A = analogRead(0);  
    int B = analogRead(1);  
    if (A<averageA/2){//Point A is passed  
        digitalWrite(2,HIGH);  
        a_Passed_Time=millis();  
        just_Passed=1;  
    }  
    else digitalWrite(2,LOW);  
    if (B<averageB/2){//Point B is passed  
        digitalWrite(3,HIGH);  
        b_Passed_Time=millis();  
        just_Passed=1;  
    }  
    else digitalWrite(3,LOW);  
    if ((A>averageA/2)&&(B>averageB/2)&&(just_Passed==1))  
    {  
        just_Passed=-1;  
        if(a_Passed_Time>b_Passed_Time){  
            Serial.println("Somebody Moved from B to A-Somebody Left the Room");  
            people_Count=people_Count-1;  
        }  
        if(a_Passed_Time<b_Passed_Time){  
            Serial.println("Subject Moved from A to B-Somebody Entered the Room");  
            people_Count=people_Count+1;  
        }  
        Serial.print("Number of Individuals in the Room = ");  
        Serial.println(people_Count);  
    }  
}
```

Changing Range of Sensed Data using different voltage pin or resistors

$$V=IR$$



With the Same Photo-Cell, using different resistors or connecting it to different power outlets V3 vs. V5 you can have different ranges for your sensing device.

The larger the range gets, the more variations on the sensed data you have and as a result your circuit is more sensitive to the variations in the sensed data.

```
void setup(){  
Serial.begin(9600);  
}  
void loop(){  
int in = analogRead(5);  
Serial.println(in);  
}
```

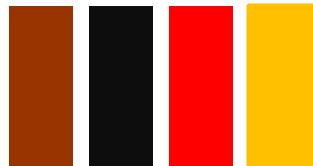
Resistor Color Coding

To identify the resistance we use a code system: we position them so that the gold strip (or band) is on the right side and then we measure:

$$10 \cdot 100$$

$$= 1000$$

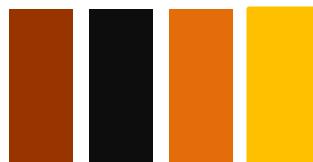
$$= 1k$$



$$10 \cdot 1000$$

$$= 10,000$$

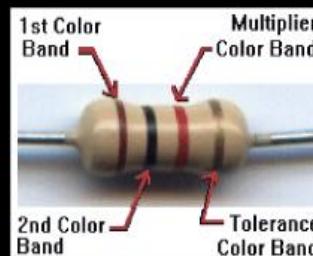
$$= 10k$$



$$10 \cdot 100,000$$

$$= 1000000$$

$$= 1 \text{ Mega}$$



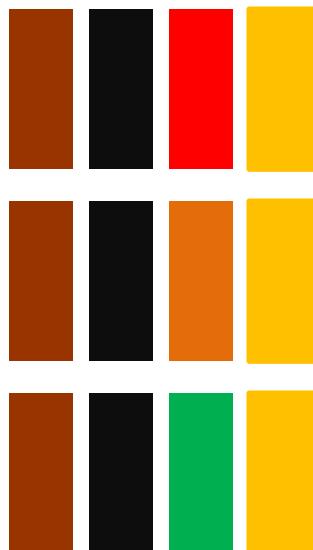
Color	First Strip	Second Strip	Third Strip
Black	0	0	x1
Brown	1	1	x10
Red	2	2	x100
Orange	3	3	x1,000
Yellow	4	4	x10,000
Green	5	5	x100,000
Blue	6	6	x1,000,000
Purple	7	7	
Gray	8	8	
White	9	9	

Higher Voltage and **Higher Resistance** gives you **Better Light Detection** Range

10^*100
 $=1000$
 $=1k$

10^*1000
 $=10,000$
 $=10k$

$10^*100,000$
 $=1000000$
 $=1 \text{ Mega}$



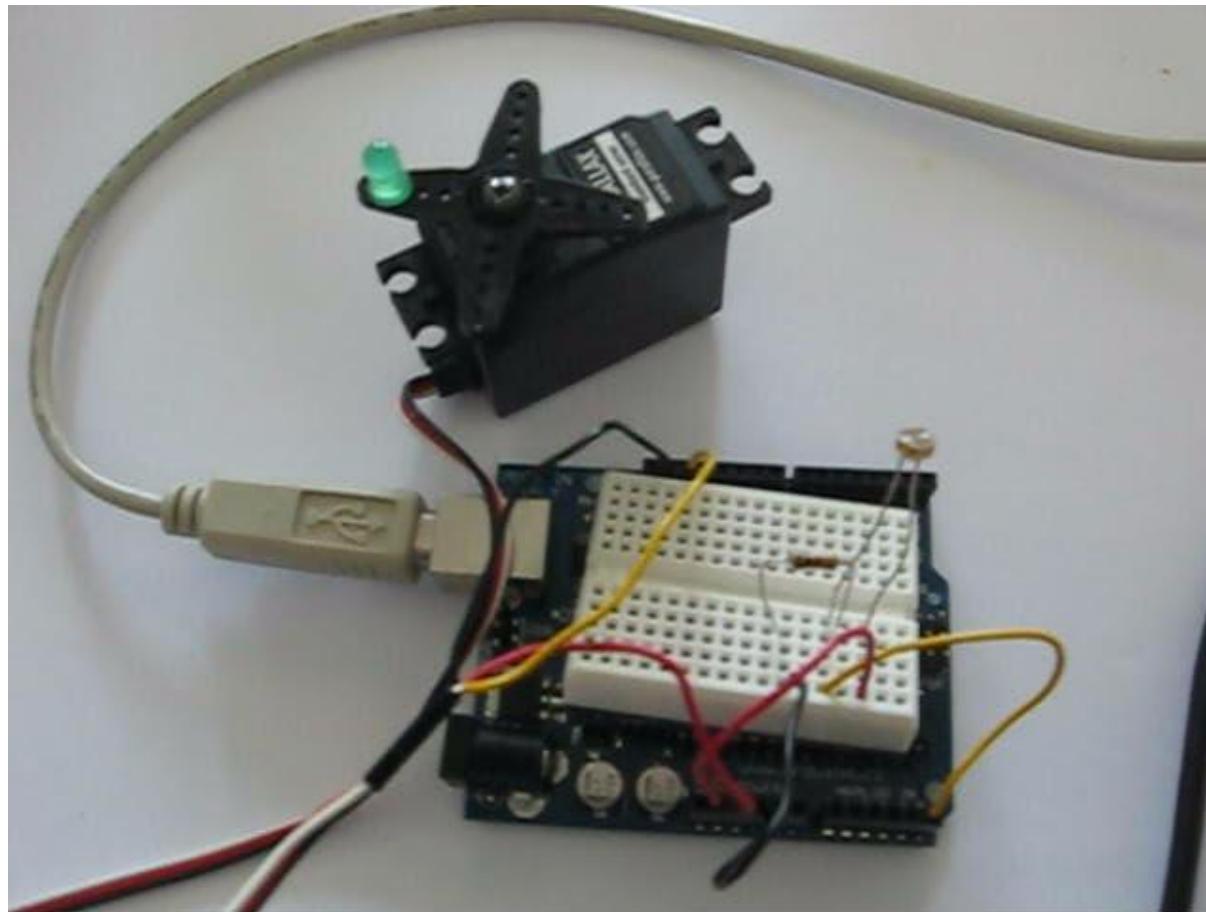
Resistor	Voltage	3 volt	5 volt
	1K	0-625 Dark-Light	0-950 Dark-Light
10 K	0-650 Dark-Light	0-1020 Dark-Light	
1 Meg	200-674	990-1023	

Responsive System

Input Photocell-Output Standard Servo

Photocell Covered-Go to 0 Degrees

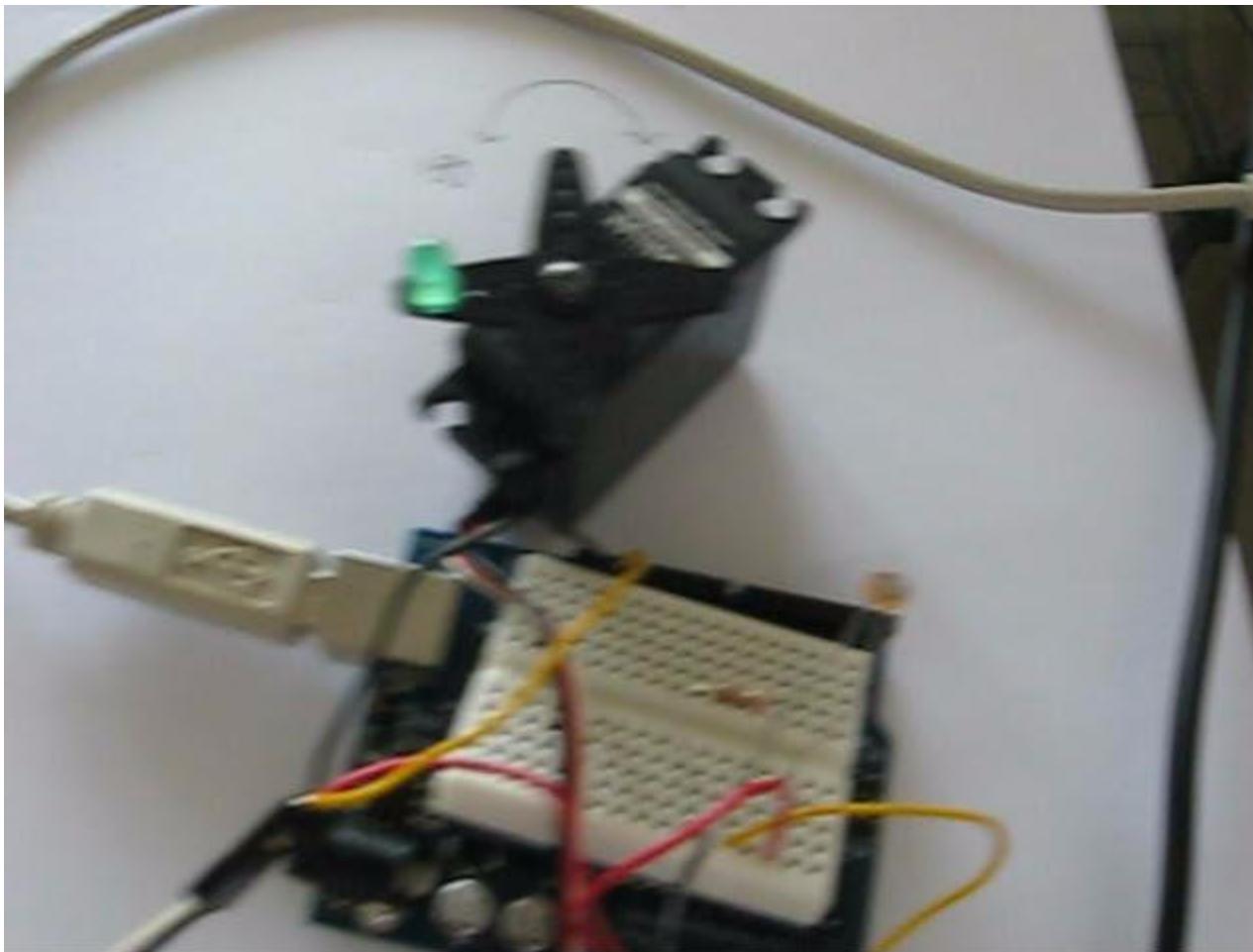
Photocell Not Covered-Go to 180 Degrees



Responsive System

Input Photocell-Output Standard Servo

Rotation Relative To Light Intensity



Responsive System

Input Photocell-Output Standard Servo

Photocell Covered-Go to 0 Degrees

Photocell Not Covered-Go to 180 Degrees

```
#include <Servo.h>

int avrage;
int val;//Variable determining the servo movement
Servo myServo;
void setup(){
Serial.begin(9600);//Begining Serial Connection
pinMode(13,OUTPUT);//Servo White wire connection
for (int i=0; i<20; i++){
avrage=avrage+analogRead(0);
myServo.attach(13);
}
avrage=avrage/20;
Serial.println("System Ready");
Serial.println(avrage);
}
void loop(){
int in = analogRead(0);// Reading Sensed data from Arduino
if (in>avrage/2) val=180;
if (in<avrage/2) val=0;
Serial.println(in);
myServo.write(val);
}
```

Responsive System

Input Photocell-Output Standard Servo

Rotation Relative To Light Intensity

```
#include <Servo.h>
Servo myServo;

int avrage;
int light_min=80;
int light_max=600;
float lighttoAngle;
void setup(){
Serial.begin(9600); //Begining Serial Connection
pinMode(13,OUTPUT); //Servo White wire connection
myServo.attach(13);
for (int i=0; i<20; i++){
avrage=avrage+analogRead(0);
}
avrage=avrage/20;
Serial.println("System Ready");
Serial.println(avrage);
}
void loop(){
int in = analogRead(0); // Reading Sensed data from Arduino
lighttoAngle=map(in,light_min, light_max,0,180);

Serial.print("Light=");
Serial.println(in);
Serial.print("Angel=");
Serial.println(int(lighttoAngle));
myServo.write(lighttoAngle);
delay(100);
}
```

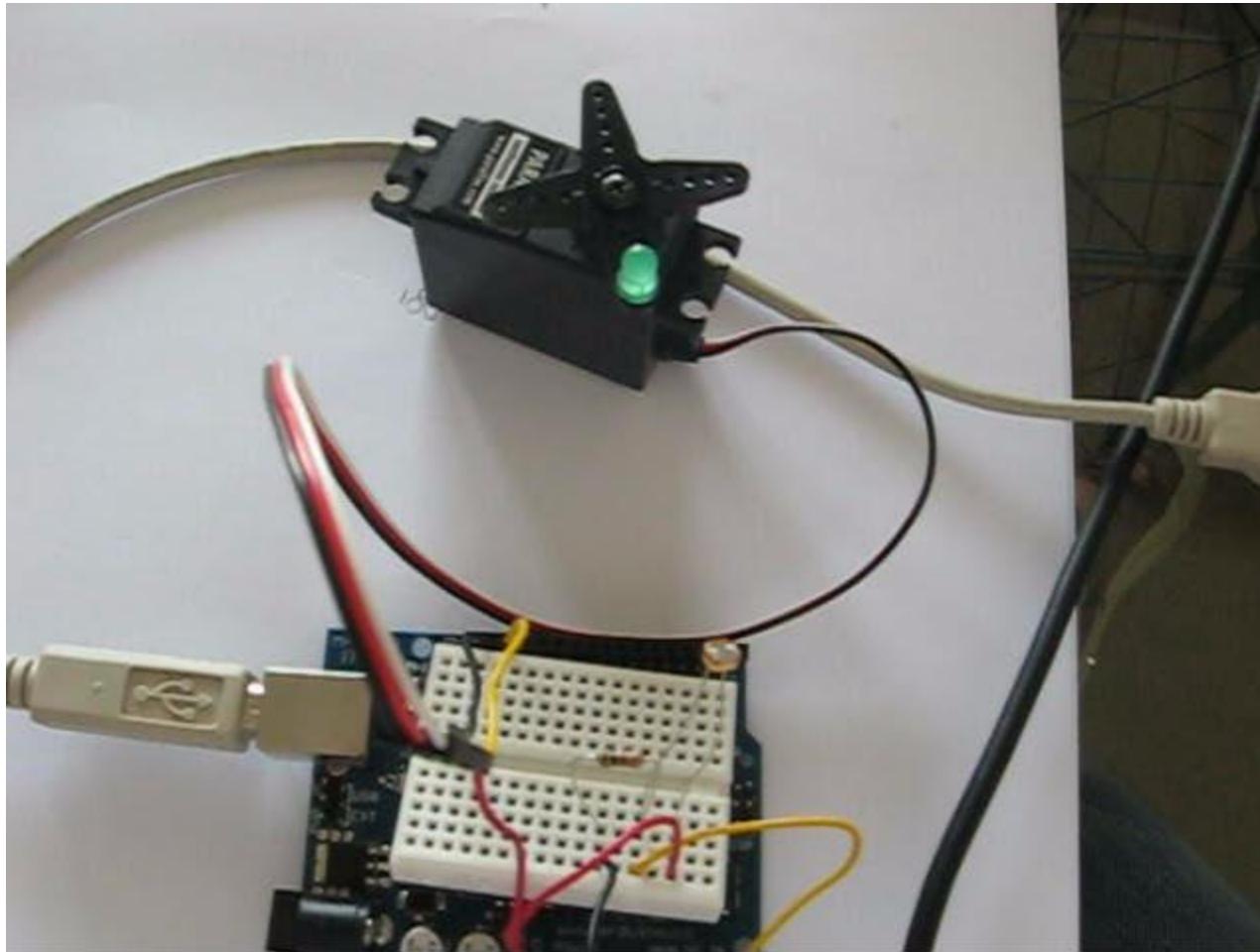
Arduino-Light to Angle Conversion



$$\frac{x}{\text{Max}(x)-\text{Min}(x)} = \frac{f(x)}{\text{Max } f(x)-\text{Min } f(x)}$$

Responsive System

Input Photocell-Output Continuous Servo Direction Change



Responsive System

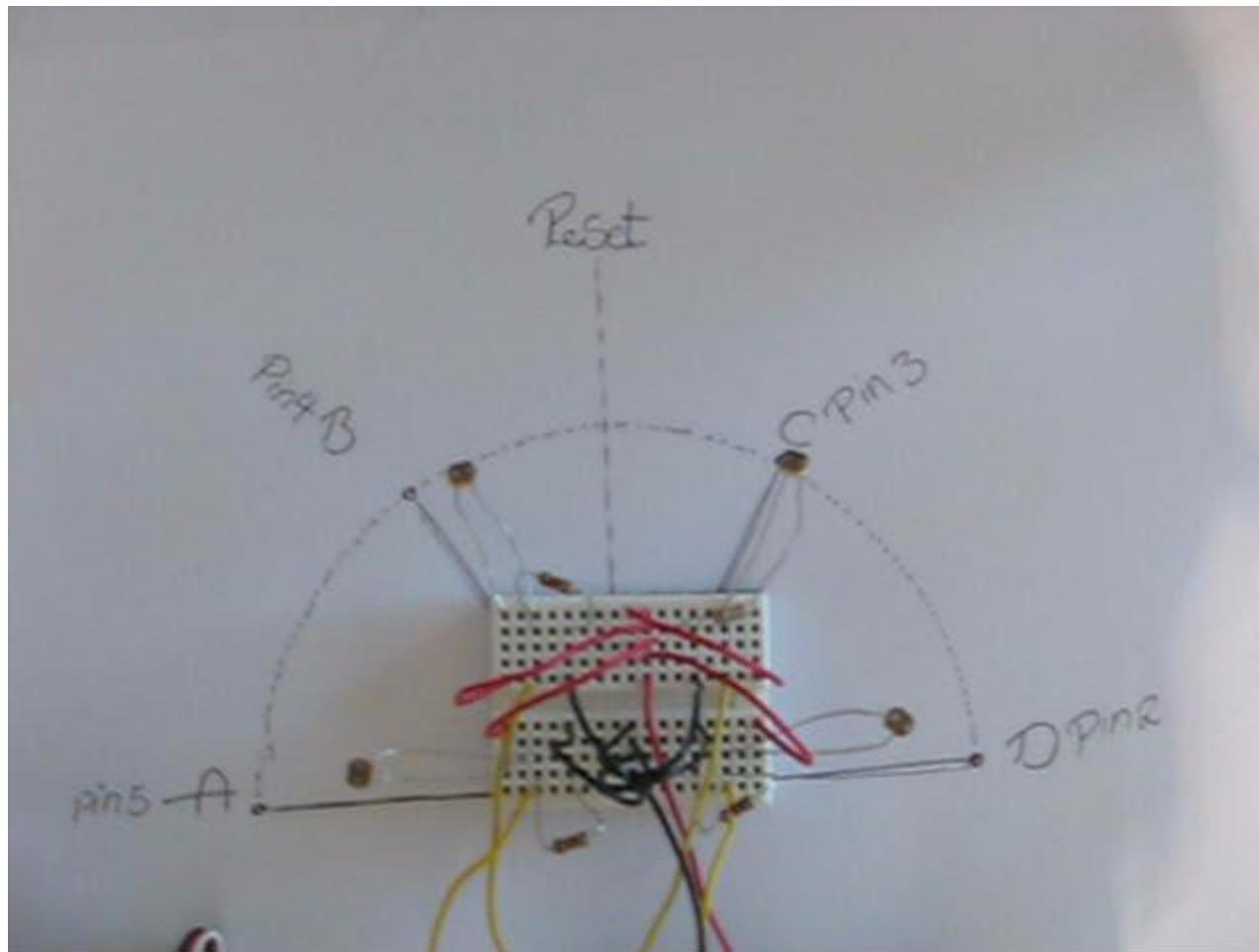
Input Photocell-Output Continuous Servo

Direction Change

```
int avrage;
int val;
void setup(){
Serial.begin(9600); //Beginning Serial Connection
pinMode(13,OUTPUT); //Servo White wire connection
for (int i=0; i<20; i++){
avrage=avrage+analogRead(0);
}
avrage=avrage/20;
Serial.println("System Ready");
Serial.println(avrage);
}
void loop(){
int in = analogRead(0); // Reading Sensed data from Arduino
Serial.println(in);
if(in<avrage-20) val=1200;
if((in>avrage-20)&&(in<avrage+20)) val=1500;
if(in>avrage+20) val=1800;
digitalWrite(13,HIGH);
delayMicroseconds(val); // 1.5ms This is the frequency at which the servo motor should
be static
digitalWrite(13,LOW);
delay(20); // 20ms
}
```

Responsive System

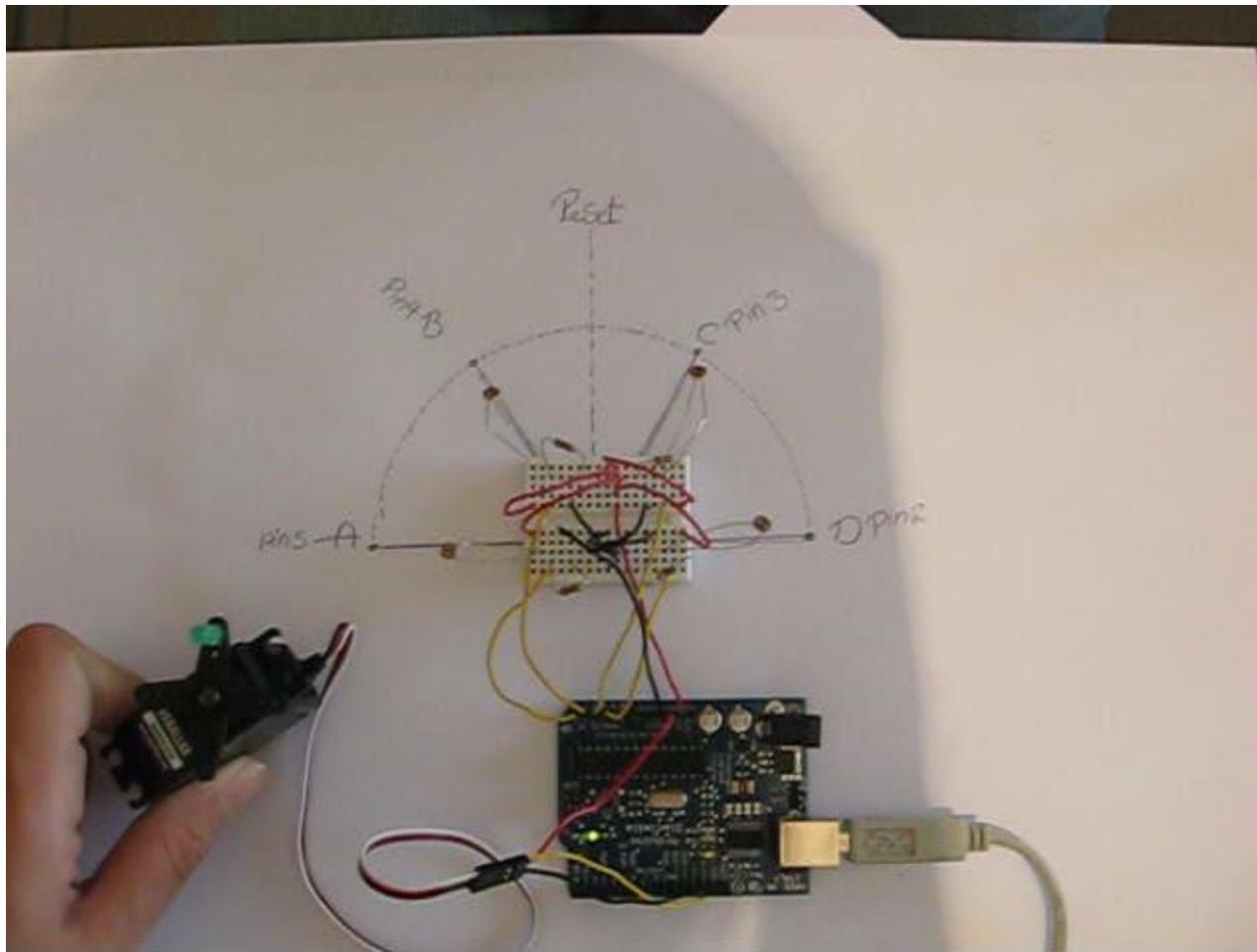
Input Photocell-Output Standard Servo Facing a Definitive Point



Responsive System

Input Photocell-Output Standard Servo

Facing a Definitive Point



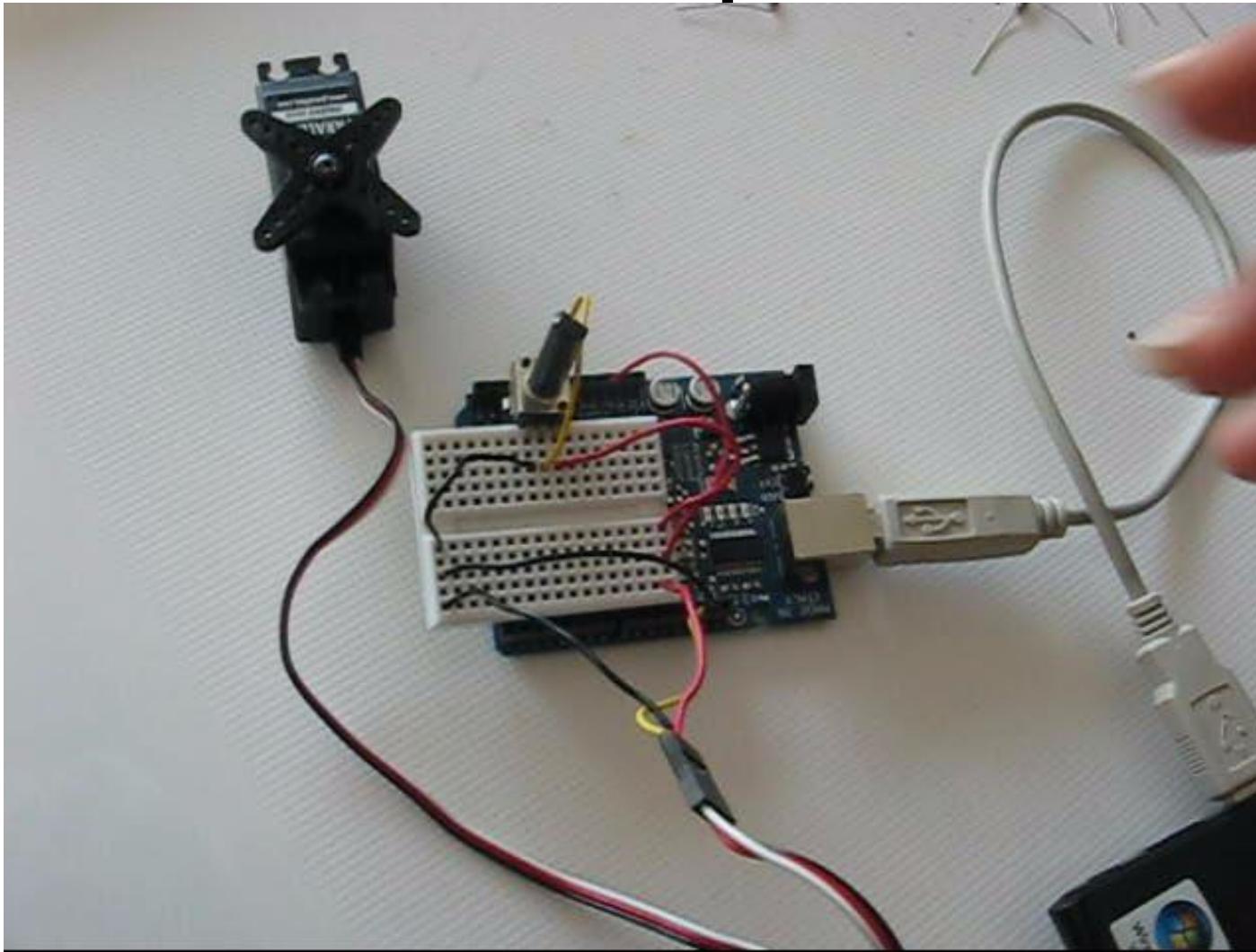
Responsive System Input Photocell- Output Standard Servo- Facing a Definitive Point

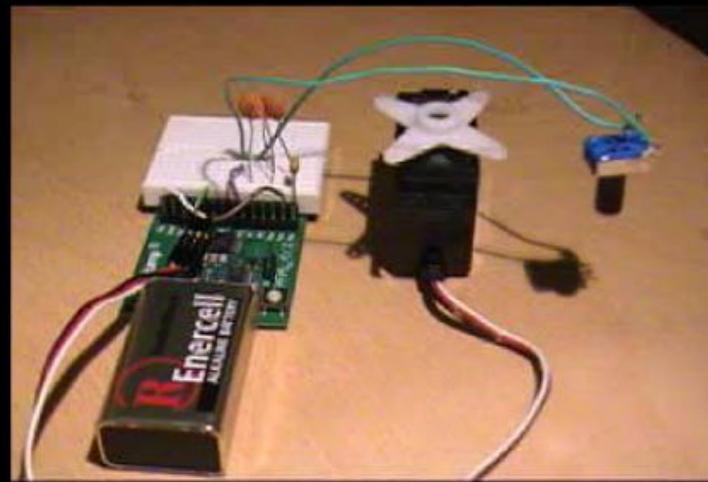
```
int avrageA,avrageB,avrageC,avrageD;  
int Angle=90;  
int counter;  
void setup(){  
Serial.begin(9600); //Beginning Serial Connection  
pinMode(13,OUTPUT); //Servo White wire connection  
for (int i=0; i<20; i++){  
    avrageA=avrageA+analogRead(5);  
    avrageB=avrageB+analogRead(4);  
    avrageC=avrageC+analogRead(3);  
    avrageD=avrageD+analogRead(2);  
}  
avrageA=avrageA/20;  
avrageB=avrageB/20;  
avrageC=avrageC/20;  
avrageD=avrageD/20;  
Serial.println("System Ready");  
Serial.println(avrageA);  
Serial.println(avrageB);  
Serial.println(avrageC);  
Serial.println(avrageD);  
}  
void loop(){  
int inA = analogRead(5);  
int inB = analogRead(4);  
int inC = analogRead(3);  
int inD = analogRead(2);  
if(inA<avrageA/2){  
    Angle=170; counter=0;  
}  
if(inB<avrageB/2){  
    Angle=120; counter=0;  
}  
if(inC<avrageC/2){  
    Angle=60; counter=0;  
}  
if(inD<avrageD/2){  
    Angle=0; counter=0;  
}  
if((inA>avrageA/2)&&(inB>avrageB/2)&&(inC>avrageC/2)&&(inD>avrageD/2)){  
    counter=counter+1;  
    if(counter==100){  
        Angle=90;  
    }  
}  
for(int a=0; a<5; a++){  
    int pulseWidth = Angle*11+500; // See the formula  
    digitalWrite(13,HIGH);  
    delayMicroseconds(pulseWidth);  
    digitalWrite(13,LOW);  
    delay(10);  
}
```

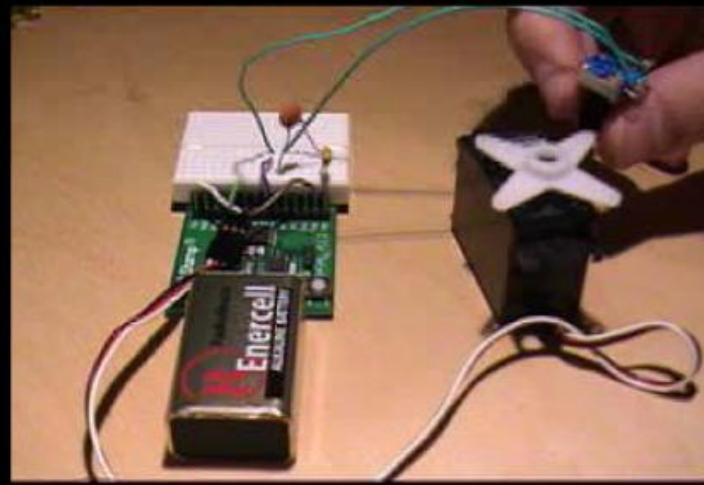
Responsive System

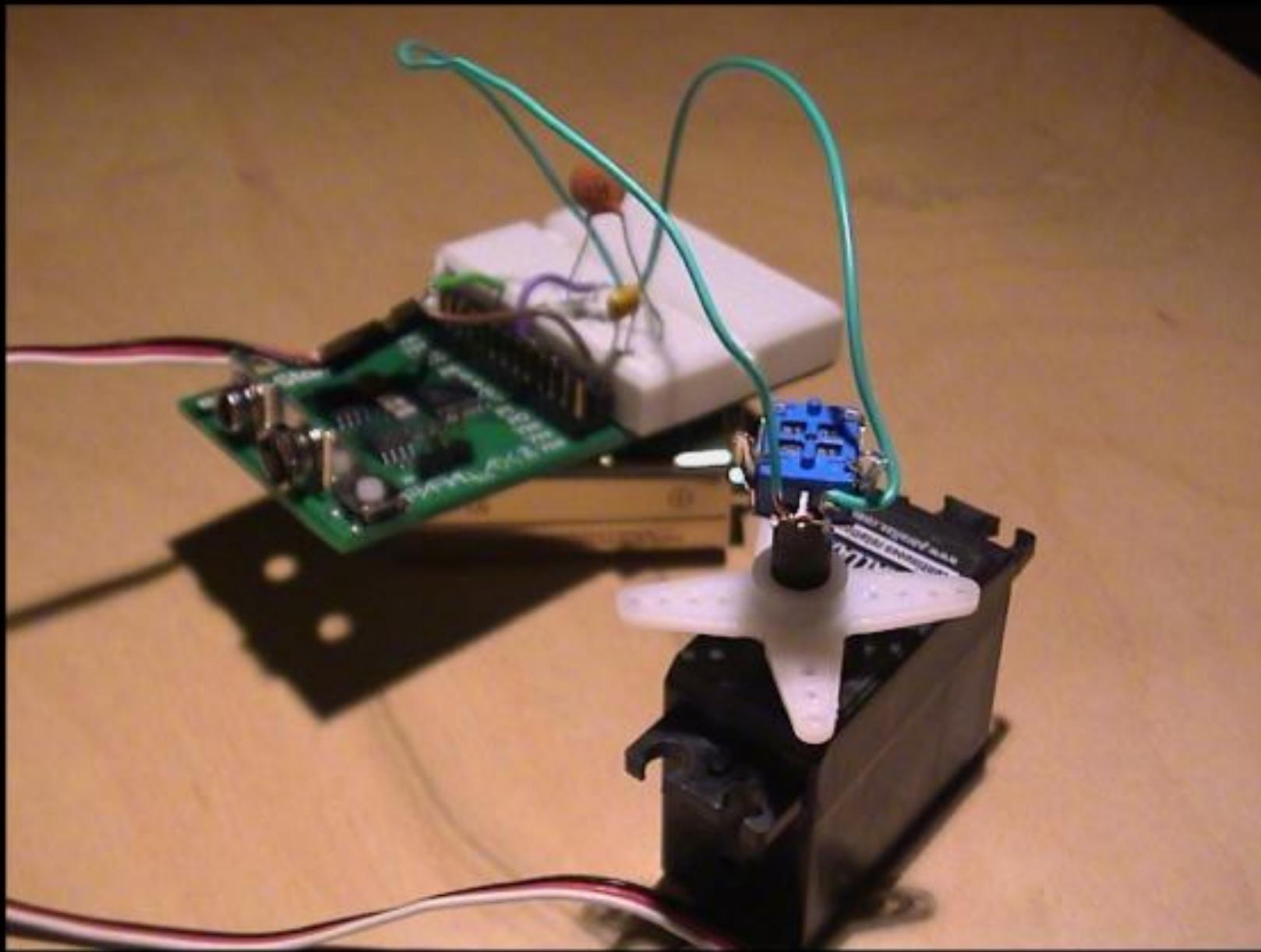
Input Potentiometer-

Output Standard Servo-

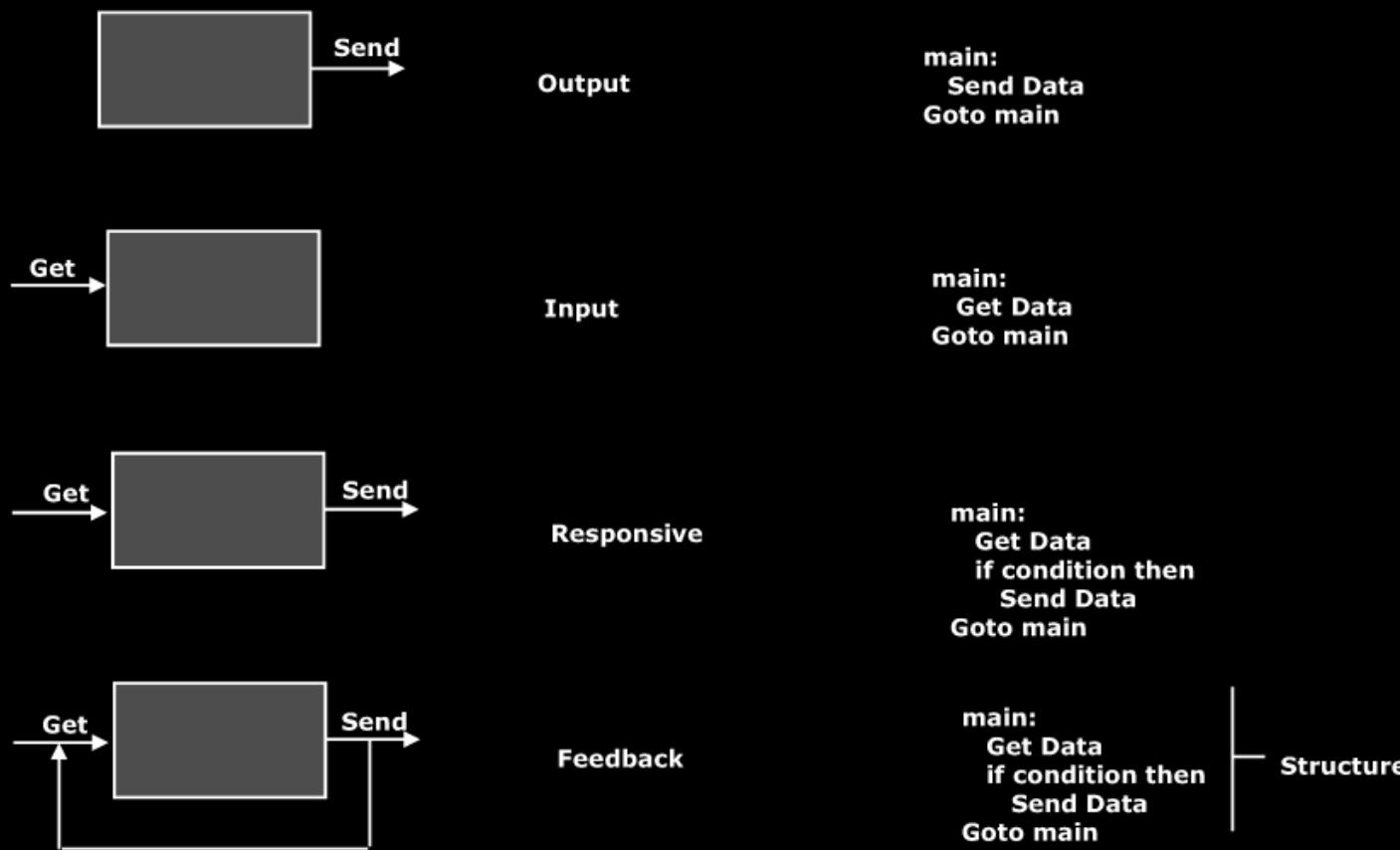








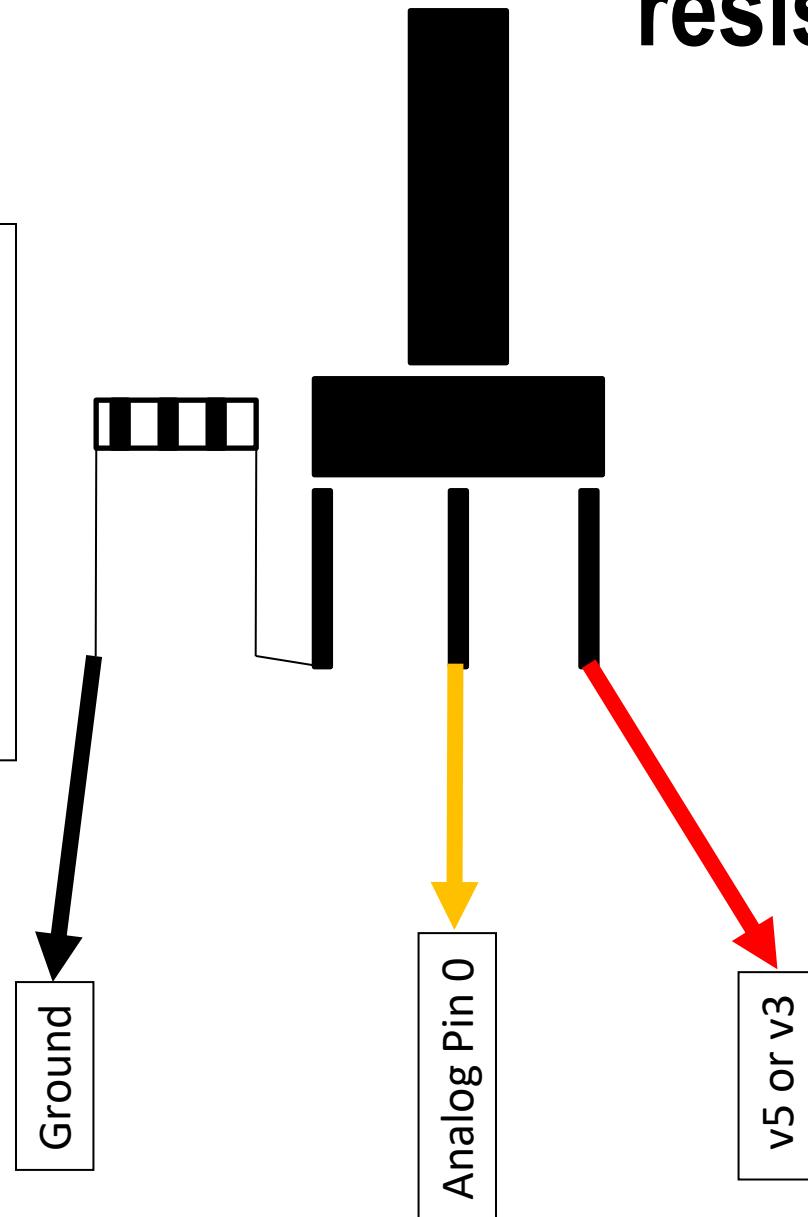
Changing the arrangement of the sensors and actuators in the same responsive system in a way that the response of the actuator to the sensed data can affect the setting of the sensor in some way and as a result lead to a different read of the properties of the environment, will give us a feed back system as opposed to a responsive system



Changing the range of a Potentiometer using resistors

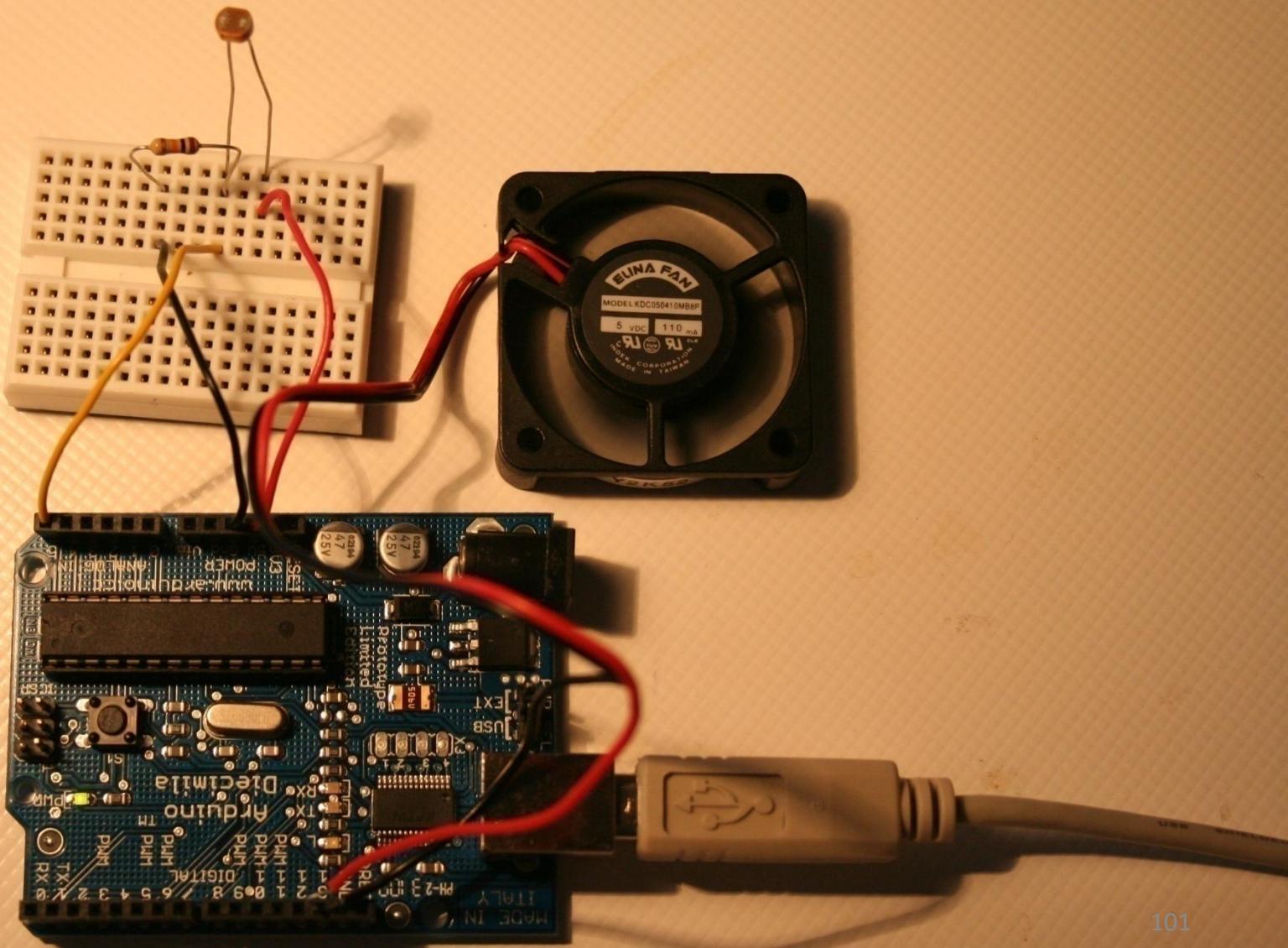
If you add a resistor between the ground and the potentiometer connected to the ground, you change the range of a resistor.

The other way to change the range is connecting the potentiometer to 3v instead of 5v



Analog Input – Digital Output

Photocell controlling Fan



Analog Input – Digital Output

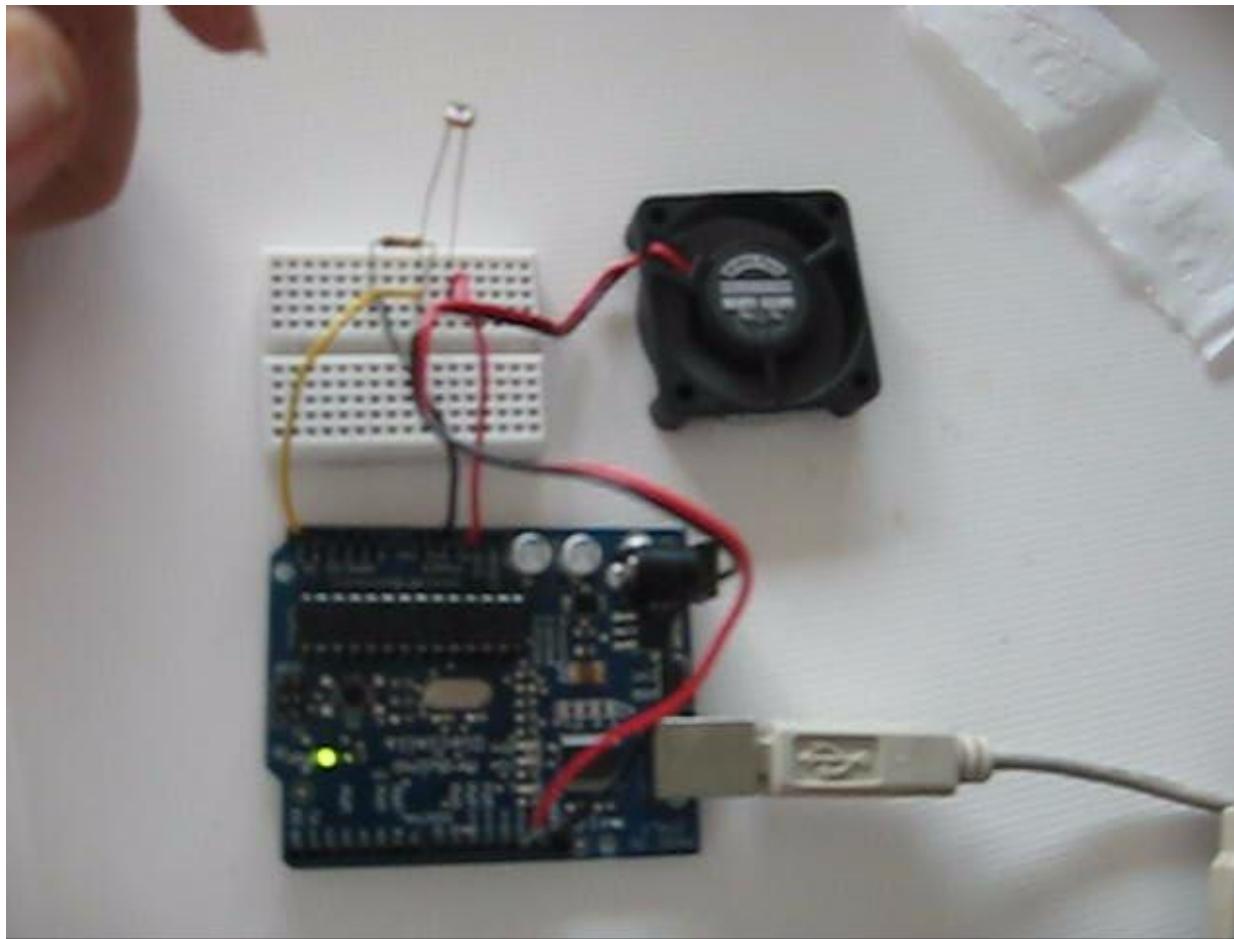
Photocell controlling Fan

```
int avrage;
void setup(){
    Serial.begin(9600); //Beginning Serial Connection
    pinMode(13,OUTPUT); //Servo White wire connection
    for (int i=0; i<20; i++){
        avrage=avrage+analogRead(5);
    }
    avrage=avrage/20;
    Serial.println("System Ready");
    Serial.println(avrage);
}
void loop(){
    int in = analogRead(5);
    if(in<avrage-10)
        digitalWrite(13,HIGH);
    else
        digitalWrite(13,LOW);
    Serial.println(in);
}
```

Analog Input – Digital Output

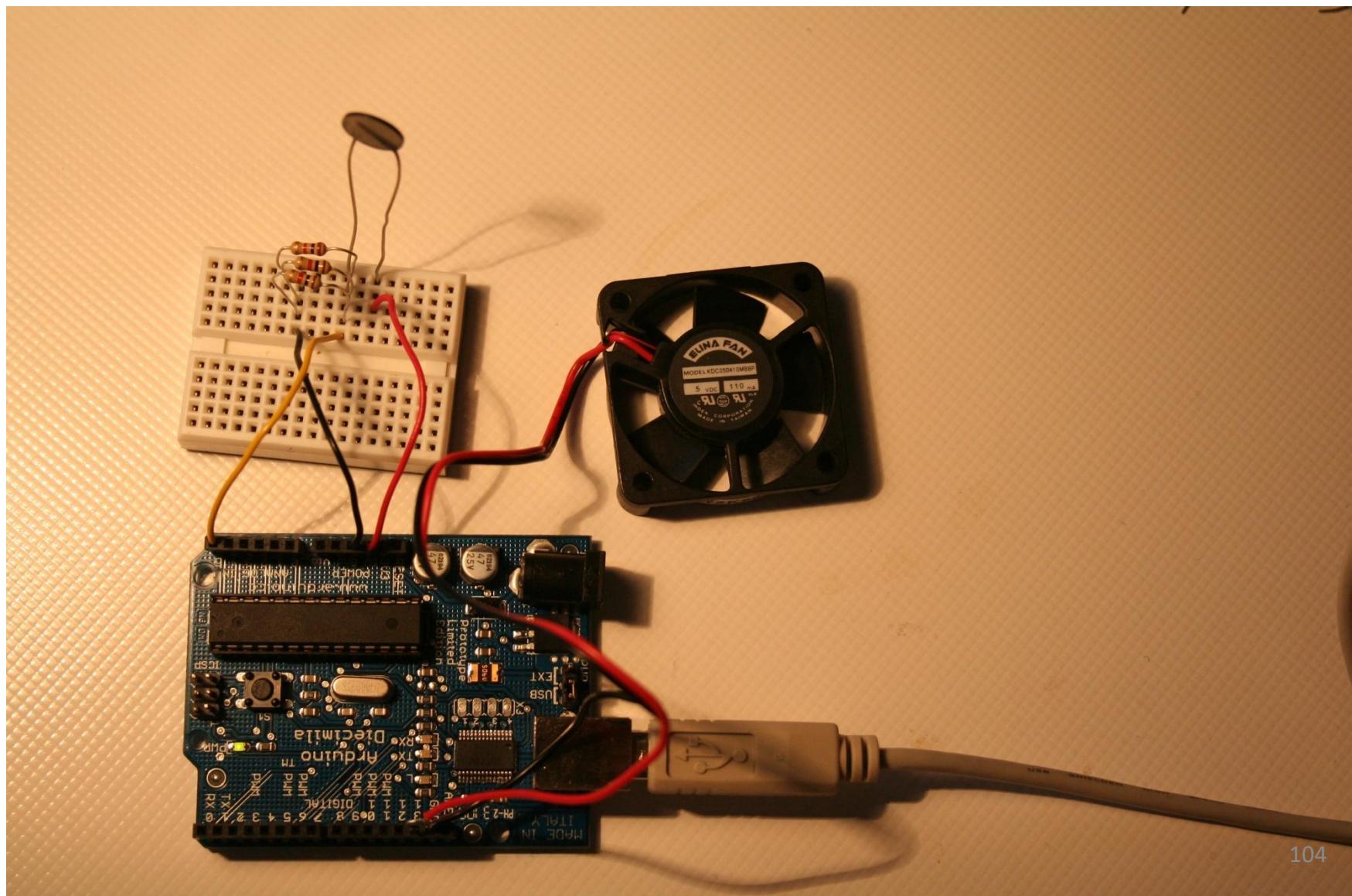
Photocell controlling Fan

Changing the setting to a feedback system



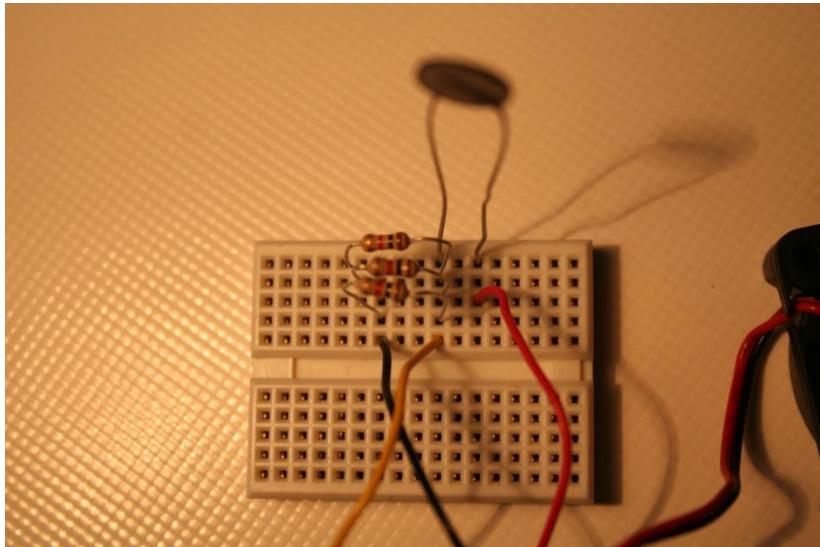
Analog Input – Digital Output

Temperature controlling Fan



Analog Input – Digital Output

Temperatures controlling Fan



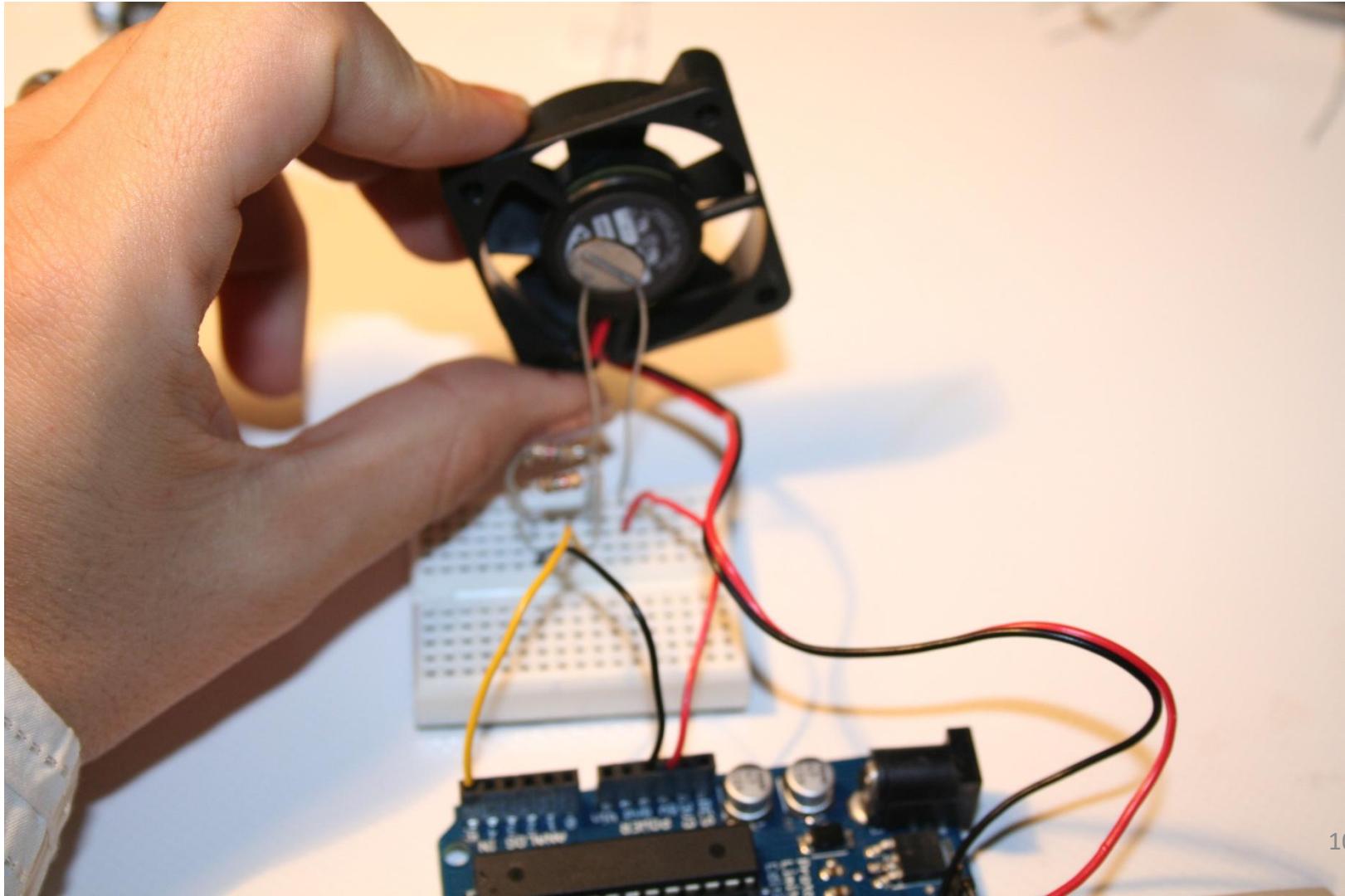
Using Resistance in Parallel to Increase the Sensitivity of the Sensor

```
int avrage;
void setup(){
  Serial.begin(9600); //Begining Serial Connection
  pinMode(13,OUTPUT); //Servo White wire connection
  for (int i=0; i<20; i++){
    avrage=avrage+analogRead(5);
  }
  avrage=avrage/20;
  Serial.println("System Ready");
  Serial.println(avrage);
}
void loop(){
  int in = analogRead(5);
  if(in>avrage-20)
    digitalWrite(13,HIGH);
  else
    digitalWrite(13,LOW);
  Serial.println(in);
}
```

Analog Input – Digital Output

Temperature controlling Fan

Changing a Responsive System to feedback system

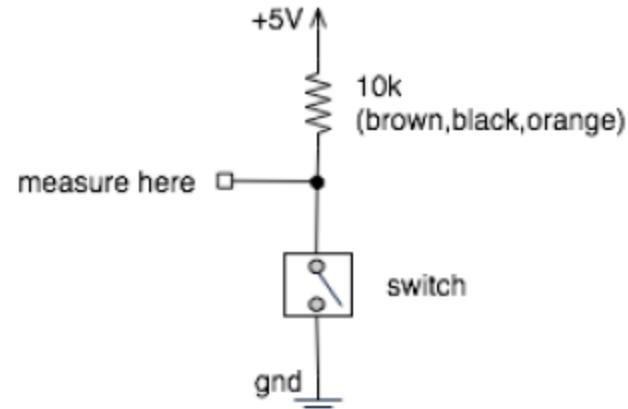
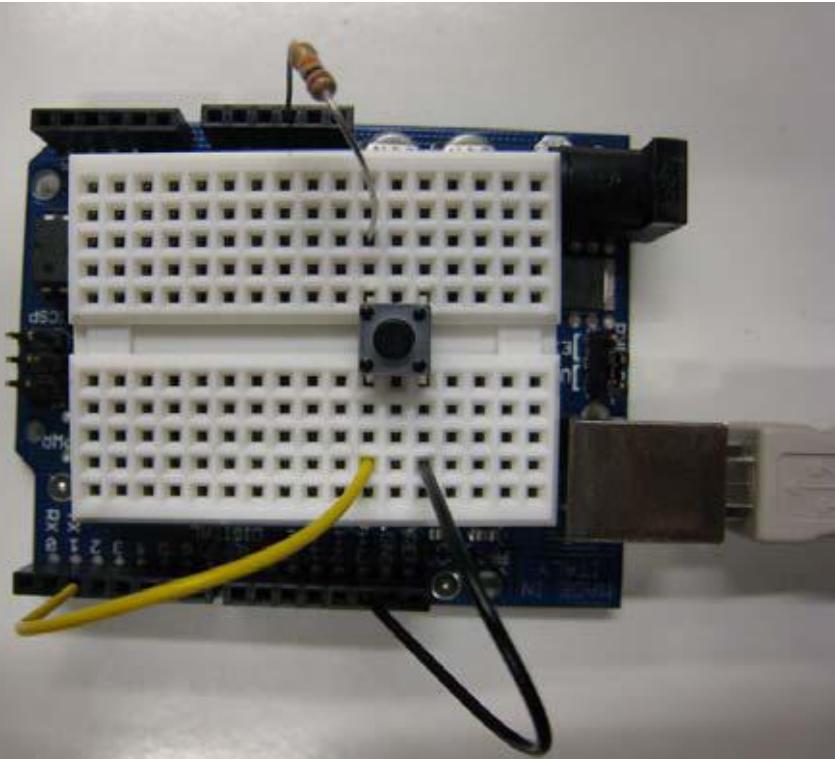


Digital Input – Output to Processing

Controlling the Screen with a Button

1. Controlling an LED with Push Button

A push-button will open (or close) a circuit when pressed. Its circuit is shown below. The black wire goes to ground, the yellow to the pin (2 for this case), and the 10K resistor connects to 5V. If you connect the 10K resistor to ground and the black wire to 5V you inverse the state of the push button.

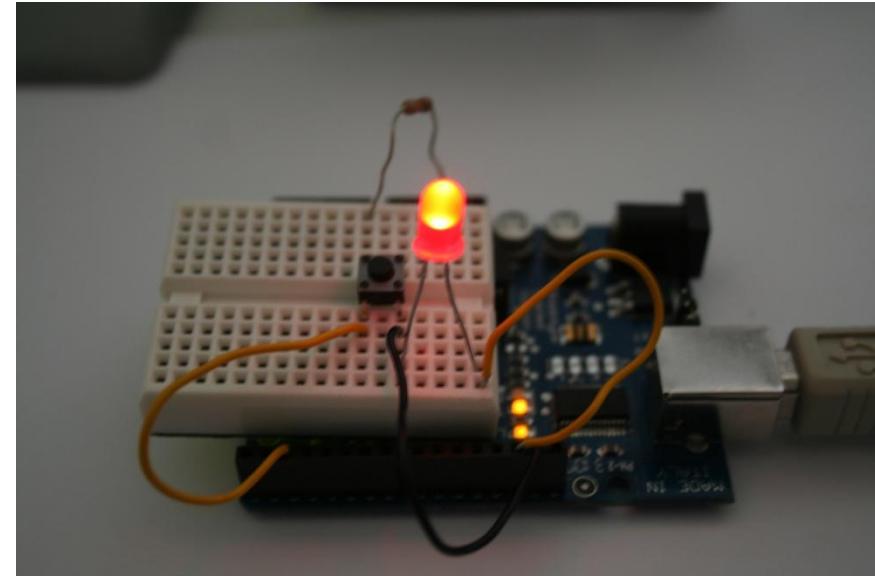
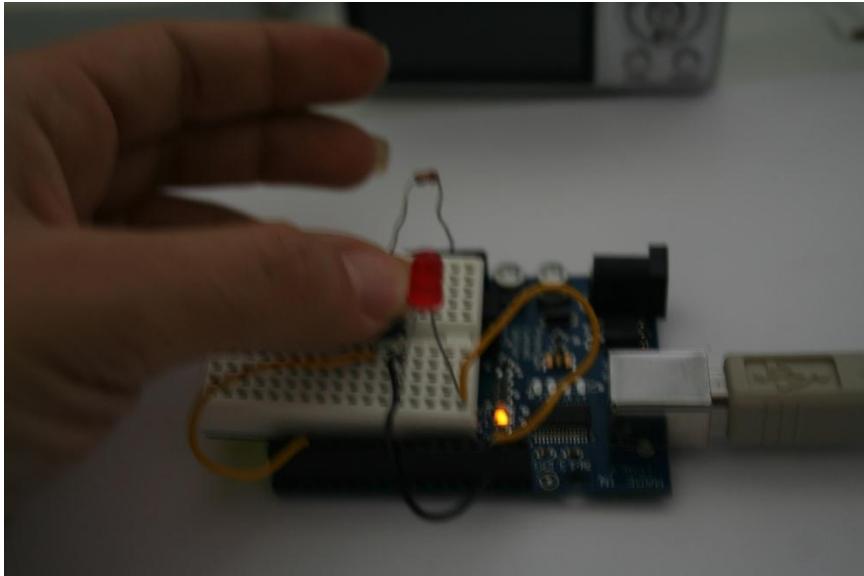


Digital Input – Output to Processing

Controlling the Screen with a Button

1. Controlling an LED with Push Button

A push-button will open (or close) a circuit when pressed. Its circuit is shown below. The black wire goes to ground, the yellow to the pin (2 for this case), and the 10K resistor connects to 5V.



Digital Input – Output to Processing

Controlling the Screen with a Button

1. Controlling an LED with Push Button

A push-button will open (or close) a circuit when pressed. Its circuit is shown below. The black wire goes to ground, the yellow to the pin (2 for this case), and the 10K resistor connects to 5V.

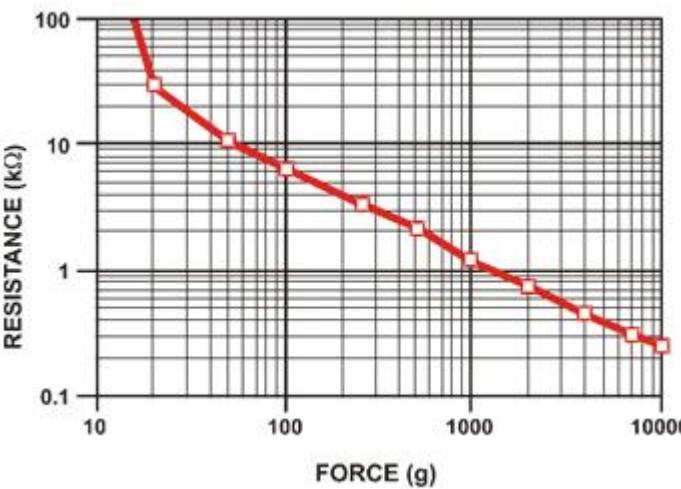
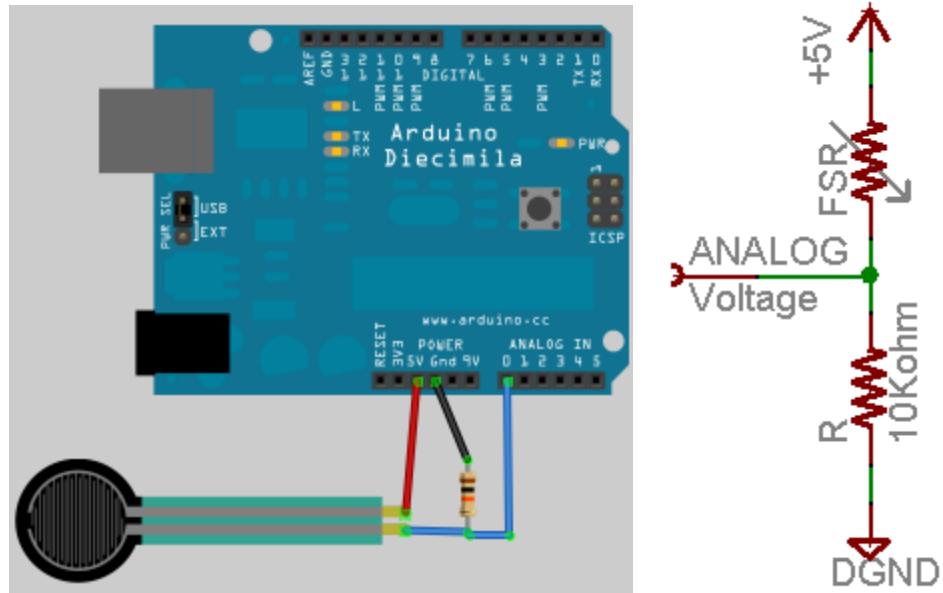
```
void setup(){
    Serial.begin(9600);
    pinMode(13,OUTPUT);
    pinMode(2,INPUT);
}
void loop(){
    int in=digitalRead(2);
    Serial.println(in);
    if (in==0) digitalWrite(13,LOW);
    if (in==1) digitalWrite(13,HIGH);
}
```

Sensor: Force Sensitive Resistor (FSR)

FSRs are sensors that allow you to detect physical pressure, squeezing and weight. FSR is basically a resistor that changes its resistive value (in ohms Ω) depending on how much it's pressed. These sensors are fairly low cost, and easy to use but they're rarely accurate. They also vary some from sensor to sensor perhaps 10%. So basically when you use FSR you should only expect to get ranges of response. While FSRs can detect weight, they're a bad choice for detecting exactly how many pounds of weight are on them. However, for most touch-sensitive applications like "has this been squeezed or pushed and about how much" they're a good deal for the money!

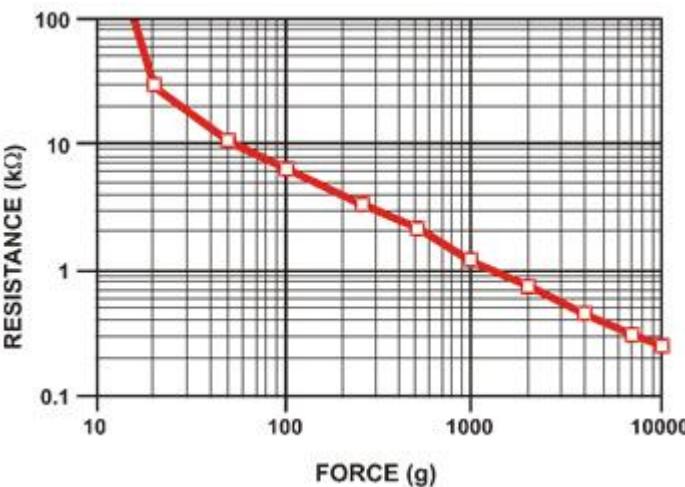
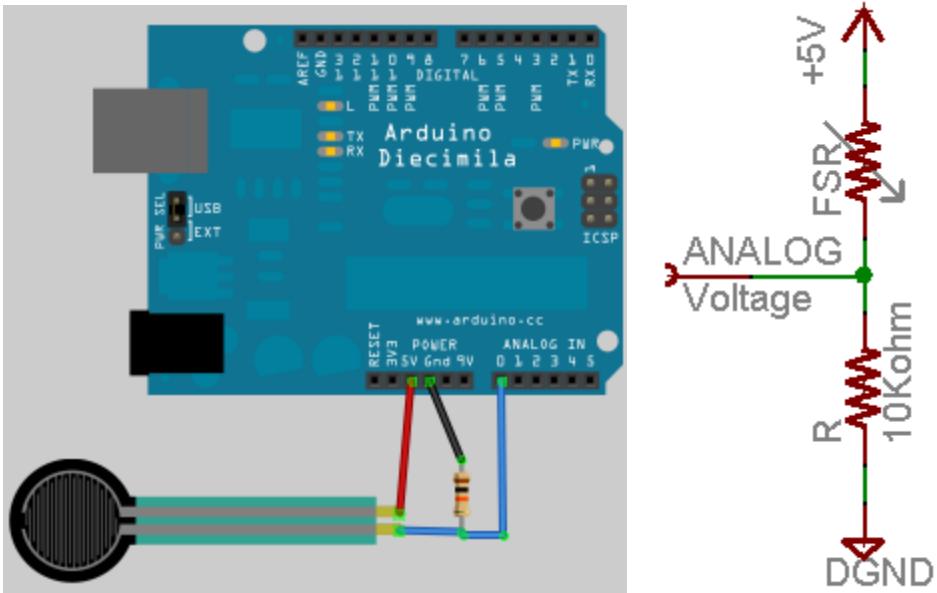
The FSR that is included in your package is Interlink 402 model. The 1/2" diameter round part is the sensitive bit. The FSR is made of 2 layers separated by a spacer. The more one presses, the more of those Active Element dots touch the semiconductor and that makes the resistance go down. FSR's resistance changes as more pressure is applied. When there is no pressure, the sensor looks like an infinite resistor (open circuit), as the pressure increases, the resistance goes down. It is important to notice that the graph isn't really linear (it's a log/log graph) and that at especially low force measurements it quickly goes from infinite to 100K Ω .

Because FSRs are basically resistors, they are non-polarized. That means you can connect them up 'either way' and they'll work just fine! The best way to connect to these is to simply plug them into a breadboard, or use a clamp-style connector like alligator clips, female header, or a terminal block. It is also possible to solder yet, the piece is really delicate



Sensor: Force Sensitive Resistor (FSR)

```
void setup(){  
  Serial.begin(9600);  
}  
void loop(){  
  int in=analogRead(0);  
  Serial.println(in);  
  delay(1000);  
}
```

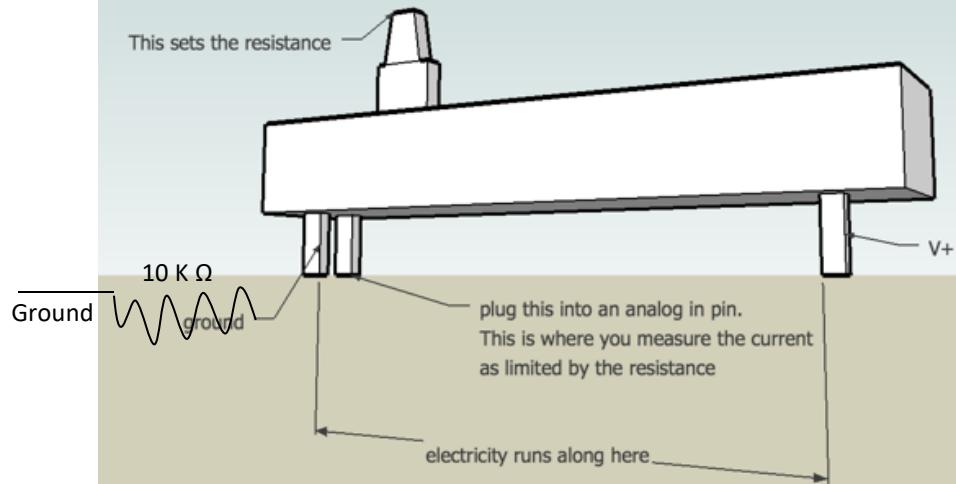


Sensor: Slide Potentiometer (10 KΩ)

Slide Potentiometer is basically a variable resistance with a tactile interface. It can be used to detect linear disposition.

You need a pull down resistor (10K) while connecting the Potentiometer to ground. Connecting the sensor to 5V or 3V changes the range of the sensibility of the sensor. Alternating between the Voltage and Ground pin changes the direction of range of sensibility of the sensor.

```
void setup(){  
Serial.begin(9600);  
}  
void loop(){  
int in=analogRead(0);  
Serial.println(in);  
delay(1000);  
}
```



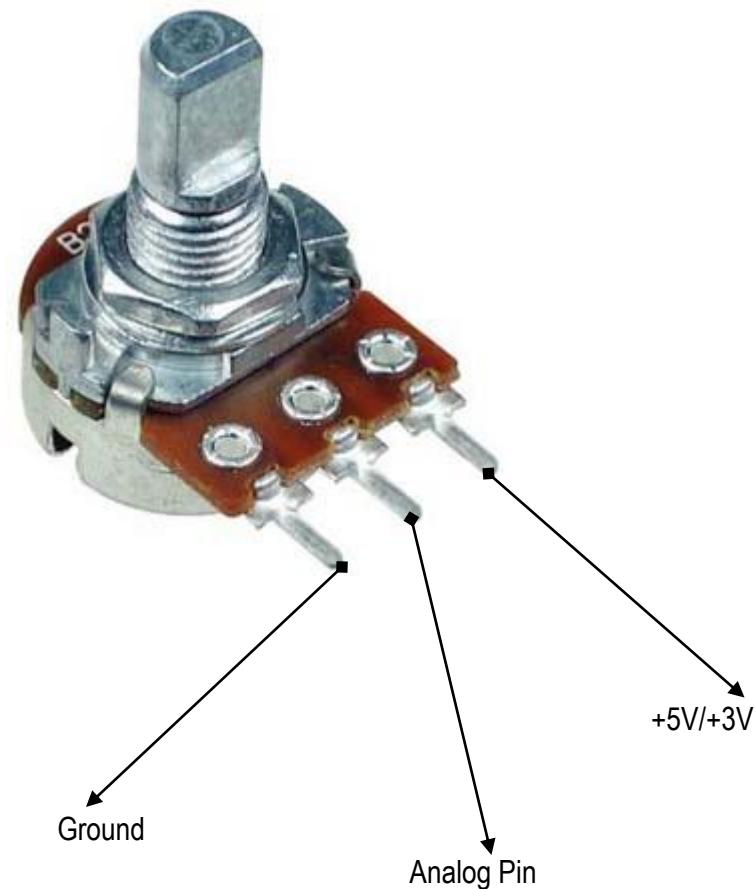
Sensor: Rotary Potentiometer (500 Ω B Single)

Rotary Potentiometer is basically a variable resistance with a tactile interface. It can be used to detect Rotational disposition.

Connecting the sensor to 5V or 3V changes the range of the sensibility of the sensor. Alternating between the Voltage and Ground pin changes the direction of range of sensibility of the sensor.

Keep in mind that rotary potentiometers can be linear or logarithmic. The one that is included in your package is a linear one.

```
void setup(){  
  Serial.begin(9600);  
}  
  
void loop(){  
  int in=analogRead(0);  
  Serial.println(in);  
  delay(1000);  
}
```

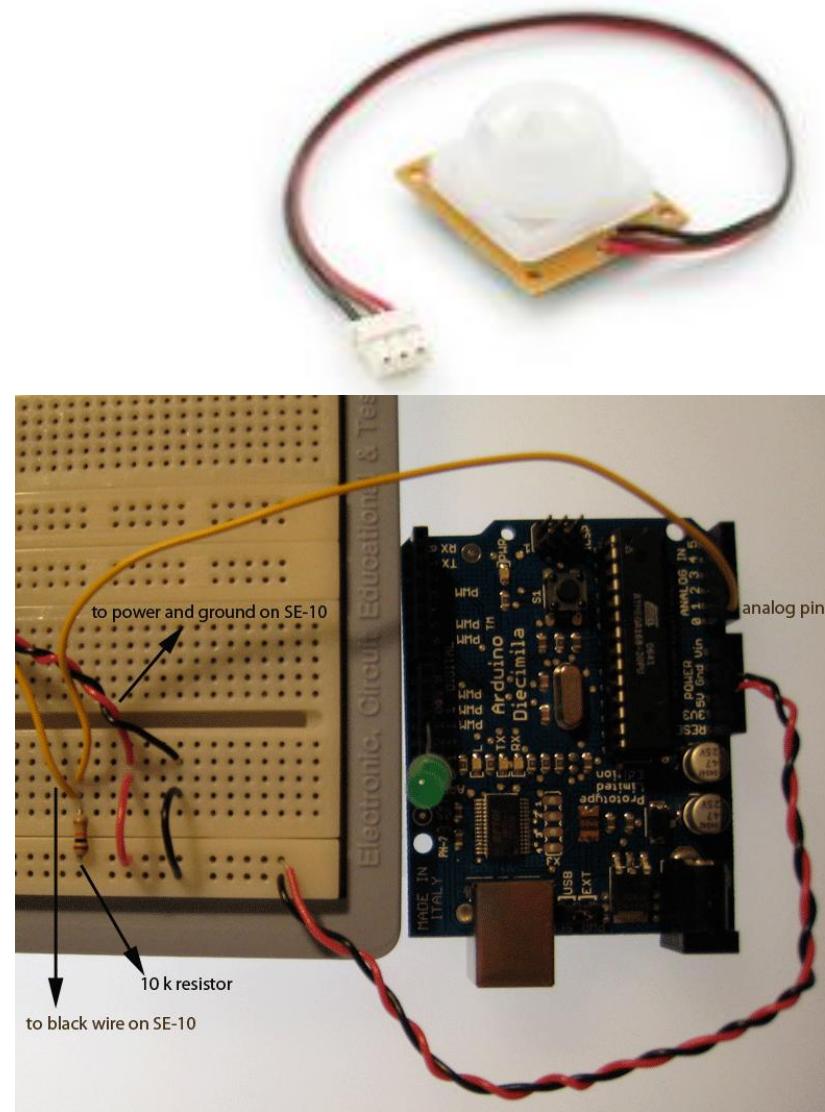


Sensor: PIR Motion Sensor

This is a simple to use motion sensor. Power it up and wait 1-2 seconds for the sensor to get a snapshot of the still room. If anything moves after that period, the 'alarm' pin will go low. Red wire is power (5 to 12V). Brown wire is GND. Black wire is open collector Alarm. The alarm pin is an open collector meaning you will need a pull up resistor on the alarm pin. This means that on one hand the alarm pin is connected to the analog input and on the other hand it is connected to 5 volt via a 10K resistor as shown in the photograph.

Take note that by default the sensor read is 1023 and once motion is detected it alternates between 1023 and 18. so you need to code the sensor in a way that consistent read means no motion and alternating between low and high read means motion.

```
void setup () {  
Serial.begin (9600);  
delay (2000);  
// it takes the sensor 2 sec to scan the area around it before detecting presence.  
}  
void loop(){  
int in=analogRead(0);  
Serial.println(in);  
delay(1000);  
}
```



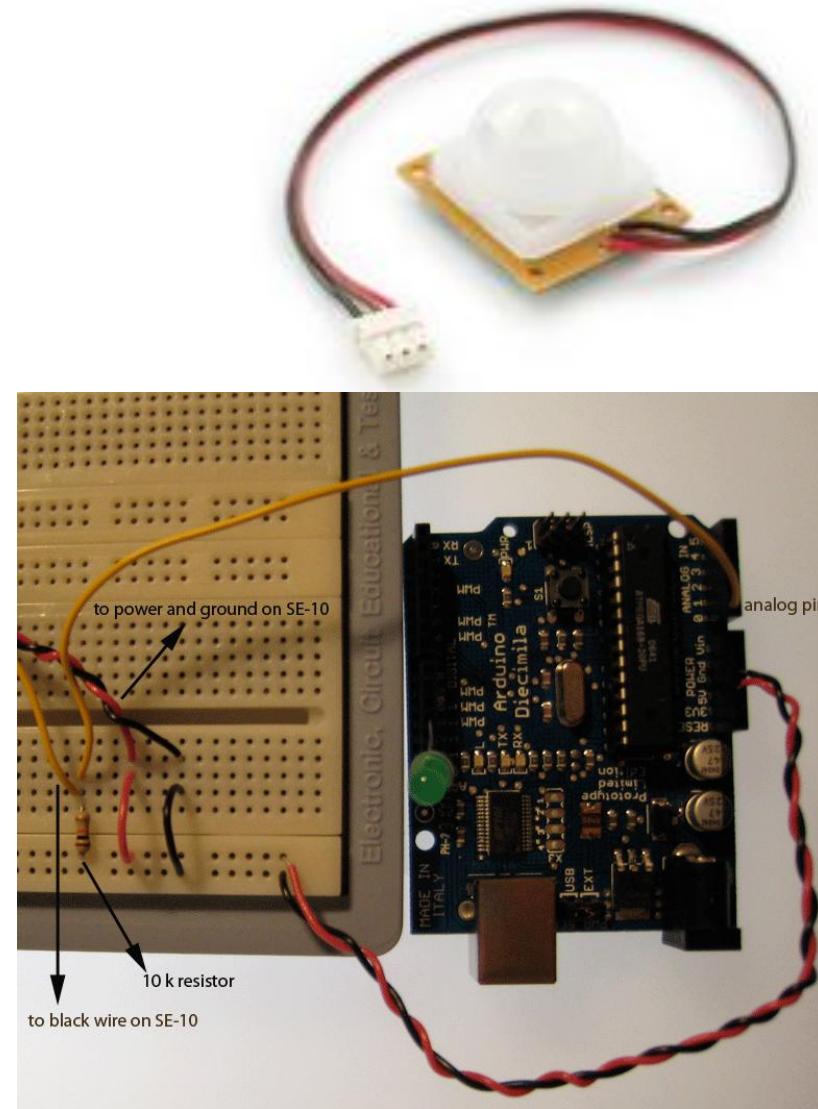
http://www.sparkfun.com/commerce/product_info.php?products_id=8630

<http://itp.nyu.edu/physcomp/sensors/Reports/PIRMotionSensor>

Sensor: PIR Motion Sensor

```
// example for the PIR motion sensor
int timer = 500;
int alarmPin = 0;
int alarmValue = 0;
int ledPin = 11;
void setup () {
    Serial.begin (9600);
    pinMode(ledPin, OUTPUT);
    pinMode(alarmPin, INPUT);
    delay (2000); // it takes the sensor 2 seconds to scan the area around it
}
void loop (){
    alarmValue = analogRead(alarmPin);
    if (alarmValue < 100){
        blinky(); // blinks when the motion has been detected, just for confirmation.
    }
    delay(timer);
    Serial.println (alarmValue);
    delay (10);
}

void blinky(){
for(int i=0; i<3; i++) {
    digitalWrite(11,HIGH);
    delay(200);
    digitalWrite(11,LOW);
    delay(200);
}
}
```



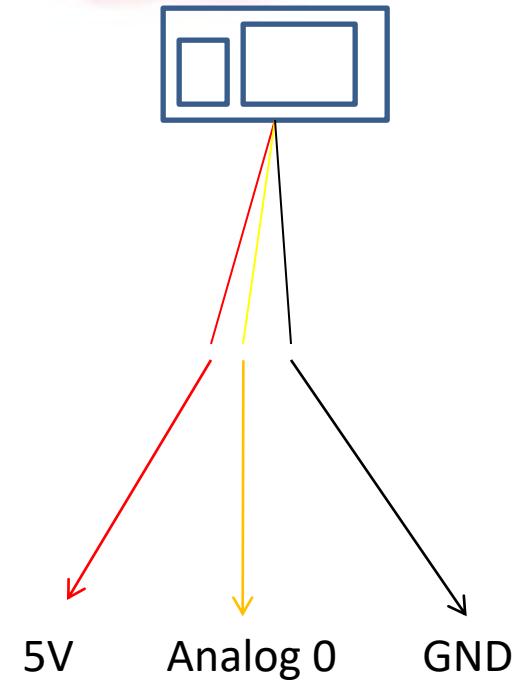
http://www.sparkfun.com/commerce/product_info.php?products_id=8630

<http://itp.nyu.edu/physcomp/sensors/Reports/PIRMotionSensor>

Sensor: GP2D12 Analog Distance Sensor and JST Cable

It is an analog Distance sensor. Connection is easy: Black to Ground, Red to 5V and Yellow to Analog Pin. The one that you have is sensitive between 10-80cm. You can buy ones with different ranges.

```
void setup () {  
  Serial.begin (9600);  
}  
  
void loop(){  
  int in=analogRead(0);  
  Serial.println(in);  
  delay(1000);  
}
```

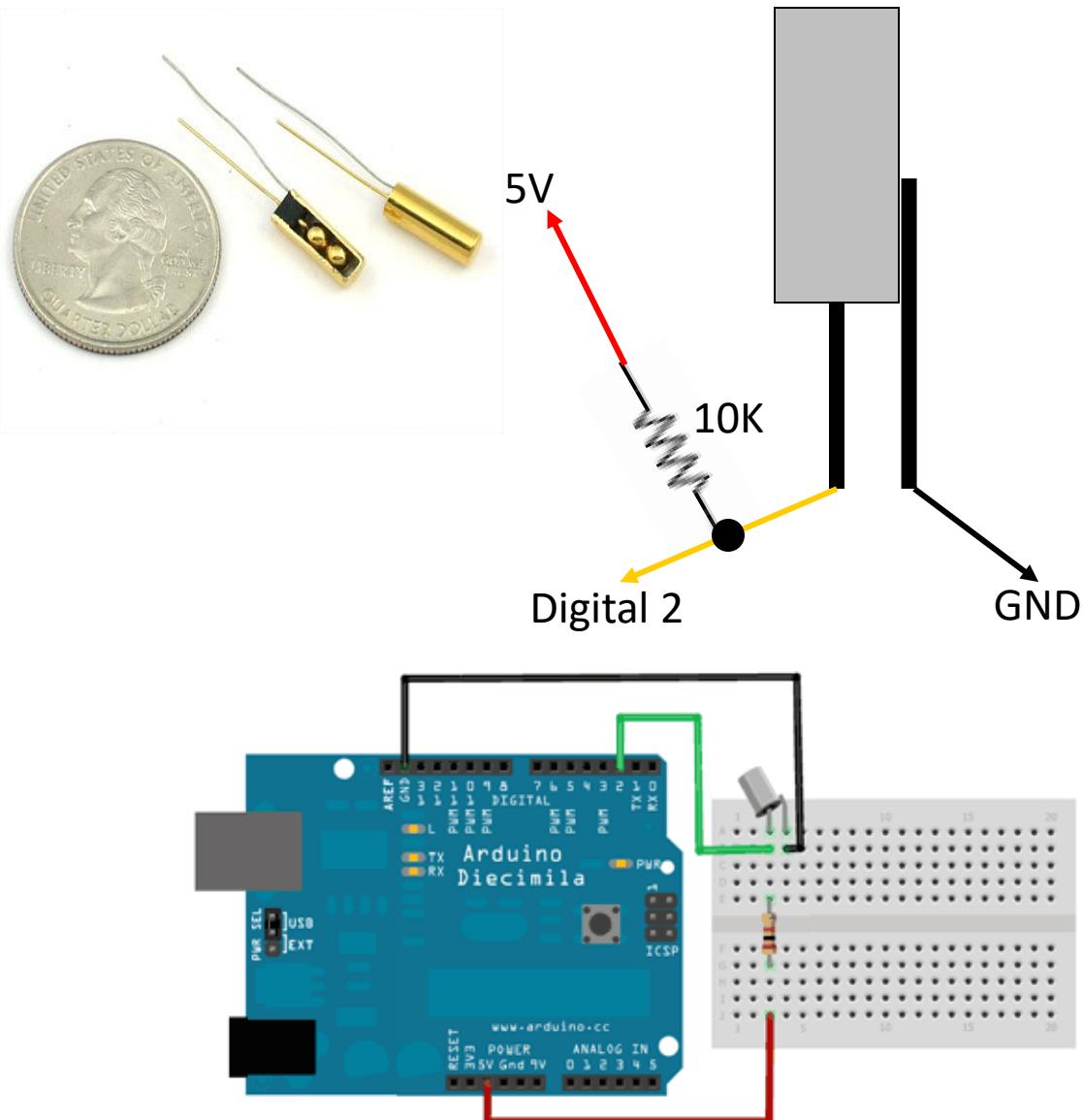


<http://www.lynxmotion.com/p-260-sharp-gp2d12-ir-sensor.aspx>

Sensor: Tilt Ball Switch Diff Angle:30

The "poor man's" accelerometer! Tilt sensors are switches that can detect motion/orientation. The metal tube has a little metal ball that when tilted upright, the ball rolls onto the contacts sticking out causing them to close together.

```
int val;  
void setup()  
{  
  Serial.begin(9600); // sets the serial port to 9600  
  pinMode(3, INPUT);  
}  
void loop()  
{  
  val = digitalRead(2); // read digital I/O pin 2  
  Serial.println(val); // prints the value read  
  delay(1000);  
}
```

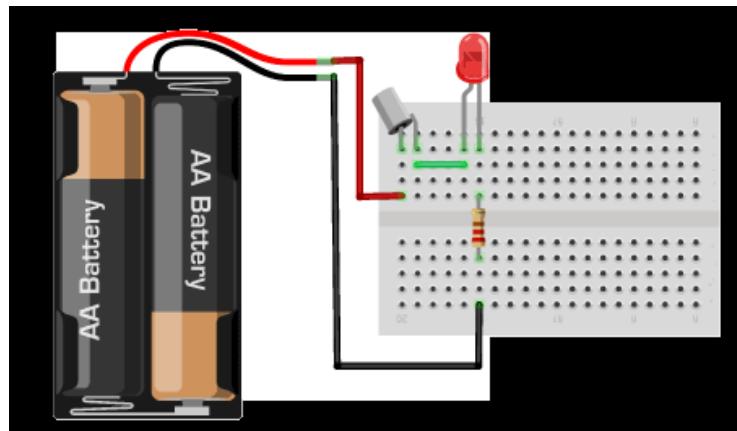


http://www.adafruit.com/index.php?main_page=product_info&products_id=173

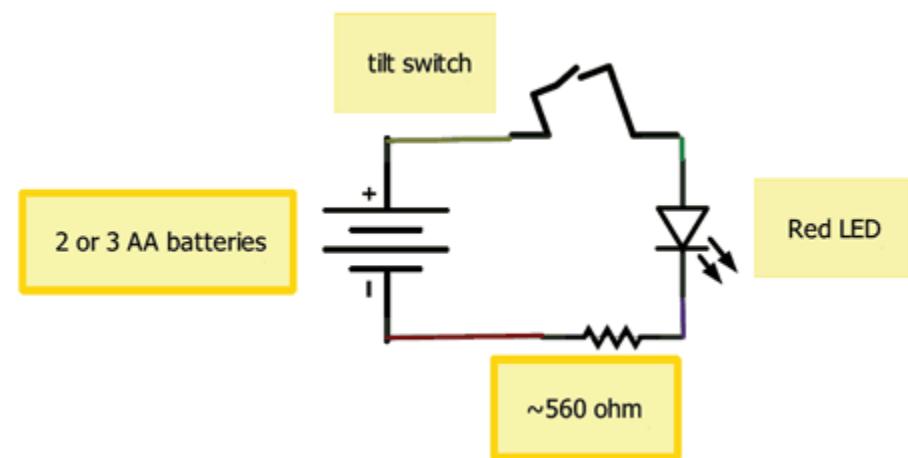
<http://www.ladyada.net/learn/sensors/tilt.html>

Sensor: Tilt Ball Switch

The "poor man's" accelerometer! Tilt sensors are switches that can detect basic motion/orientation. The metal tube has a little metal ball that rolls around in it, when its tilted upright, the ball rolls onto the contacts sticking out of end and shorts them together.



OR

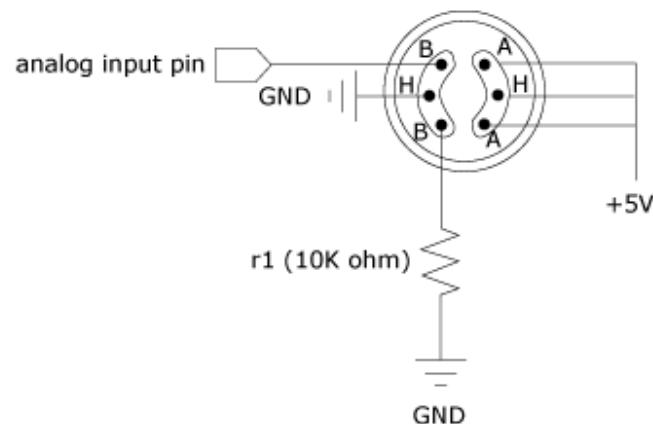


http://www.adafruit.com/index.php?main_page=product_info&products_id=173

<http://www.ladyada.net/learn/sensors/tilt.html>

Sensor: MQ7 Air Quality Sensor

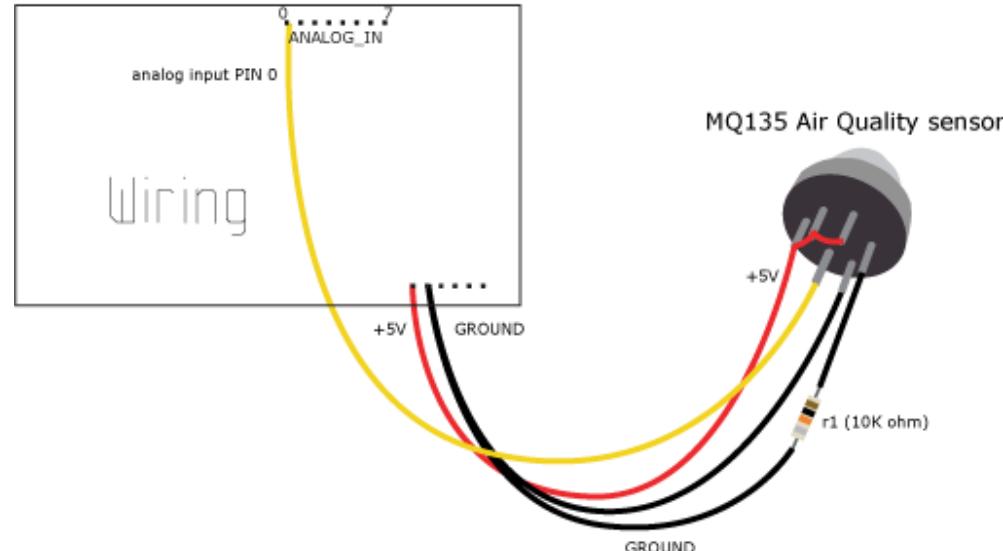
This is a simple-to-use Carbon Monoxide (CO) sensor, suitable for sensing CO concentrations in the air. The MQ-7 can detect CO concentrations anywhere from 20 to 2000ppm.



This sensor has a high sensitivity and fast response time. The sensor's output is an analog resistance. The drive circuit is very simple; all you need to do is power the heater coil with 5V, add a load resistance, and connect the output to an ADC.

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

void loop(){
  int in=analogRead(0);
  Serial.println(in);
}
```



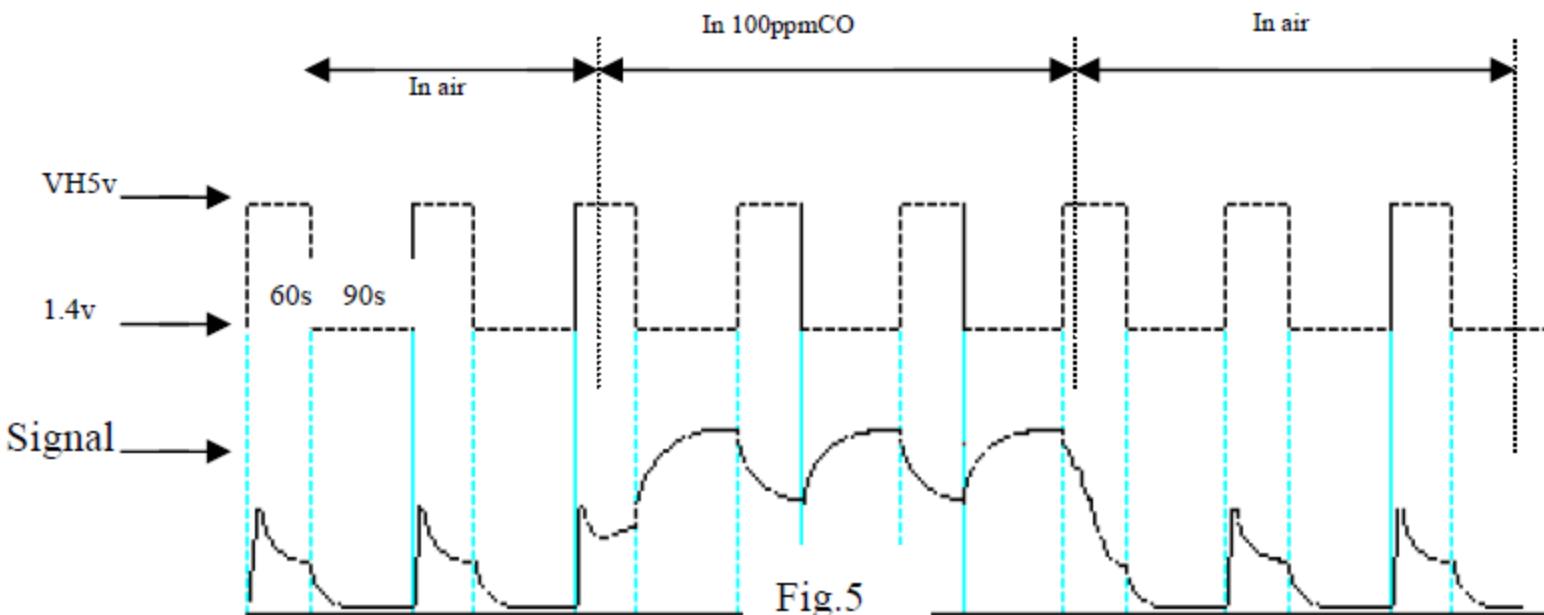
<http://wiring.org.co/learning/basics/airqualitymq135.html>

<https://www.parallax.com/Portals/0/Downloads/docs/prod/sens/MQ-7Datasheet.pdf>

Sensor: MQ7 Air Quality Sensor

This is a simple-to-use Carbon Monoxide (CO) sensor, suitable for sensing CO concentrations in the air. The MQ-7 can detect CO concentrations anywhere from 20 to 2000ppm.

This sensor has a high sensitivity and fast response time. The sensor's output is an analog resistance. The drive circuit is very simple; all you need to do is power the heater coil with 5V, add a load resistance, and connect the output to an ADC.



<http://wiring.org.co/learning/basics/airqualitymq135.html>

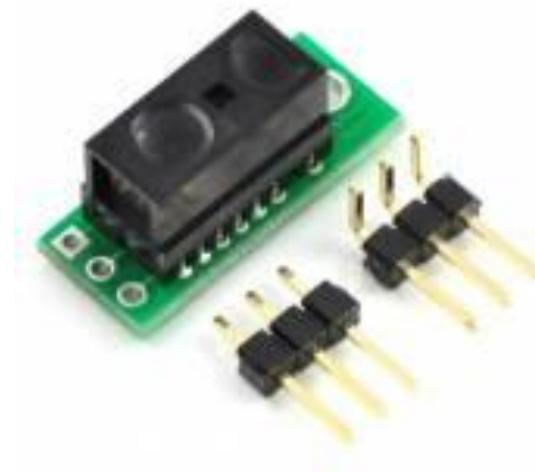
<https://www.parallax.com/Portals/0/Downloads/docs/prod/sens/MQ-7Datasheet.pdf>

*Blow to the sensor, copy the serial data and visualize in excel to see the change

Sensor: IR Distance Sensor 2-10 cm

This small digital distance sensor detects objects between 2 and 10 cm (0.8" and 4") away. With its quick response time, small size, and low current draw, this sensor is a good choice for non-contact object detection, and our compact carrier PCB makes it easy to integrate into your project.. The Power is 5V.

```
void setup(){  
    Serial.begin(9600);  
}  
void loop(){  
    int in=analogRead(0);  
    Serial.println(in);  
}
```

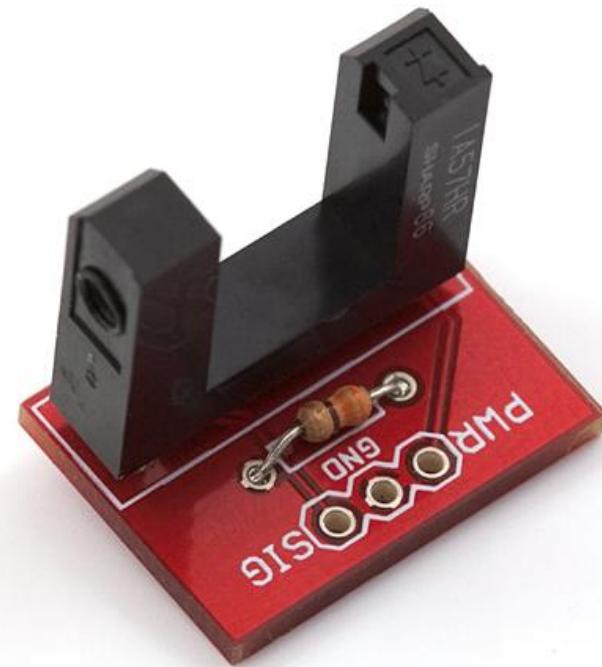


<http://www.pololu.com/catalog/product/1134>

Sensor: IR Distance Sensor 2-10 cm

This sensor is composed of an infrared emitter on one upright and a shielded infrared detector on the other. By emitting a beam of infrared light from one upright to the other, the sensor can detect when an object passes between the uprights, breaking the beam. Used for many applications including optical limit switches, pellet dispensing, general object detection, etc. Gap width = 10mm. The Power is 5V.

```
void setup(){  
  Serial.begin(9600);  
  pinMode(12,INPUT);  
}  
  
void loop(){  
  int in=digitalRead(12);  
  Serial.println(in);  
}
```



http://www.sparkfun.com/commerce/product_info.php?products_id=9322

http://www.sparkfun.com/commerce/product_info.php?products_id=9299

Rules of combining Resistors

If R₁ and R₂ are connected serial:

$$R_3 = R_1 + R_2$$

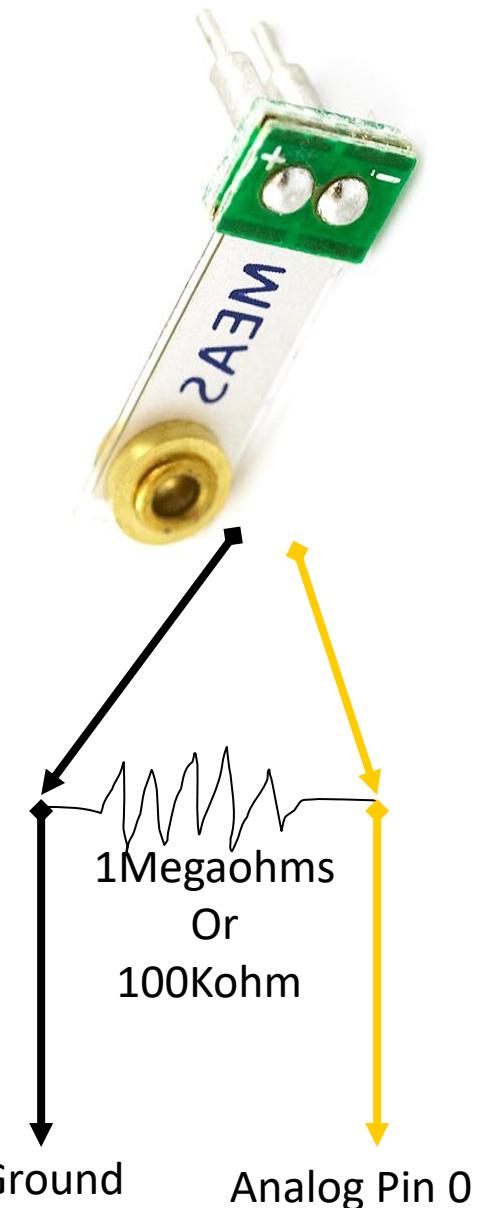
If R₁ and R₂ are Connected Parallel

$$\frac{1}{R_3} = \frac{1}{R_1} + \frac{1}{R_2}$$

Sensor: Piezo Vibration Sensor - Small Horizontal

Piezo Vibration sensors can be used as nock sensors. The Minisense 100 is a low-cost cantilever-type vibration sensor loaded by a mass to offer high sensitivity at low frequencies. Useful for detecting vibration and 'tap' inputs from a user. A small AC and large voltage (up to +/-90V) is created when the film moves back and forth. A simple resistor should get the voltage down to ADC (Analog Digital Conversion) levels. Can also be used for impact sensing or a flexible switch.

```
void setup() {  
    Serial.begin(9600);  
}  
void loop() {  
    int val = analogRead(0);  
    if (val>10)Serial.println(val);  
    if (val>10) digitalWrite(13,HIGH); //Turn on LED Connected to pin 13  
    if (val<=10) digitalWrite(13,LOW); //Turn off LED Connected to pin 13  
}
```



http://www.sparkfun.com/commerce/product_info.php?products_id=9198

<http://forums.adafruit.com/viewtopic.php?f=8&t=15280>

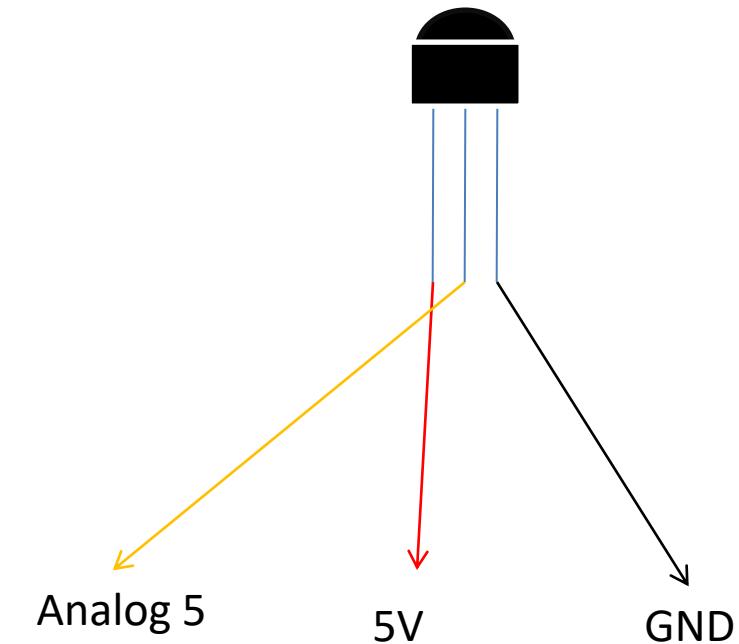
www.arduino.cc/en/Tutorial/KnockSensor

Sensor: Temp Sensor LM35DZ(0-100) or LM335A (-40-100)

Piezo Vibration sensors can be used as nock sensors. The Minisense 100 is a low-cost cantilever-type vibration sensor loaded by a mass to offer high sensitivity at low frequencies. Useful for detecting vibration and 'tap' inputs from a user. A small AC and large voltage (up to +/-90V) is created when the film moves back and forth. A simple resistor should get the voltage down to ADC (Analog Digital Conversion) levels. Can also be used for impact sensing or a flexible switch.

Make sure you are not wiring it Vice Versa because this will ruin the unit

```
void setup(){  
Serial.begin(9600); //Beginning Serial Connection  
}  
void loop(){  
float in = analogRead(0); // Reading Sensed data from Arduino  
in=(5.0 * in* 100.0)/1023.0; //convert the analog data to temperature  
Serial.println(in); // Writing Sensed Data to Serial Port  
}
```



<http://pscmpf.blogspot.com/2008/12/arduino-lm35-sensor.html>

<http://www.ladyada.net/learn/sensors/tmp36.html>

http://www.adafruit.com/index.php?main_page=product_info&cPath=35&products_id=165

http://www.sparkfun.com/commerce/product_info.php?products_id=9438

Motion Detection with Movement State Switches [libelium.com]

/*
 Parallel PIR Sensor and Detecting Motion
 Source: <http://www.arduino.cc/playground/Code/PIRsense>
 The code switches an LED according to the sensor output:
 Motion detected>> LED On, No motion detected>> LED off
 Also, the begining and the end of a continuous motion is
 determined.

PIR detects motion upto 20 ft away by using a Fresnel lens
 and infrared-sensitive elemnt to detect changing pattern
 of passive infrared emitted by objects in its vicinity.

The sensors output will be HIGH when motion is detected
 Yet, even if motion is present, it goes to LOW from time-
 to time, as if no motion is present.

This program ignores LOW-phases shorter than a given time,
 assuming continuous motion is present during these phases.

*/
 int calibrationTime = 30; //Time for the sensor to calibrate in seconds

long unsigned int lowIn; //
 long unsigned int pause = 5000; //Margin for detection of continuous motion in milliseconds

boolean lockLow = true;
 boolean takeLowTime;

int pirPin = 3; //Digital pin that the PIR sensor is connected to
 int ledPin = 13; //Digital pin that LED is connected to

void setup(){

Serial.begin(9600);

pinMode(pirPin, INPUT);

pinMode(ledPin, OUTPUT);

//Calibration of sensor starts

digitalWrite(pirPin, LOW);

Serial.println();

Serial.print("calibrating sensor ");

for(int i = 0; i < calibrationTime; i++){
 Serial.print(".");
 delay(1000);

}

Serial.println(" done");

Serial.println("SENSOR ACTIVE");

delay(50);

//Calibration of sensor ends

}

void loop(){

if(digitalRead(pirPin) == HIGH){

//motion is detected

//turn the LED on

digitalWrite(ledPin, HIGH);

//if we were previously in noMotion State

if(lockLow){

//Now, we are in Motion State

lockLow = false;

Serial.println("--");

Serial.print("motion detected at ");

Serial.print(millis()/1000);

Serial.println(" sec");

delay(50);

// If sensor goes to low state, note the time

takeLowTime = true;

}

if(digitalRead(pirPin) == LOW){

digitalWrite(ledPin, LOW); //the led visualizes the sensors output pin state

if(takeLowTime){

lowIn = millis(); //save the time of the transition from high to LOW

takeLowTime = false; //make sure this is only done at the start of a LOW phase

}

if(!lockLow && millis() - lowIn > pause) //if the sensor detect no motion and you are in motionState and the duration of this state is more than 5 seconds

lockLow = true; //Accept that we are actually in noMotionState

Serial.print("motion ended at "); //output

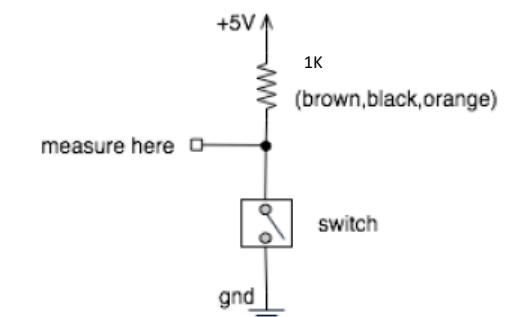
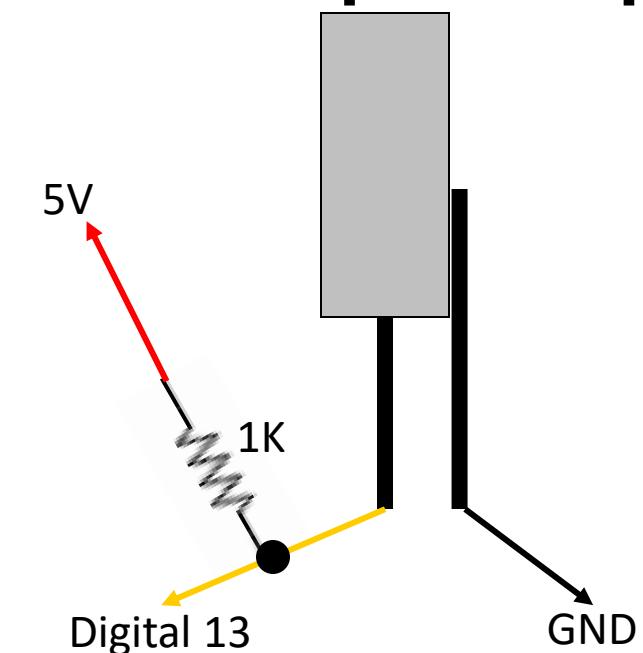
Serial.print((millis() - pause)/1000); //Calculate the time of the ending of the continuous motion

Serial.println(" sec");

delay(50);

}

}

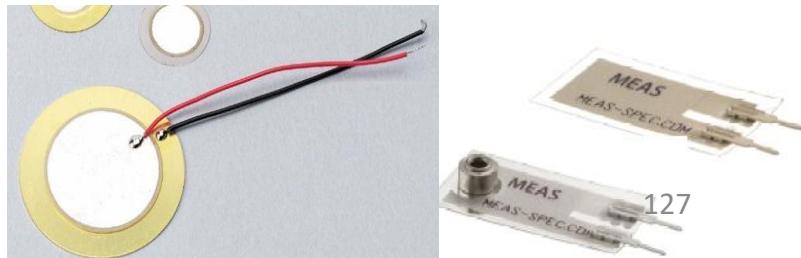
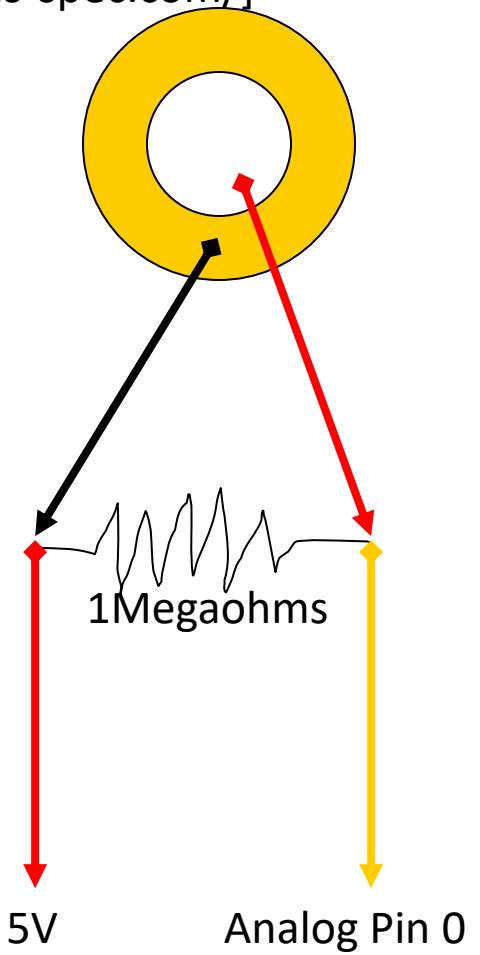


Arduino - Analog Sensor - Detecting nocking[sound] using a piezo
The code is detecting nocking and respond to it by controling an LED
Piezo and Piezo vibration Sensor[<http://www.meas-spec.com/>]

```
//Connect LED to digital pin 13 and Ground
int ledPin = 13;
//Connect knockSensor to analog pin 0 and 5V
int knockSensor = 0;
int val = 0;
int THRESHOLD = 1000;

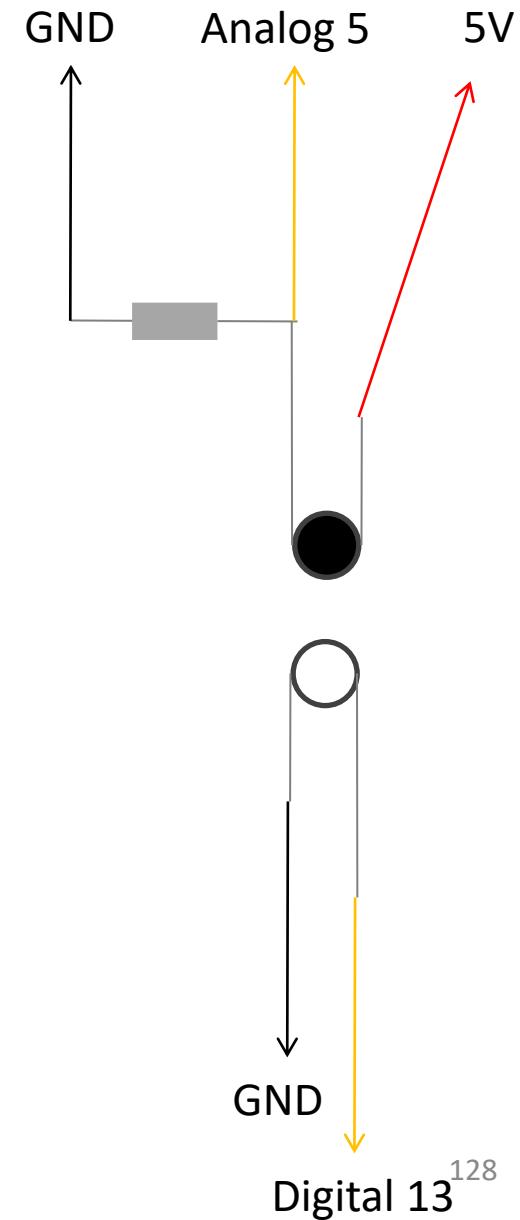
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  val = analogRead(knockSensor);
  if (val >= THRESHOLD) {
    digitalWrite(ledPin, LOW);
  }
  if (val < THRESHOLD) {
    digitalWrite(ledPin, HIGH);
  }
  Serial.println(val);
  delay(100); // we have to make a delay to avoid overloading the serial port
}
```



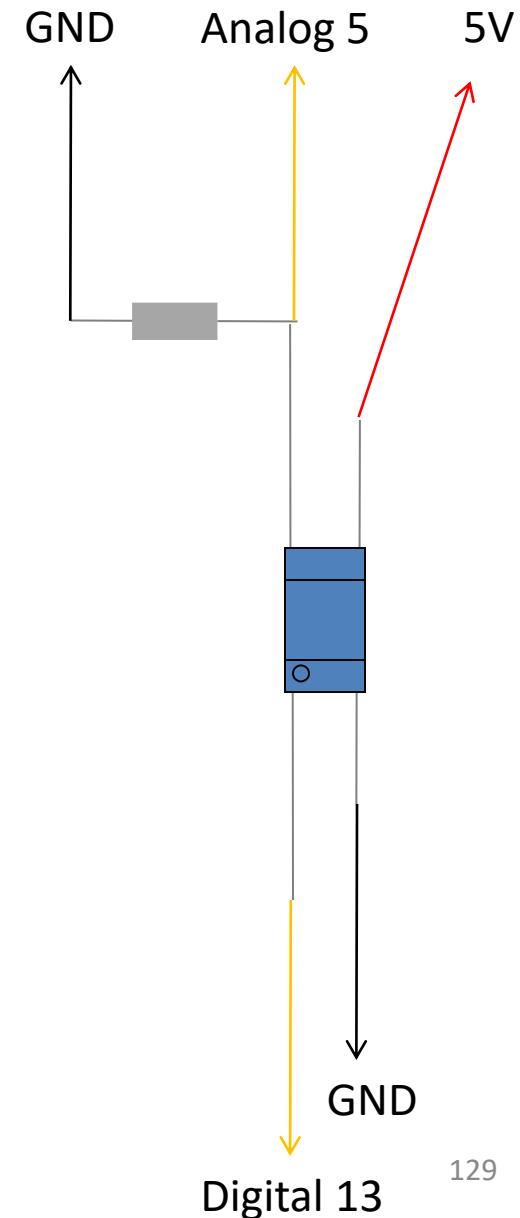
IR transmitter and Receiver

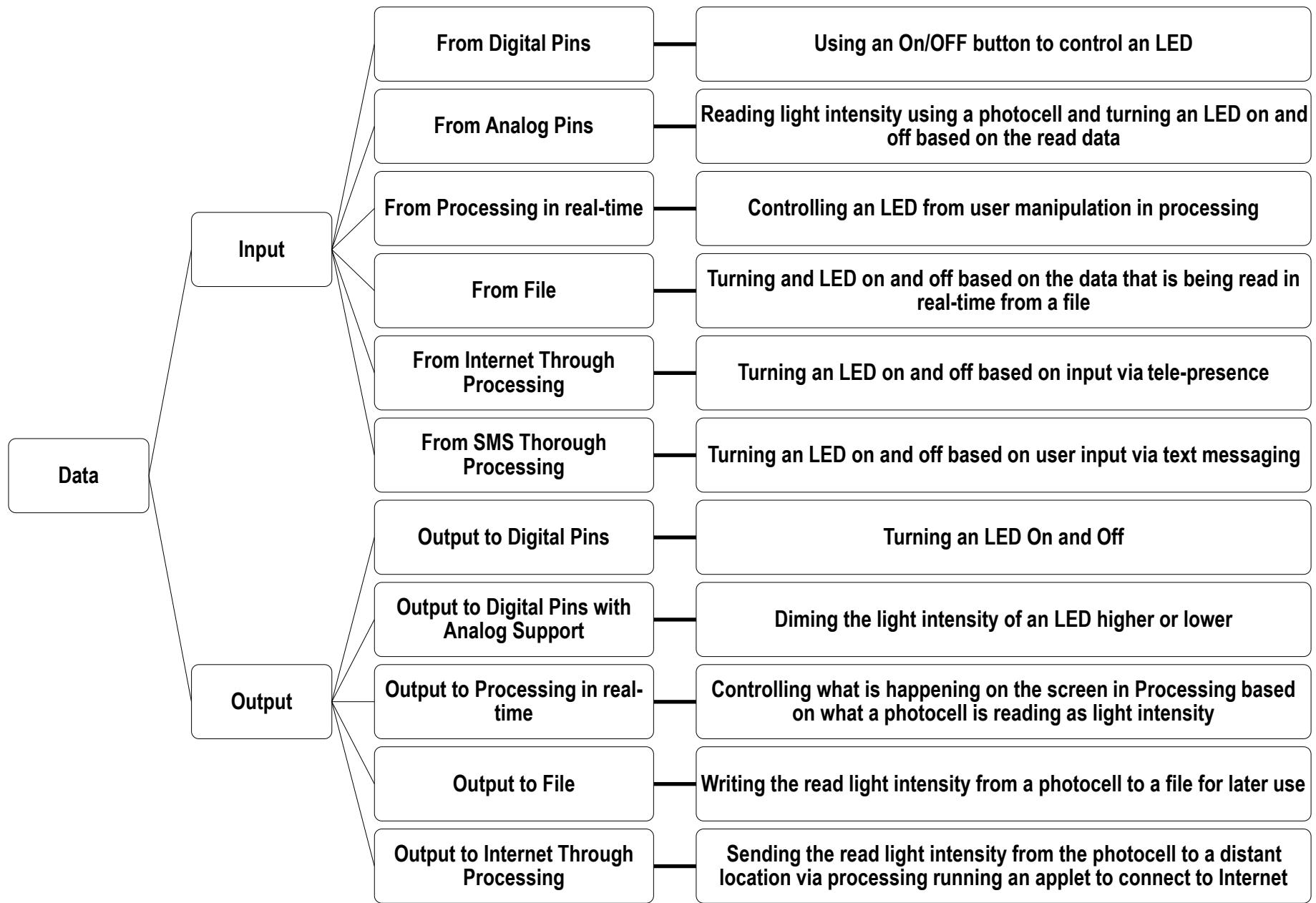
```
void setup(){  
Serial.begin(9600); //Beginning Serial Connection  
//connect infrared LED to digital pin 13  
pinMode(13,OUTPUT);  
digitalWrite(13,HIGH);  
}  
void loop(){  
int in = analogRead(5); // Reading Sensed data from Arduino  
Serial.println(in); // Writing Sensed Data to Serial Port  
}
```



IR transmitter and Receiver-Photointerruptor

```
void setup(){  
Serial.begin(9600); //Beginning Serial Connection  
//connect infrared LED to digital pin 13  
pinMode(13,OUTPUT);  
}  
void loop(){  
int in = analogRead(5); // Reading Sensed data from Arduino  
digitalWrite(13,HIGH);  
Serial.println(in); // Writing Sensed Data to Serial Port  
}
```





CSE423: Embedded System

Summer-2020



Understanding Arduino Code/Sketch (1)

```
void setup()
{
}
void loop()
{
}
```

But it compiles without error!

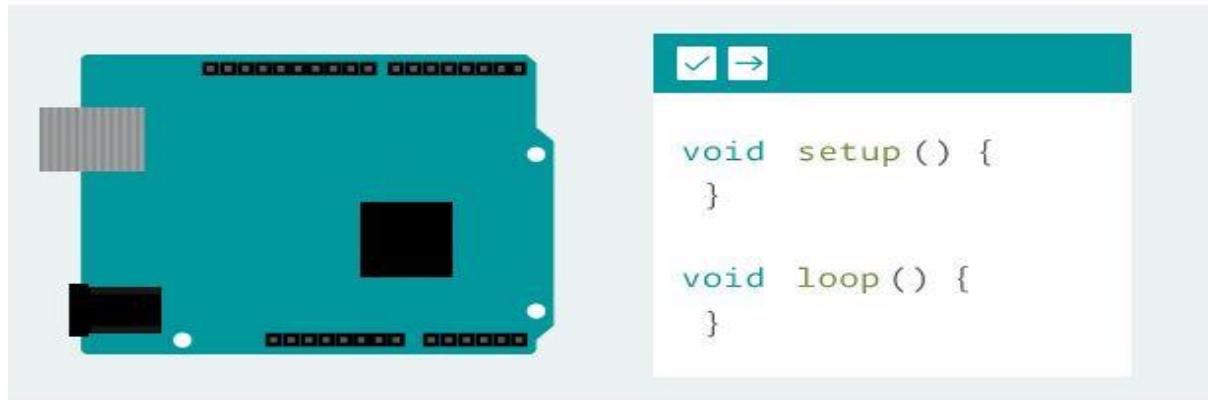
Todays Lecture



- *Understanding Arduino Code (Sketch) step by step*
- *Implementation of code*

Understanding Codes

- Based on **C/C++**
- The basic function has two required sub-routines:
 - **void setup()**
 - **void loop()**
- Includes and **define** as defined as **C**





Understanding Codes

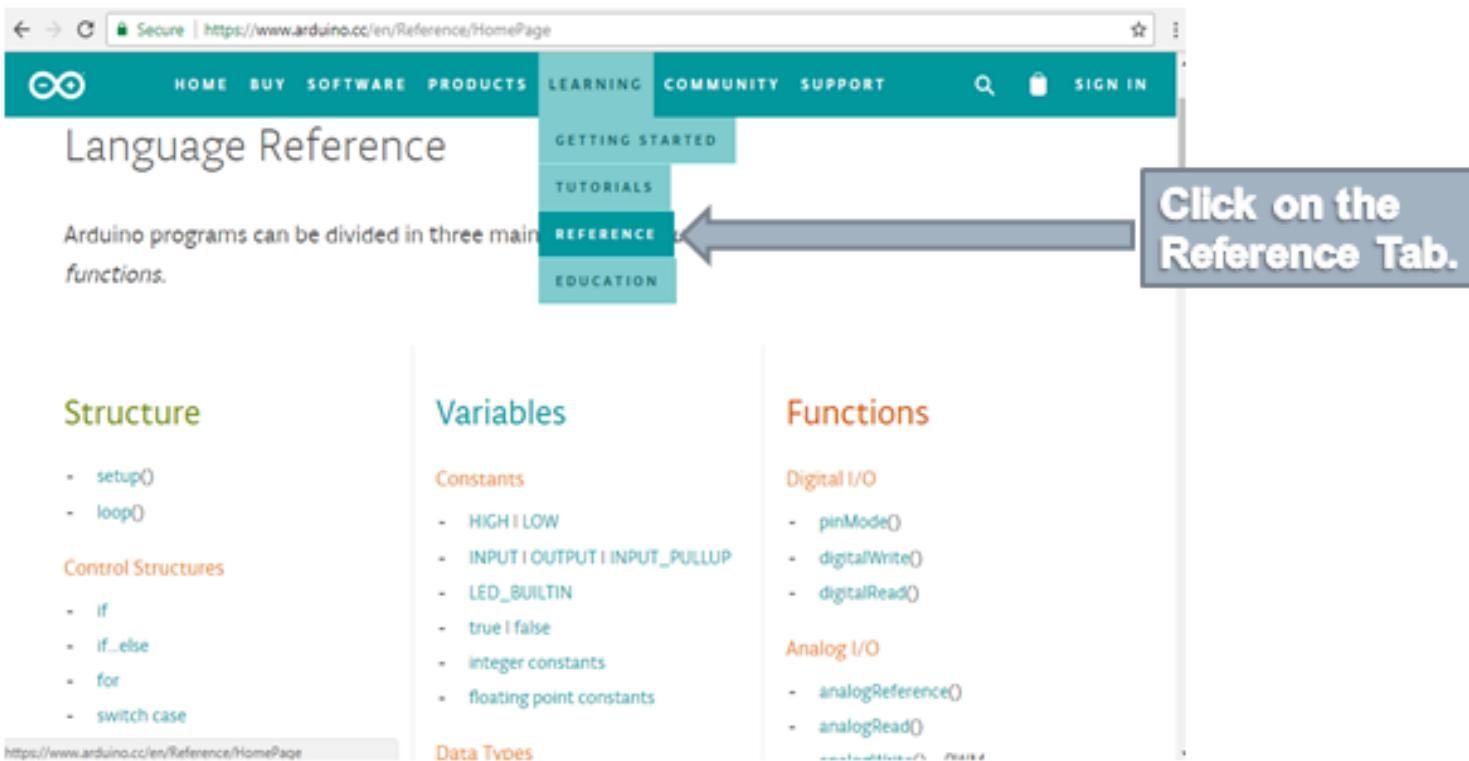
- A simple program: Do Nothing

```
void setup()
{
}
void loop()
{
}
```

But it compiles without error!

Understanding Codes

- Link: <https://www.arduino.cc/en/Reference/HomePage>



Click on the Reference Tab.

Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case

<https://www.arduino.cc/en/Reference/HomePage>

Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Data Types

Functions

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()



Understanding Codes: Constants

- **Digital:**
 - **HIGH** | **LOW** (Defining Pin Levels)
 - **true** | **false** (Defining Logic Levels)
- **GPIO Configuration:**
 - **INPUT** | **OUTPUT** (Defining Pin Levels)
- **Numerical:**
 - **Integer:** **B11010101**, **123**, **-57**, **0x3C**
 - **Float:** **-1.2**, **1.7e5**, **-62E-12**



Understanding Codes: Data Types

int	long	char	float
16b signed	32b signed	8b signed	32b signed
unsigned int	unsigned long	boolean	double
16b unsigned	32b unsigned	1b/8b binary	32b signed
word	byte	string	array
16b unsigned	8b unsigned	8b*nchar " " signed	8b*nelem unsigned
short	unsigned char	String	void
16b signed	8b signed	?? signed	0b signed

Understanding Codes: Basic Operators

Arithmetic	Comparison	Logic/Pointer	Bitwise	Assignment
=	==	&&	&	++
+	!=			--
-	<	!	^	+ =
*	>		~	-- =
/	<=	*var	<<	*=, /=
%	>=	&var	>>	&=, =

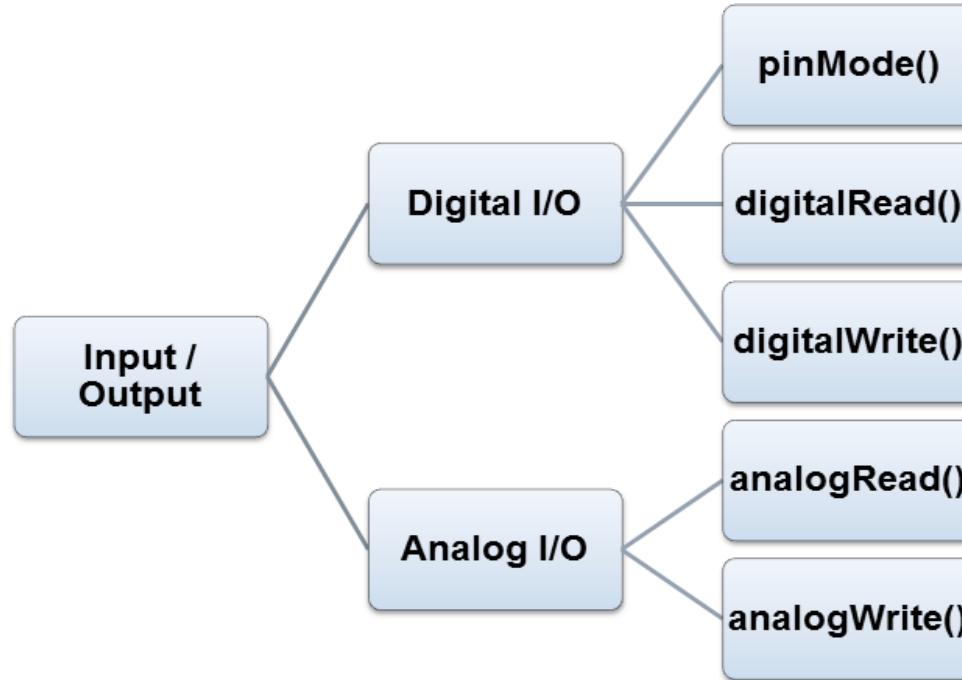


Understanding Codes: Flow controls

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



Understanding Codes: Basic I/O



Understanding Codes: Digital I/O



□ 1. pinMode()

pinMode()

Description

Configures the specified pin to behave either as an input or an output.

Syntax

pinMode(pin, mode)

Parameters

pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT (see the [digital pins](#) page for a more complete description of the functionality.)

Returns

None

```
int ledPin = 13;

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

void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```

Understanding Codes: Digital I/O



□ 2. digitalWrite()

digitalWrite()

Description

Write a **HIGH** or a **LOW** value to a digital pin.

NOTE: If you do not set the `pinMode()` to `OUTPUT`, and connect an LED to a pin, when calling `digitalWrite(HIGH)`, the LED may appear dim.

Syntax

`digitalWrite(pin, value)`

Parameters

pin: the pin number

value: **HIGH** or **LOW**

Returns

none

```
int ledPin = 13;

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

void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```

Understanding Codes: Digital I/O

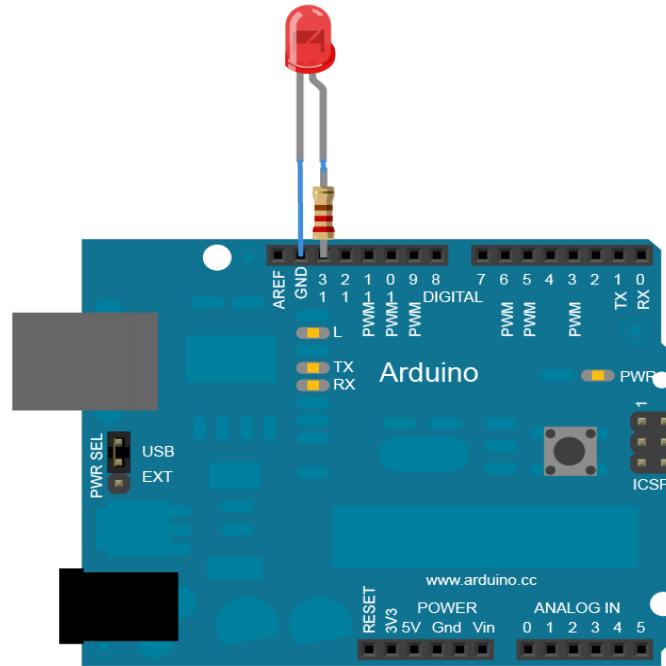


□ 2. digitalWrite()

```
int ledPin = 13;

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

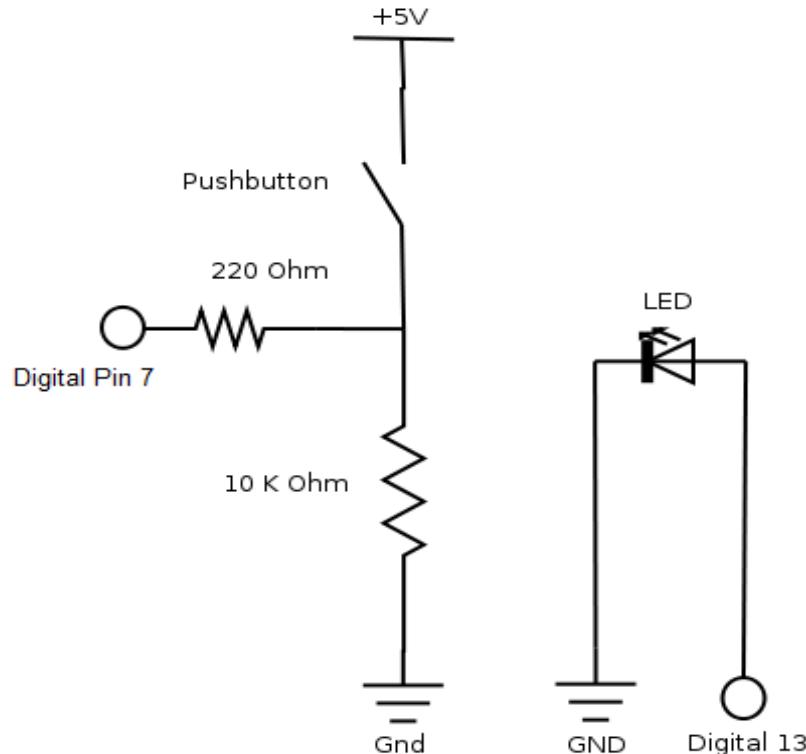
void loop()
{
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```



Understanding Codes: Digital I/O



□ 3. digitalRead()

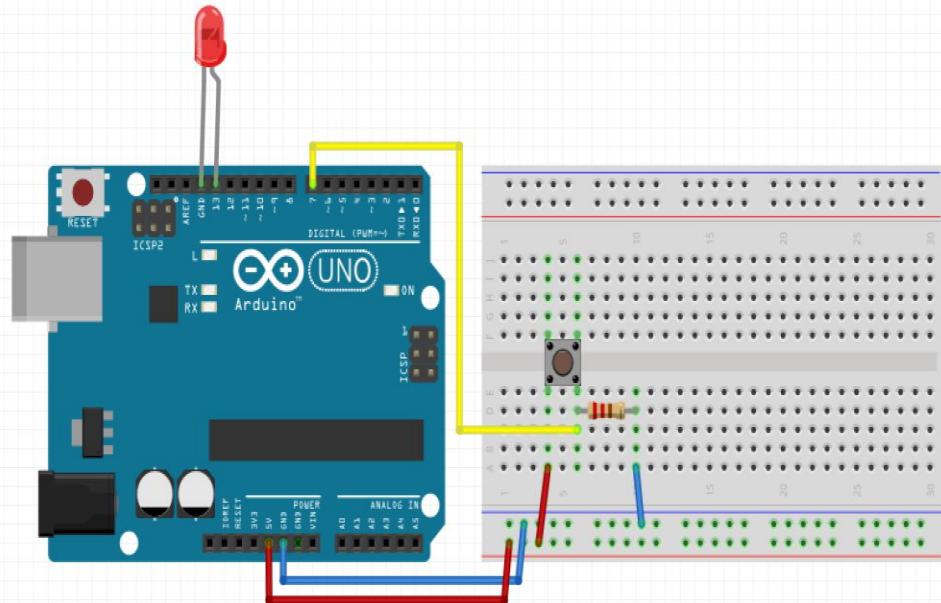


```
int ledPin = 13;
int inPin = 7;
int val = 0;

void setup()
{
    pinMode(ledPin, OUTPUT);
    // sets the digital pin 13 as output
    pinMode(inPin, INPUT);
    // sets the digital pin 7 as input
}

void loop()
{
    val = digitalRead(inPin);
    // read the input pin
    digitalWrite(ledPin, val);
    // sets the LED to the button's value
}
```

Understanding Codes: Digital I/O



```
int ledPin = 13;
int inPin = 7;
int val = 0;

void setup()
{
    pinMode(ledPin, OUTPUT);
    // sets the digital pin 13 as output
    pinMode(inPin, INPUT);
    // sets the digital pin 7 as input
}

void loop()
{
    val = digitalRead(inPin);
    // read the input pin
    digitalWrite(ledPin, val);
    // sets the LED to the button's value
}
```

CSE423: Embedded System

Summer-2020

Serial Communication



The image shows an Arduino Uno R3 microcontroller board. On the left, there's a ATmega328P microcontroller chip. In the center, there's a white infinity symbol logo with a minus sign on the left and a plus sign on the right, followed by the word "ARDUINO" below it. To the right of the logo, the word "UNO" is written inside a white oval with a dotted border. Below the board, a large orange-to-teal diagonal banner contains the text "SERIAL COMMUNICATION IN ARDUINO" in white, bold, sans-serif font. In the bottom left corner of the slide, there is a screenshot of a terminal window showing three lines of text: "Number is 0", "This number is 1", and "This number is 2". In the bottom right corner, there is a watermark for "techZeero".

Todays Lecture



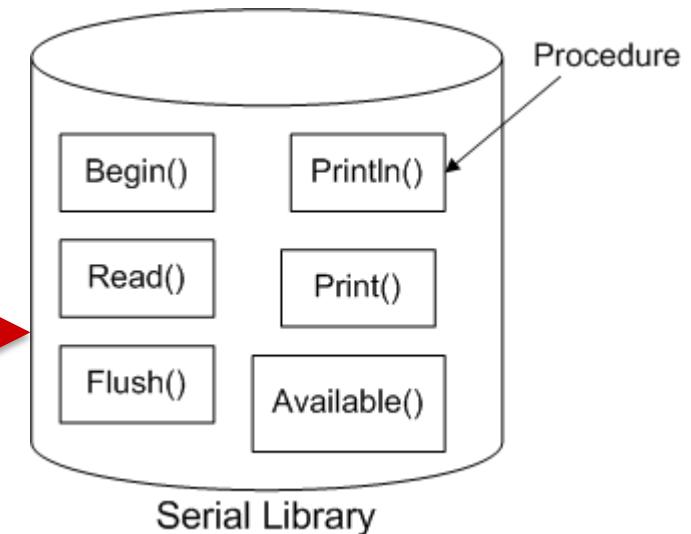
- *Understanding Serial Communication*
- *Understanding Serial Library*
- *Application of Serial Library*
- *Additional functions*

What is Library

A **procedure** is a list of things to do. A **library** is a big collection of procedures, where all the procedures are related!

Let's say, we want to control a motor, you may want to find a Motor Control Library: a collection of procedures that have already been written for you and available to use.

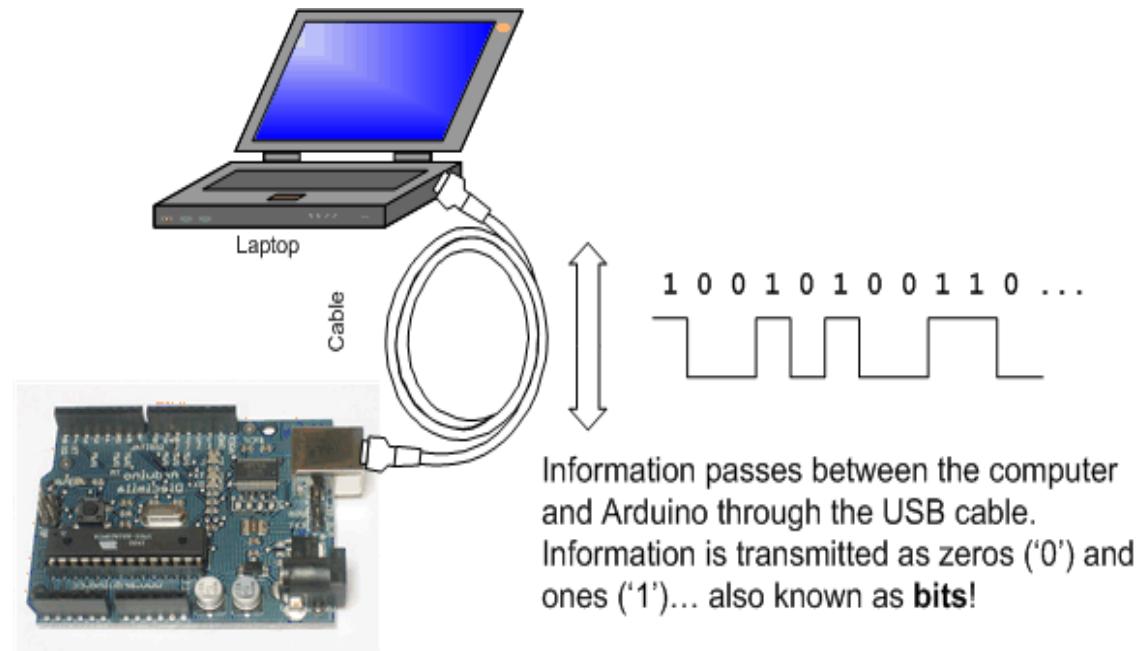
The **library** which is widely used in Arduino platform is the **Serial Library**, which allows the Arduino to send data back to the computer



What is Serial?



The word serial means "**one after the other**". For example, a **serial killer** doesn't stop with one murder, but stabs many people one after the other. **Serial data transfer** is when we transfer data one bit at a time, one right after the other.



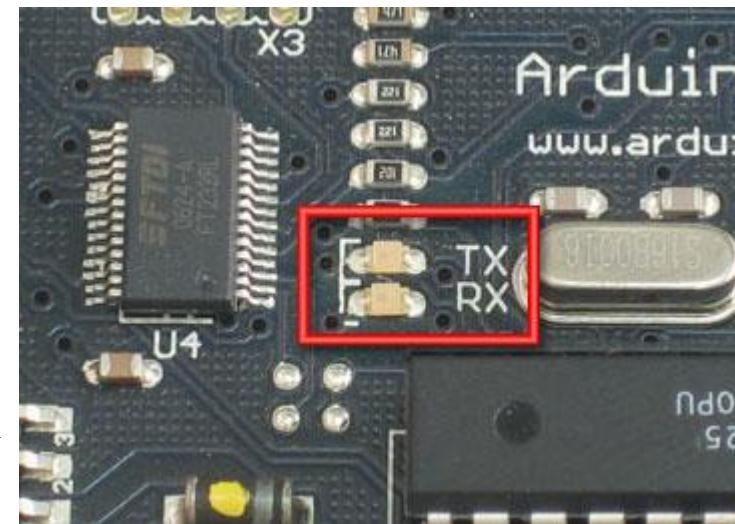
What is Serial?

Information is passed back & forth between the computer and Arduino by, essentially, setting a pin high or low

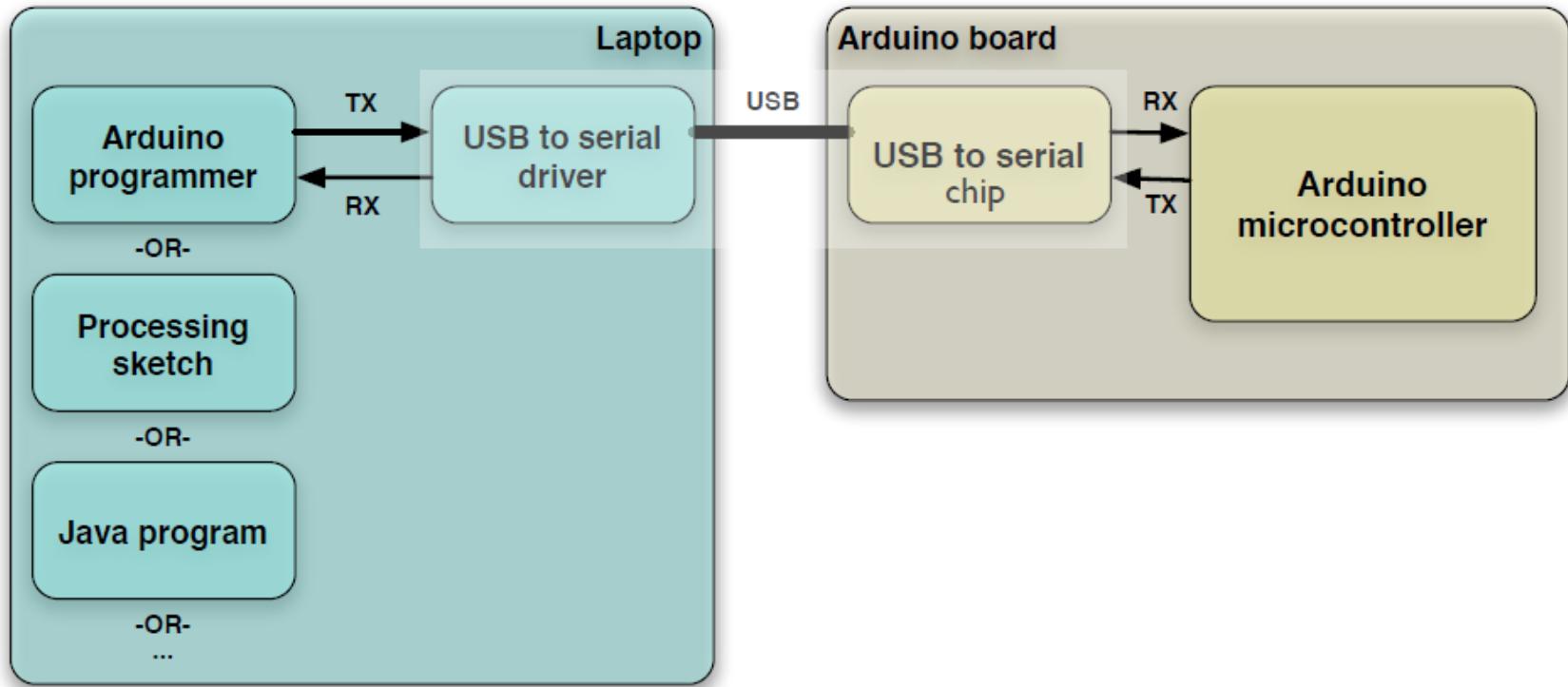
Serial Communication:

We've actually used the Serial communications capability already by sending sketches to the Arduino! When you **Compile/Verify** what you're really doing is turning the sketch into **binary data** (ones and zeros). When you **Upload** it to the Arduino, the bits are shoved out one at a time through the USB cable to the Arduino where they are stored in the main chip.

Next time you upload a sketch, look carefully at the two LEDs near the USB connector, they'll blink when data is being transmitted. One blinks when the Arduino is receiving data (**RX**) and one blinks when the Arduino is transmitting data (**TX**)



Serial communication



What happens if we write this?



```
void setup() // run once, when the sketch starts
{
    Serial.begin(9600); // set up Serial library at 9600 bps
    Serial.println("Hello world!"); // prints hello with ending line break
}
```

```
void loop() // run over and over again
{
    // do nothing!
}
```

Understanding Serial Library



We definitely see that there is a **Serial** thing going on, and it looks like there is a procedure call as well. This is a **library procedure call**. The library is called **Serial** and inside the library is a procedure called **begin**.

library name	.	procedure name	(input values)	;
Serial	.	begin	(9600)	;

If there's no library name, it means that the procedure is in the 'default' collection of procedures we use.

So there's some mystery procedure that's called **begin**, well it's not too tough to figure out what it might do. It's the procedure that gets the Serial stuff ready. But what's the **9600** about? The comment says **9600 bps**, and just so you know bps stands for **bits-per-second** (we will refer to this as the **baud rate**).

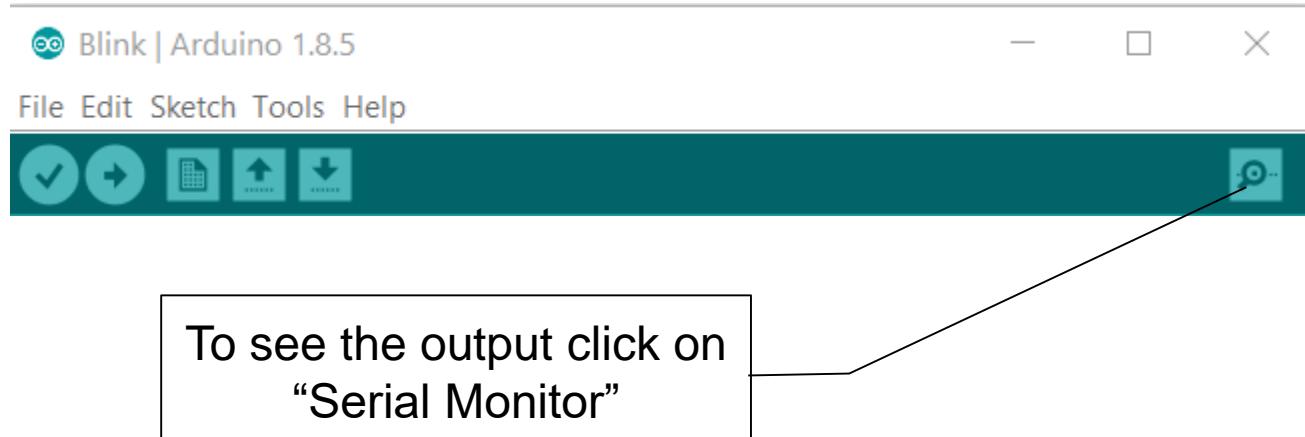
OK so **Serial.begin** sets up the Arduino with the transfer rate we want, in this case **9600 bits per second**

Understanding Serial Library



```
Serial.println("Hello world!");
```

This line also uses the **Serial** library, this time it's calling a procedure called **println** which is just a shorthand for "print line". Note that the 6th letter in **println** is the letter **L** not the number 1. This time the input is a quotation, the line of text we would like it to print. We use two ""'s (double quotes) to indicate the beginning and end of a line of text.



Understanding Serial Library



Output is shown in Separate Window

A screenshot of a Windows-style application window titled "Serial Monitor". The window has a teal header bar with a dropdown menu set to "9600 baud". The main area is a black text box displaying the text "Hello world!" repeated eight times. A white input field is at the top right, and a "Send" button is to its right. A vertical scroll bar is on the right side of the text area. The bottom status bar shows the number "18".

```
Hello world!
```

As serial communication passes information back and forth
so, Arduino Board must be connected to see this output.

Understanding Serial Library



Here, we are addressing couple of most popular libraries related with Serial:

Serial.available()	Serial.flush()	Serial.find()
Serial.write()	Serial.readString()	Serial.read()

Reference Link:

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

CSE423: Embedded System

Summer-2020

Understanding Arduino Code/Sketch (2)



Todays Lecture

- *Understanding Arduino Code (Sketch) step by step*
- *Implementation of code*

Understanding Codes: Analog I/O

□ 1. AnalogRead()

analogRead()

Description

Reads the value from the specified analog pin.

This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023.

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax

analogRead(pin)

Parameters

pin: the number of the analog input pin to read from (0 to 5)

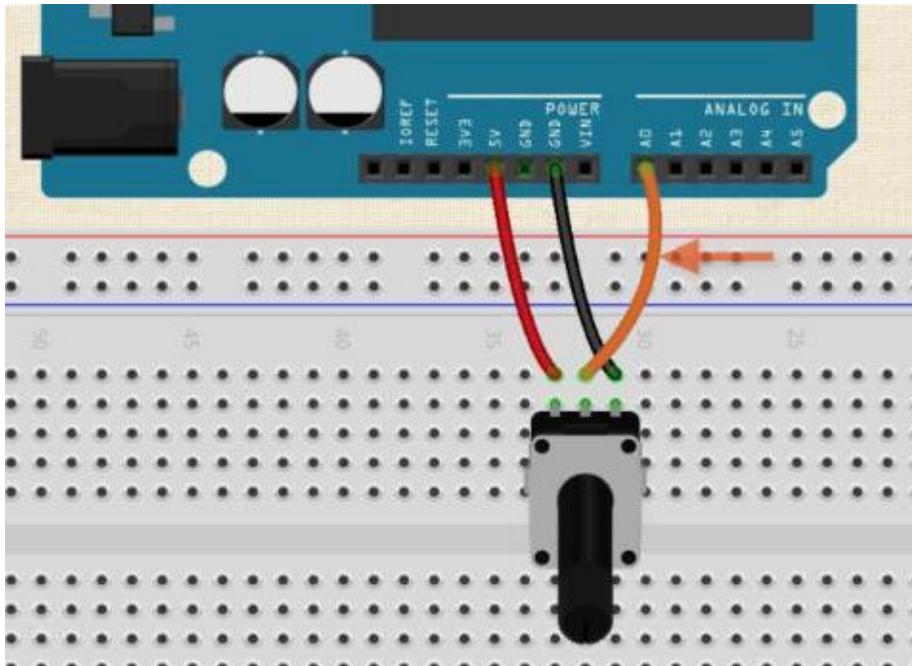
Returns

int (0 to 1023)

```
int analogPin = 0;  
int val = 0;  
  
void setup()  
{  
    Serial.begin(9600);  
    // setup serial  
}  
  
void loop()  
{  
  
    val = analogRead(analogPin);  
    // read the input pin  
    Serial.println(val);  
    // debug value  
}
```

Understanding Codes: Analog I/O

□ 1. AnalogRead()



```
int analogPin = 0;  
  
int val = 0;  
  
void setup()  
{  
    Serial.begin(9600);  
    // setup serial  
}  
  
void loop()  
{  
  
    val = analogRead(analogPin);  
    // read the input pin  
    Serial.println(val);  
    // debug value  
}
```

Understanding Codes: Analog I/O

□ 2. AnalogWrite() -PWM

analogWrite()

Description

Writes an analog value (PWM wave) to a pin.

On most Arduino boards, this function works on pins 3, 5, 6, 9, 10, and 11.

The frequency of the PWM signal on most pins is approximately 490 Hz.

You do not need to call pinMode() to set the pin as an output before calling analogWrite().

Syntax

analogWrite(pin, value)

Parameters

pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on).

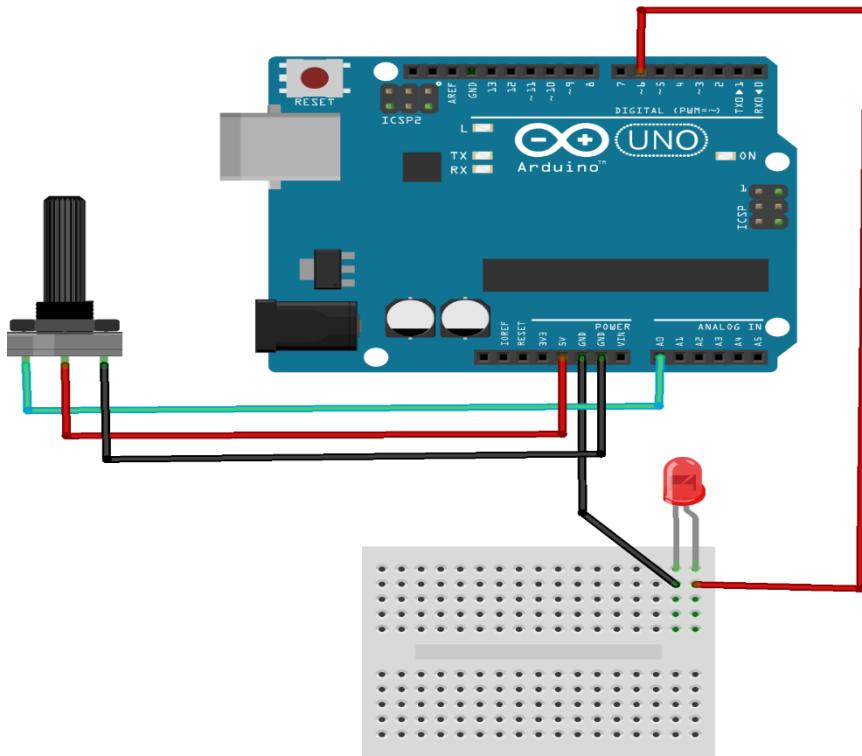
Returns

nothing

```
int ledPin = 6;  
  
int analogPin = 0;  
  
int val = 0;  
  
void setup()  
{  
    pinMode(ledPin, OUTPUT);  
}  
  
void loop()  
{  
  
    val = analogRead(analogPin);  
    // read the input pin  
    analogWrite(ledPin, val / 4);  
    // analogRead values go from 0 to 1023,  
    // analogWrite values from 0 to 255  
}
```

Understanding Codes: Analog I/O

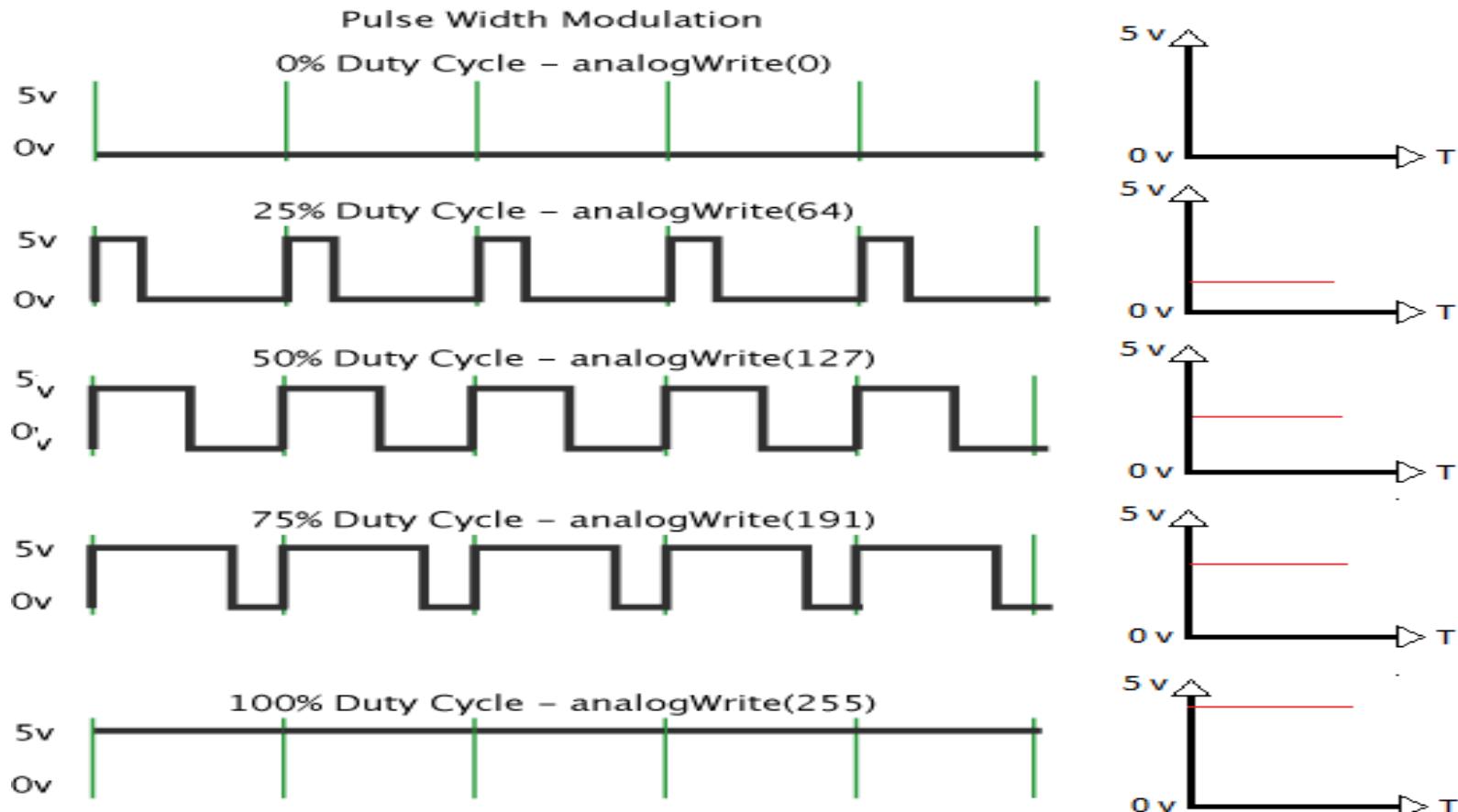
□ 2. AnalogWrite() -PWM



```
int ledPin = 6;  
  
int analogPin = 0;  
  
int val = 0;  
  
void setup()  
{  
    pinMode(ledPin, OUTPUT);  
}  
  
void loop()  
{  
  
    val = analogRead(analogPin);  
    // read the input pin  
    analogWrite(ledPin, val / 4);  
    // analogRead values go from 0 to 1023,  
    // analogWrite values from 0 to 255  
}
```

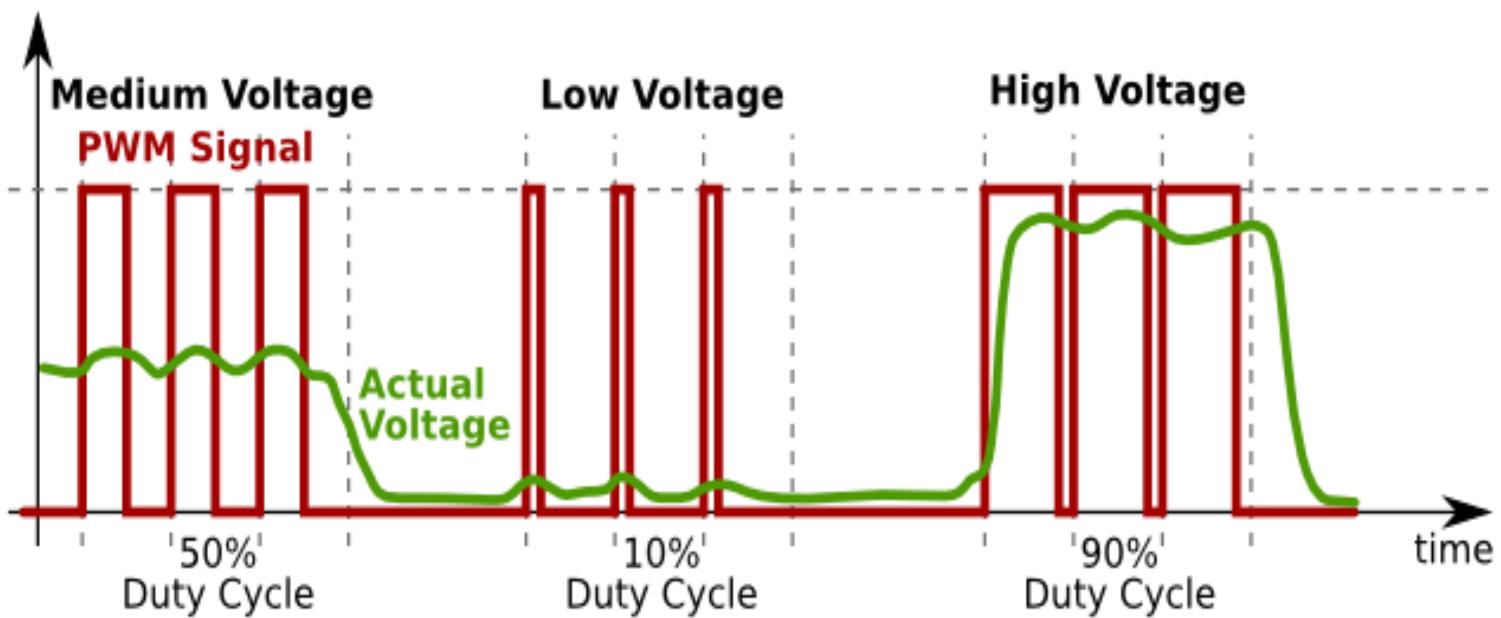
Understanding Codes: Analog I/O

□ 2. AnalogWrite() - PWM



Understanding Codes: Analog I/O

□ 2. AnalogWrite() -PWM



Understanding Codes: Functions

MATH	MATH	MATH	Timing
<code>min()</code>	<code>constrain()</code>	<code>sqrt()</code>	<code>millis()</code>
<code>max()</code>	<code>map()</code>	<code>random()</code>	<code>micros()</code>
<code>abs()</code>	<code>pow()</code>	<code>randomSeed()</code>	<code>delay()</code>
<code>sin()</code>	<code>cos()</code>	<code>tan()</code>	<code>delayMicroseconds()</code>

Understanding Codes: Functions

□ 1. min()

min(x, y)

Description

Calculates the minimum of two numbers.

```
sensVal = min(sensVal, 100);
```

Parameters

x: the first number, any data type

y: the second number, any data type

Returns

The smaller of the two numbers.

```
min(a++, 100);    // avoid this  
  
min(a, 100);  
a++;             // use this .
```

Understanding Codes: Functions

□ 2. max()

max(x, y)

Description

Calculates the maximum of two numbers.

```
sensVal = max(senVal, 20);
```

Parameters

x: the first number, any data type

```
max(a-- , 0); // avoid this  
  
max(a , 0);  
a-- ; //use this
```

y: the second number, any data type

Returns

The larger of the two parameter values.

Understanding Codes: Functions

□ 3. constrain()

constrain(x, a, b)

Description

Constrains a number to be within a range.

Parameters

x: the number to constrain, all data types

a: the lower end of the range, all data types

b: the upper end of the range, all data types

```
sensVal = constrain(sensVal, 10, 150);  
// limits range of sensor values to between 10 and 150
```

Returns

x: if x is between a and b

a: if x is less than a

b: if x is greater than b

Understanding Codes: Functions

□ 4. abs()

abs(x)

Description

Computes the absolute value of a number.

Parameters

x: the number

Returns

x: if x is greater than or equal to 0.

-x: if x is less than 0.

```
abs(a++);    // avoid this  
  
abs(a);  
a++;          // use this
```

Understanding Codes: Functions

□ 5. map()

map(value, fromLow, fromHigh, toLow, toHigh)

Description

Re-maps a number from one range to another. That is, a **value** of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

```
y = map(x, 1, 50, 50, 1);
```

```
y = map(x, 1, 50, 50, -100);
```

```
/* Map an analog value to 8 bits (0 to 255) */
void setup() {}

void loop()
{
    int val = analogRead(0);
    val = map(val, 0, 1023, 0, 255);
    analogWrite(9, val);
}
```

Understanding Codes: Functions

□ 6. `pow()`

`pow(base, exponent)`

Description

Calculates the value of a number raised to a power. Pow() can be used to raise a number to a fractional power. This is useful for generating exponential mapping of values or curves.

Parameters

base: the number (*float*)

exponent: the power to which the base is raised (*float*)

Returns

The result of the exponentiation (*double*)

Understanding Codes: Functions

□ 7. `sin()`

`sin(rad)`

Description

Calculates the sine of an angle (in radians). The result will be between -1 and 1.

Parameters

`rad`: the angle in radians (*float*)

Returns

the sine of the angle (*double*)

Understanding Codes: Functions

□ 8. random()

random()

Description

The random function generates pseudo-random numbers.

Syntax

random(max)

random(min, max)

Parameters

min - lower bound of the random value, inclusive (*optional*)

max - upper bound of the random value, exclusive

Returns

a random number between min and max-1 (*long*)

```
long randNumber;  
  
void setup(){  
    Serial.begin(9600);  
}  
  
void loop(){  
    randNumber = random(300);  
    Serial.println(randNumber);  
  
    delay(50);  
}
```

Understanding Codes: Functions

□ 9. millis()

millis()

Description

Returns the number of milliseconds since the Arduino board began running the current program.
This number will overflow (go back to zero), after approximately 50 days.

Parameters

None

Returns

Number of milliseconds since
the program started (*unsigned long*)

```
unsigned long time;

void setup(){
    Serial.begin(9600);
}
void loop(){
    Serial.print("Time: ");
    time = millis();
    //prints time since program started
    Serial.println(time);
    // wait a second so as not to send massive amounts of data
    delay(1000);
}
```

Understanding Codes: Functions

□ 10. delay()

delay()

Description

Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax

delay(ms)

Parameters

ms: the number of milliseconds to pause (*unsigned long*)

Returns

nothing

```
int ledPin = 13;           // LED connected to digital pin 13

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

void loop()
{
    digitalWrite(ledPin, HIGH); // sets the LED on
    delay(1000);             // waits for a second
    digitalWrite(ledPin, LOW); // sets the LED off
    delay(1000);             // waits for a second
}
```

Understanding Codes: Functions

□ 11. delayMicroseconds()

delayMicroseconds()

Description

Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.

Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

Syntax

delayMicroseconds(us)

Parameters

us: the number of microseconds
to pause (*unsigned int*)

Returns

None

```
int outPin = 8;                      // digital pin 8

void setup()
{
    pinMode(outPin, OUTPUT);          // sets the digital pin as output
}

void loop()
{
    digitalWrite(outPin, HIGH);       // sets the pin on
    delayMicroseconds(50);           // pauses for 50 microseconds
    digitalWrite(outPin, LOW);        // sets the pin off
    delayMicroseconds(50);           // pauses for 50 microseconds
}
```

Understanding Codes: **Serial.xxx**

(Serial)	flush()	print()
available()	parseFloat()	println()
begin()	parseInt()	write()
end()	peek()	read()
find()	setTimeout()	readBytes()
findUntil()	serialEvent()	readBytesUntil()

CSE423: Embedded System

Summer-2020

Introduction to TinkerCAD (Online Simulator-1)



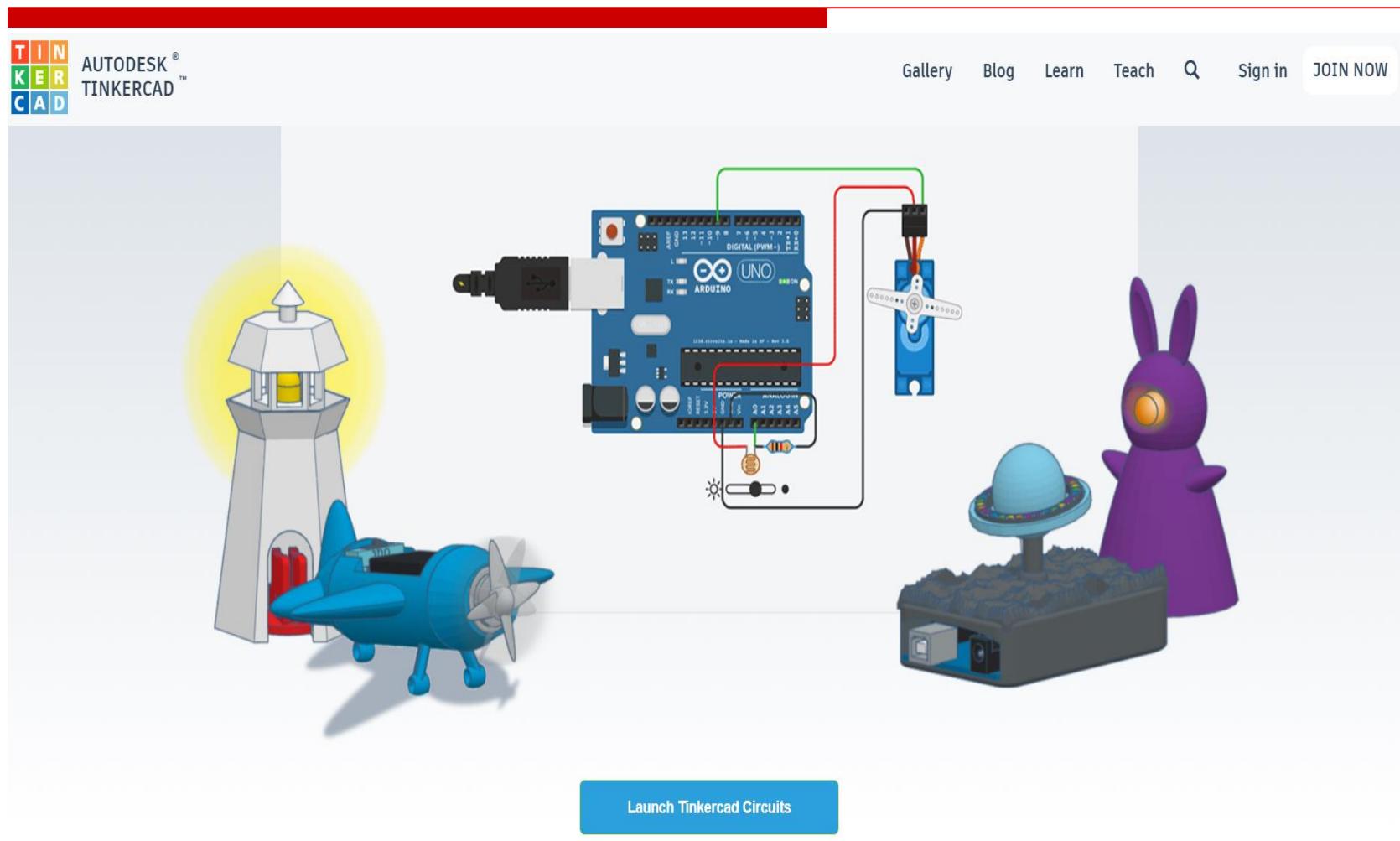
Todays Lecture



- *Introduction to TinkerCAD*
- *How to use TinkerCAD*
- *Tutorial available at:*

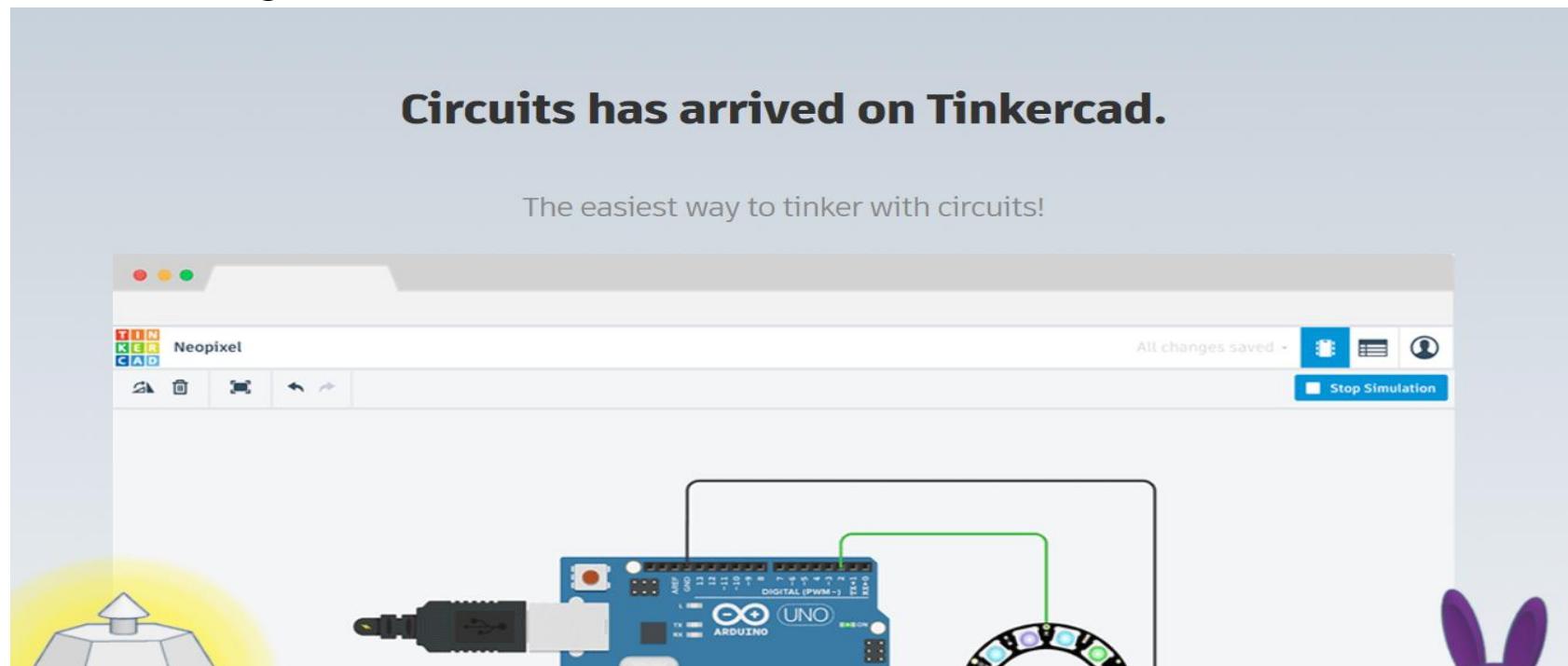
<https://youtu.be/yyG0koj9nNY>

TinkerCAD Website <https://www.tinkercad.com/circuits>



What is TinkerCAD?

TinkerCAD is a free online service for creating basic 3D shapes and developing digital prototypes of electronic components. These prototypes include basic circuits with LED lights, buzzers, switches, and even light sensors.



Why TinkerCAD Circuits ?



TinkerCad offers many benefits :

- 1- Online : You do not need to install anything in your PC.
- 2- OpenSoure : Free, no license needed, for everyone.

Creating an Account



The screenshot shows the "Join | Tinkercad" page at <https://www.tinkercad.com/join>. The main heading is "Start Tinkering" with the sub-question "How will you use Tinkercad?". There are three main buttons: a blue one for "Educators start here", a green one for "Students, join a Class", and a blue one for "Create a personal account". Below these, there's a link for users who "Already have an account?" with a "Sign In" button.

Start Tinkering
How will you use Tinkercad?

In school?

Educators start here

Students, join a Class

On your own

Create a personal account

Already have an account?
[Sign In](#)

TinkerCAD Circuits



- Click on Circuits to switch from **3D Designs** to **Circuits** mode, and then click on **Create new Circuit**. If everything is all right let's start working and discover this platform.

A screenshot of the TinkerCAD Circuits interface. At the top, there is a navigation bar with the TinkerCAD logo, a search bar, and a user profile icon. The main area is titled "Circuits" and features a "Create new Circuit" button. On the left, a sidebar shows the user profile "ahmnouira", a search bar, and categories: "3D Designs" (selected), "Circuits" (highlighted in blue), "Lessons", "Projects" (with "Project 2" and "Project 1"), and a "Create project" button. The main content area displays four circuit projects: "Smooth Lappi" (20 days ago, Private), "Funky Blorr" (2 months ago, Private), "Super Migelo" (2 months ago, Private), and "Funky Bojo" (2 months ago, Private). Each project is shown with its name, upload date, privacy status, and a small thumbnail image.

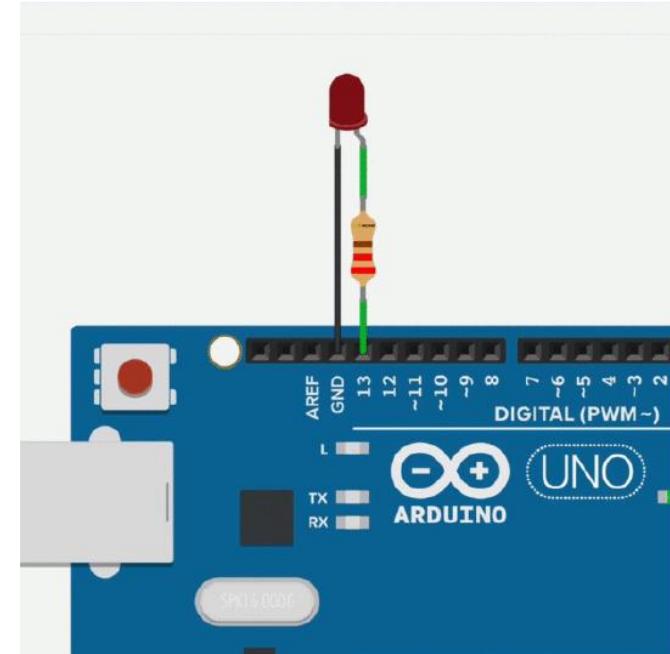
Let's Blink a LED in TinkerCAD!



□ Step 1: LED Resistor Circuit

The LED's legs are connected to two pins on the Arduino: **ground and pin 13**. The component between the LED and pin 13 is a resistor, which helps limit the current to prevent the LED from burning itself out.

The colored stripes identify the resistor's value, and for this circuit, anywhere from 100 ohms to 1000 ohms will work great.



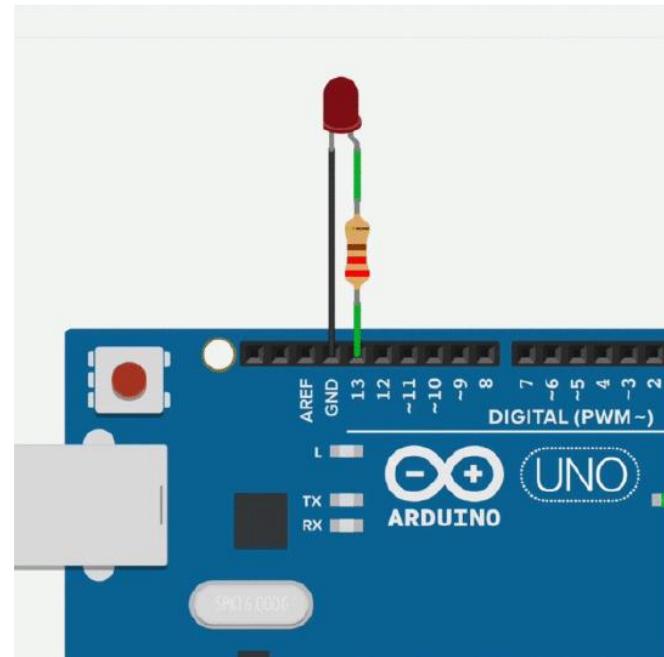
The LED, on the other hand, is polarized, which means it only works when the legs are connected a certain way. The positive leg, called the anode, usually has a longer leg, and gets wired to power, in this case coming from your Arduino's output pin. The negative leg, called the cathode, with its shorter leg, connects to ground.

Let's Blink a LED in TinkerCAD!



□ Step 1: LED Resistor Circuit

- In the Tinkercad Circuits components panel, drag a resistor and LED onto the workplane.
- Edit the resistor's value by adjusting it to 220 ohms in the component inspector which appears when the resistor is selected.
- Back in the components panel, find and bring over an Arduino Uno board. Click once to connect a wire to a component or pin, and click again to connect the other end.
- Connect your resistor to either side of the LED. If you connected your resistor to the LED's anode (positive, longer), connect the resistor's other leg to Arduino's digital pin 13.
- If you connected your resistor to the LED's cathode (negative, shorter leg), connect the resistor's other leg to Arduino's ground pin (GND).
- Create another wire between the unconnected LED leg and pin 13 or ground, whichever is still not connected.



Let's Blink a LED in TinkerCAD!



□ Step 2: Simple Code With Blocks

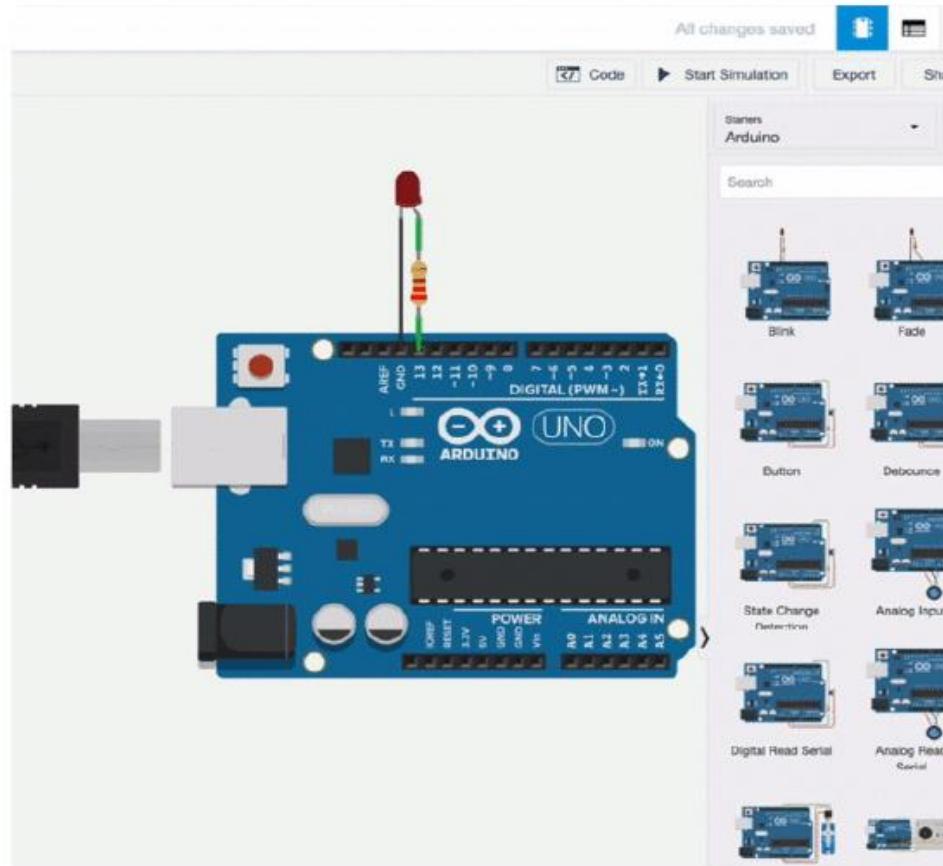
- Let's go through the simple code controlling the **blink** by opening the code editor (button labeled "**Code**"). You can resize the code editor by clicking and dragging the left edge.
- The code starts out with **two gray** comment blocks, which are just notes for us humans to read. The first blue output block sets the built-in LED HIGH, which is Arduino's way of describing "on."
- This output command will activate a 5V signal to anything connected to the specified pin. Next up is a **yellow** command block that waits for one second, simple enough.



Let's Blink a LED in TinkerCAD!



□ Step 2: Simple Code With Blocks



Let's Blink a LED in TinkerCAD!



□ Step 2: Simple Code With Blocks (Complete Code)

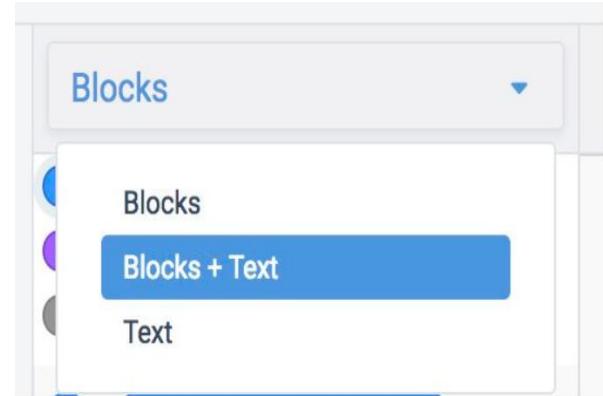
```
/* This program blinks pin 13 of the Arduino (the built-in LED) */  
void setup()  
{  
    pinMode(13, OUTPUT);  
}  
void loop()  
{  
    // turn the LED on (HIGH is the voltage level)  
    digitalWrite(13, HIGH);  
    delay(1000); // Wait for 1000 millisecond(s)  
    // turn the LED off by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

Let's Blink a LED in TinkerCAD!



□ Step 3: Blink Arduino Code Explained

When the code editor is open, you can click the dropdown menu on the left and select "Blocks + Text" to reveal the Arduino code generated by the code blocks. All the extra symbols are part of Arduino's syntax, but don't be intimidated! It takes time to learn to write proper code from scratch. We'll go through each piece here, and you can always use the blocks for comparison as you level up.



This first section is title block comment, describing what the program does. Block comments are bookended by an opening /* and closing */.

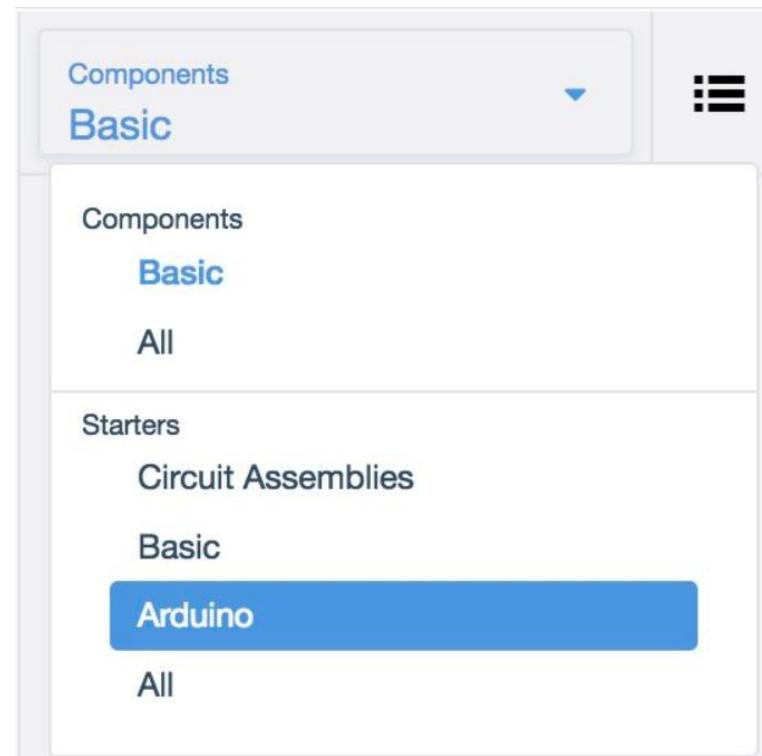
Next is the code's setup, which helps set up things your program will need later. It runs once when the program starts up, and contains everything within its curly braces { }. Our blink sketch's setup configures pin 13 as an output, which prepares the board to send signals to it, rather than listen.

Let's Blink a LED in TinkerCAD!



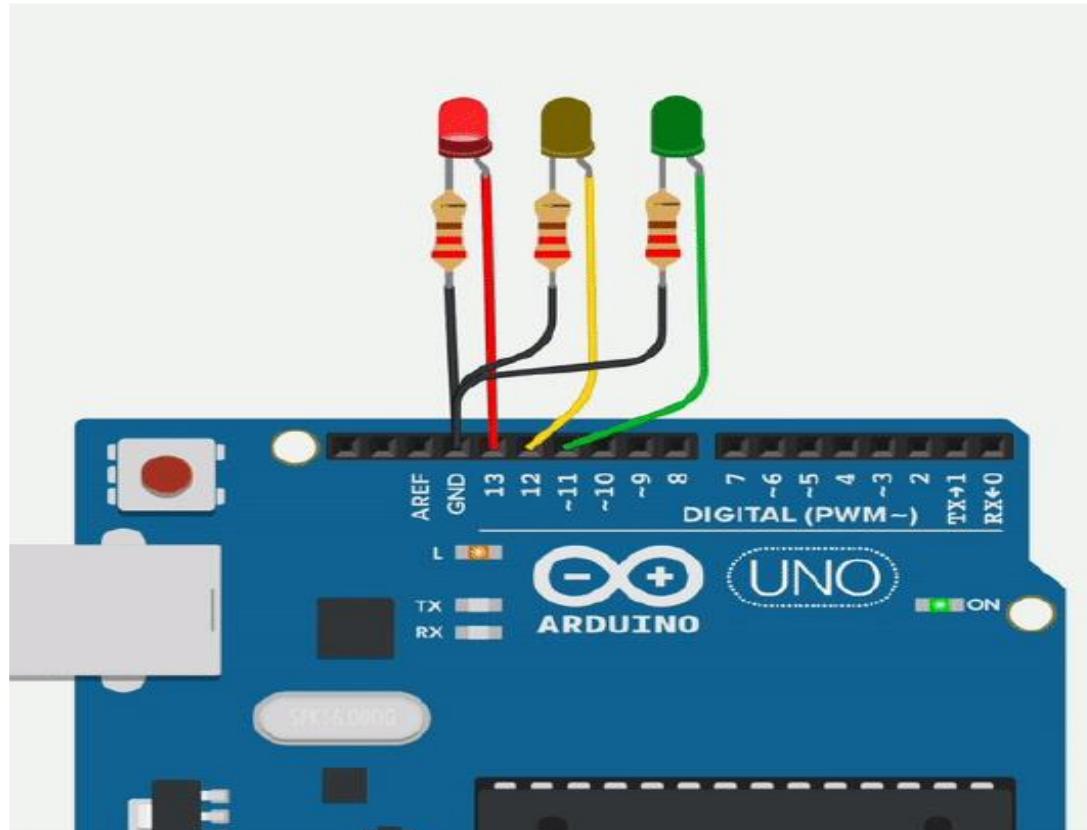
□ Step 4: Use the Blink Circuit Starter

Grab this circuit and code combo any time using the starter available in the components panel ([dropdown menu -> Starters -> Arduino](#)).



Task

- Try the same LED Blink for Multiple LEDs



CSE423: Embedded System

Summer-2020

Simulation with TinkerCAD (Online Simulator-2)



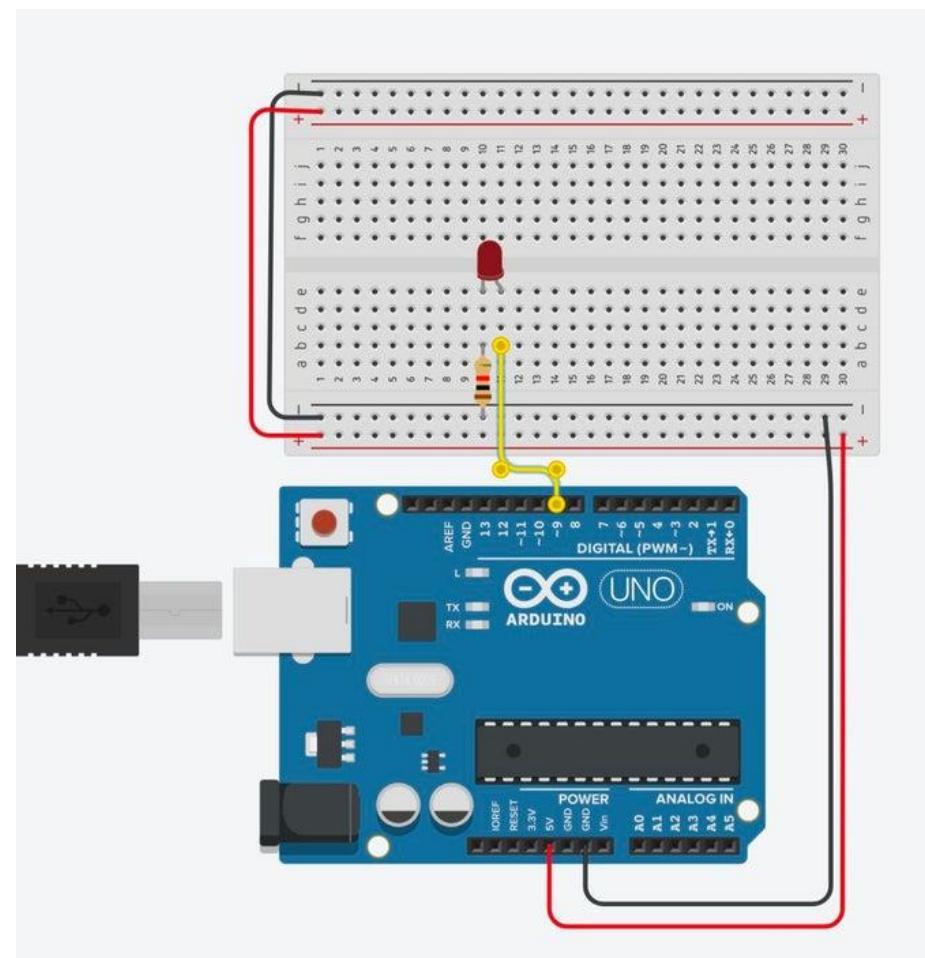
Todays Lecture



- *Understanding TinkerCAD in detail*
- *LED dimmer using PWM supported pin*
- *Tutorial available at:*
<https://youtu.be/X8dHbdhnGKY>

Simulate a LED Dimmer step by step

Step-1: Connect a LED Circuit like the regular one in the breadboard.



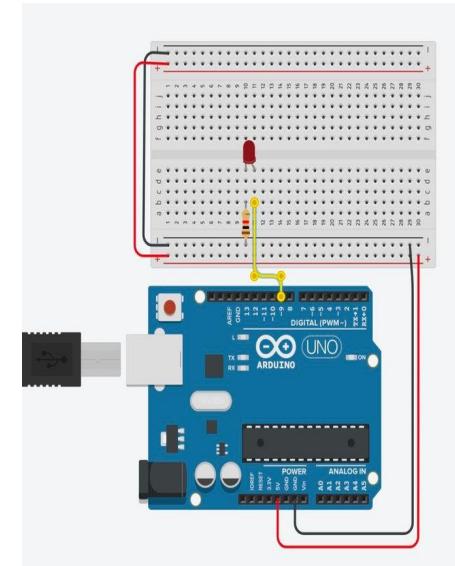
Simulate a LED Dimmer step by step



- The LED is connected in series with a resistor between Arduino pin 9 and ground.

Follow the steps for connections:

- Breadboard power (+) and ground (-) rails to Arduino 5V and ground(GND)respectively.
- LED cathode (negative, shorter leg) to one leg of a resistor (anywhere from 100-1K ohms is fine)
- Other resistor leg to ground LED anode (positive, longer leg) to Arduino pin 9



Simulate a LED Dimmer step by step



□ Step 2: Build Brightness Adjustment Code With Blocks

[Click Code Editor to open the code blocks editor](#)

- Start with a control block that counts. Set it to count up by five. Click the dropdown next to "for" and select "rename variable...", then rename it to "brightness". Adjust the "from" and "to" values to 0 and 255, respectively.
- Inside the counting loop, add an output block to set one of the special pins, and adjust it to pin 9. Navigate to Variables and drag the brightness block to the output block to set pin 9 to the current value of brightness, which changes over the course of the counting loop.
- Add a wait block, and set it to 30 milliseconds. This gives time for the light to shine at each brightness level so you have time to see it before it changes. The duration of this block can be changed to slow down or speed up the fading effect.

Simulate a LED Dimmer step by step

A screenshot of a Scratch script editor. On the left, there's a palette with categories: Output (blue), Input (purple), Control (yellow), Math (orange), and Variables (pink). Below the palette are several script blocks:

- wait (1 secs)
- repeat (10 times)
 - repeat (while)
 - if then
 - if then
 - else

A script block for a sprite is selected, showing the following code:

```
when green flag clicked
  repeat (10 times)
    [set brightness to (0 v) for (5 steps)
      go to x: (0 v) y: (0 v)
      end]
  end
end
```

A context menu is open over the selected script block, displaying three options: "Duplicate", "Delete 4 Blocks", and "Help".

Output
Input
Notation
Control
Math
Variables

when green flag clicked

repeat (10 times)

set brightness to (0 v) for (5 steps)

go to x: (0 v) y: (0 v)

end

end

Duplicate

Delete 4 Blocks

Help

Simulate a LED Dimmer step by step



□ Step 3: Brightness Adjustment Arduino Code Explained

The counting loop you created fades the LED from off to all the way on. To fade the LED back off again, we have to create another counting loop. Either drag a new counting loop into the editor, or duplicate this one, and this time change it to count down, start with 255, and go down to zero.

Simulate a LED Dimmer step by step



title block comment Fade \nThis example shows how to fade an LED o...



The Scratch script consists of two main loops. The first loop, triggered by a green flag, sets pin 9 to brightness, waits 30 milliseconds, then counts up from 0 to 255, setting pin 9 to the current brightness value and waiting 30 milliseconds between each step. The second loop, triggered by a green flag, sets pin 9 to brightness, waits 30 milliseconds, then counts down from 255 to 0, setting pin 9 to the current brightness value and waiting 30 milliseconds between each step.

```
1  /*
2   * Fade
3   * This example shows how to fade an LED on pin 9
4   * using the analogWrite() function.
5   *
6   * The analogWrite() function uses PWM, so if you
7   * want to change the pin you're using, be sure to
8   * use another PWM capable pin. On most Arduino,
9   * the PWM pins are identified with a "-" sign,
10  * like -3, -5, -6, -9, -10 and -11.
11  */
12
13 int brightness = 0;
14
15 void setup()
16 {
17     pinMode(9, OUTPUT);
18 }
19
20 void loop()
21 {
22     for (brightness = 0; brightness <= 255; brightness++)
23         analogWrite(9, brightness);
24         delay(30); // Wait for 30 millisecond(s)
25     }
26     for (brightness = 255; brightness >= 0; brightness--)
27         analogWrite(9, brightness);
28         delay(30); // Wait for 30 millisecond(s)
29     }
30 }
```

Simulate a LED Dimmer step by step



```
/* Fade
```

This example shows how to fade an LED on pin 9 using the `analogWrite()` function. The `analogWrite()` function uses PWM, so if you want to change the pin you're using, be sure to use another PWM capable pin. On most Arduino, the PWM pins are identified with a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11. */

```
int brightness = 0;

void setup()
{
    pinMode(9, OUTPUT);
}

void loop()
{
    for (brightness = 0; brightness <= 255;
        brightness += 5)
    {
        analogWrite(9, brightness);
        delay(30);      // Wait for 30 millisecond(s)
    }
}
```

Simulate a LED Dimmer step by step



- The program's loop uses two for loops to count up from 0 to 255 by increments of 5. The analogWrite() function takes two arguments: the Arduino pin number (9 in our case), and a value between 0 (off) and 255 (all the way on).

Simulate a LED Dimmer step by step

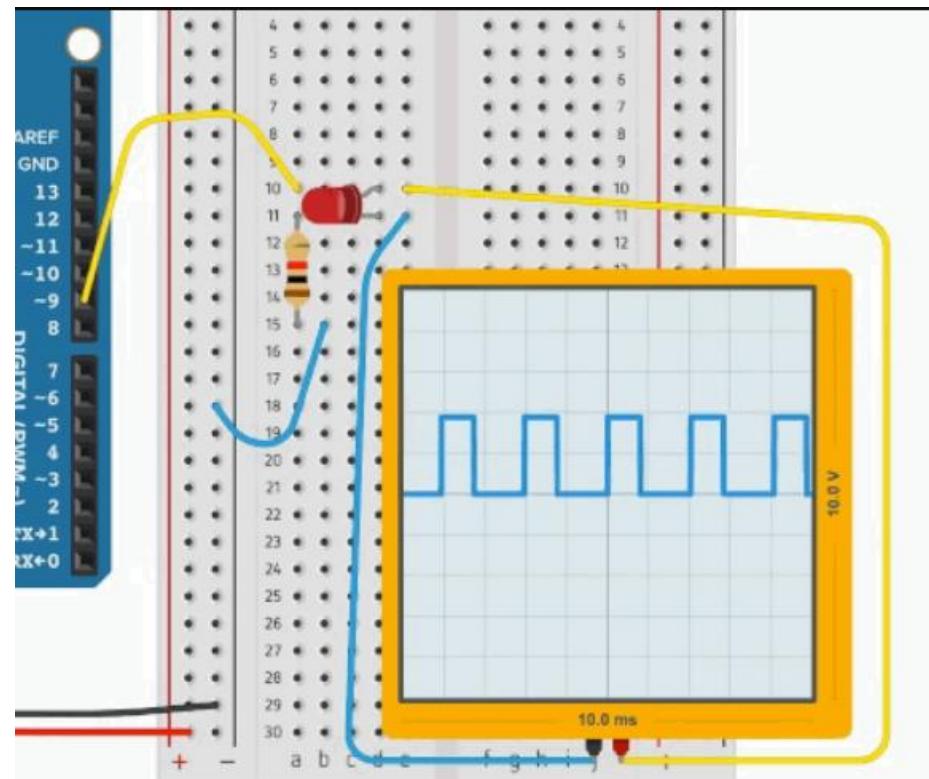


- This circuit is also available as a **fade circuit starter**. You can use this circuit starter anytime you want to fade an LED.
- Grab this circuit and code combo any time using the starter available in the components panel (**dropdown menu -> Starters -> Arduino**).
- Scroll to find the Arduino starter labeled **Fade**, and double click it to add it to the workplane (you can also click and drag instead).

- Notice how the resistor in this version is "upstream" of the LED, connected between power and the LED instead of the LED and ground. These two circuits both make the same connections, linking up the LED to a signal pin and ground, through a current limiting resistor, which will function on either side of the LED.

Simulate a LED Dimmer step by step

- Task: Check it with Pulse Width Modulation (PWM)



Simulate a LED Dimmer step by step

Pulse Width Modulation

The Arduino board is only capable of generating digital signals (HIGH and LOW), but `analogWrite()`; simulates the appearance of brightnesses between on and off using pulse width modulation (PWM). The LED flashes on and off very quickly, and your eye interprets a dimmer light. The ratio of time the LED spends on vs. off determines how bright or dim the LED appears.

Pulse width modulation (PWM) creates an oscillating digital signal, alternately driven high and low in a repeating pattern. Each high to low to high period of time is called a cycle.

You can see an oscillating digital signal on the oscilloscope like the animation above.

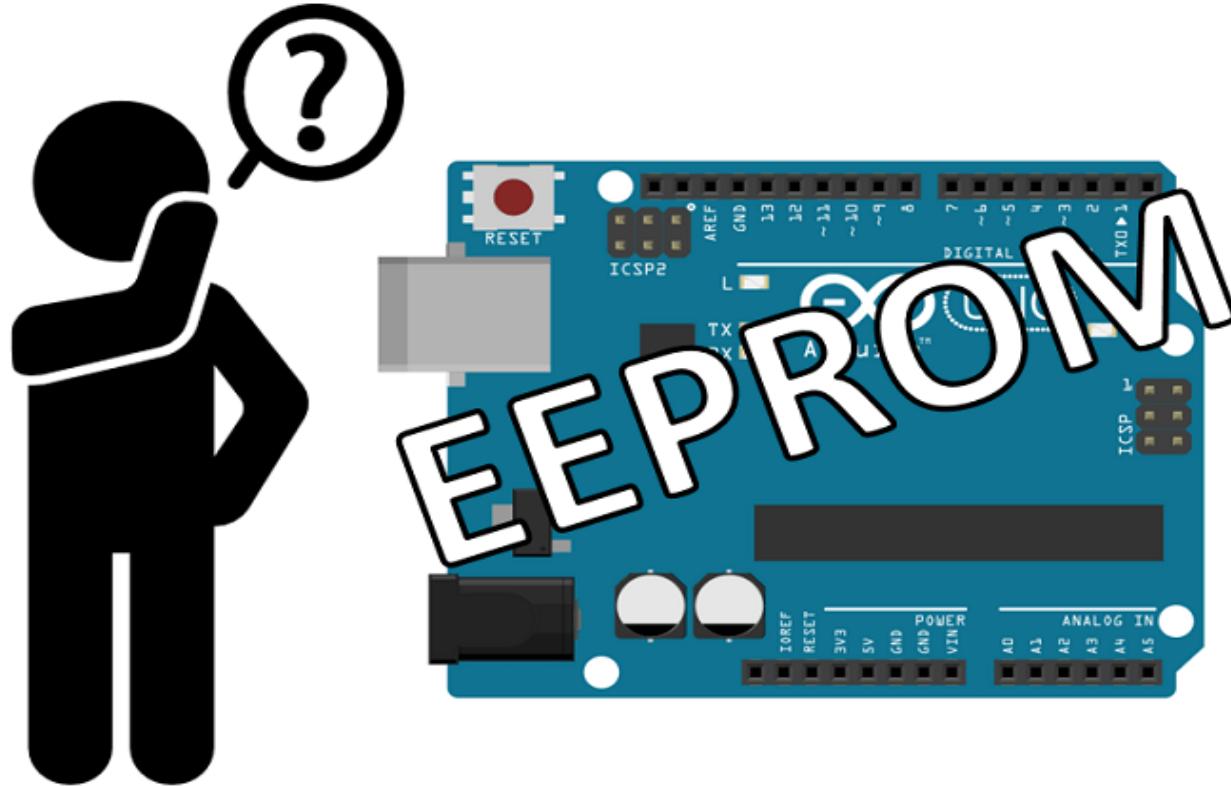
Notice that for each cycle, the width of the HIGH and LOW portions of the graph are changing, hence the term pulse width modulation, or PWM for short.

Identify the other digital pins on the Arduino Uno capable of PWM, marked with a ~: 3, 5, 6, 9, 10, and 11.

CSE423: Embedded System

Summer-2020

Storing data in EEPROM



Todays Lecture



- *Understanding EEPROM*
- *When EEPROM is applicable?*
- *Data size*
- *How to write on EEPROM*
- *How to read from EEPROM*

Storing data in EEPROM



An EEPROM is an **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory.

It is a form of non-volatile memory that can remember things with the power being turned off, or after resetting the Arduino. The beauty of this kind of memory is that we can store data generated within a sketch on a more permanent basis.

Why EEPROM to store data?



For situations where data that is unique to a situation needs a more permanent home.

- Storing unique serial number /date for commercial purpose (a function of the sketch could display the serial number on an LCD).
- You may need to count certain events and not allow the user to reset them – such as an odometer or operation cycle-counter.

Storing data in EEPROM



What sort of data can be stored?

Anything that can be represented as *bytes* of data.

Now each digit in that binary number uses one ‘bit’ of memory, and eight bits make a byte. Due to internal limitations of the microcontrollers in our Arduino boards, we can only store 8-bit numbers (one byte) in the EEPROM.

This limits the decimal value of the number to fall between zero and 255.

How to Store data in EEPROM



Step-1: To use the EEPROM, a library is required, so use the following library in your sketches:

```
#include "EEPROM.h"
```

The rest is very simple. To store a piece of data, we use the following function:

```
EEPROM.write(a,b);
```

The parameter a is the position in the EEPROM to store the integer (0~255) of data b. In this example, we have 1024 bytes of memory storage, so the value of a is between 0 and 1023.

How to Store data in EEPROM



Step-2: To retrieve a piece of data is equally as simple, use:

```
z = EEPROM.read(a);
```

Where z is an integer to store the data from the EEPROM position a .

Sample **Example** to store data in EEPROM



This sketch will create random numbers between 0 and 255, store them in the EEPROM, then retrieve and display them on the serial monitor.

Sample Example to store data in EEPROM



```
// Arduino internal EEPROM demonstration

#include <EEPROM.h>
int zz;
int EEsize = 1024; // size in bytes of your board's EEPROM

void setup()
{
    Serial.begin(9600);
    randomSeed(analogRead(0));
}

void loop()
{
    Serial.println("Writing random numbers..."); }
    for (int i = 0; i < EEsize; i++)
    {
        zz=random(255);
        EEPROM.write(i, zz);
    }
    Serial.println();
    for (int a=0; a<EEsize; a++)
    {
        zz = EEPROM.read(a);
        Serial.print("EEPROM position: ");
        Serial.print(a);
        Serial.print(" contains ");
        Serial.println(zz);
        delay(25);
    }
}
```

Sample Example to store data in EEPROM



Output from Serial Monitor

```
COM11  
Send  
Writing random numbers...  
  
EEPROM position: 0 contains 136  
EEPROM position: 1 contains 219  
EEPROM position: 2 contains 50  
EEPROM position: 3 contains 232  
EEPROM position: 4 contains 216  
EEPROM position: 5 contains 147  
EEPROM position: 6 contains 241  
EEPROM position: 7 contains 31  
EEPROM position: 8 contains 31  
EEPROM position: 9 contains 162  
EEPROM position: 10 contains 222  
EEPROM position: 11 contains 43  
EEPROM position: 12 contains 128  
EEPROM position: 13 contains 102  
EEPROM position: 14 contains 64  
EEPROM position: 15 contains 122  
EEPROM position: 16 contains 46  
EEPROM position: 17 contains 61  
EEPROM position: 18 contains 52  
EEPROM position: 19 contains 152  
EEPROM position: 20 contains 179  
EEPROM position: 21 contains 101  
EEPROM position: 22 contains 247  
EEPROM position: 23 contains 195  
EEPROM position: 24 contains 166  
EEPROM position: 25 contains 206  
EEPROM position: 26 contains 252  
EEPROM position: 27 contains 110  
EEPROM position: 28 contains 43  
EEPROM position: 29 contains 130  
EEPROM position: 30 contains 105  
EEPROM position: 31 contains 128  
EEPROM position: 32 contains 158  
EEPROM position: 33 contains 20  
EEPROM position: 34 contains 96  
EEPROM position: 35 contains 157  
EEPROM position: 36 contains 215  
EEPROM position: 37 contains 118  
EEPROM position: 38 contains 91  
EEPROM position: 39 contains 243  
EEPROM position: 40 contains 69  
  
Autoscroll No line ending 9600 baud
```

Storing data in EEPROM



- What about replacing an existing data with a new one?

The `EEPROM.update()` function is particularly useful. It only writes on the EEPROM if the value is written is different from the one already saved.

As the EEPROM has **limited life** expectancy due to limited write/erase cycles, using the `EEPROM.update()` function instead of the `EEPROM.write()` saves cycles.

Syntax: `EEPROM.update(address, value);`

Task

- Try the described example mentioned earlier and see the output!