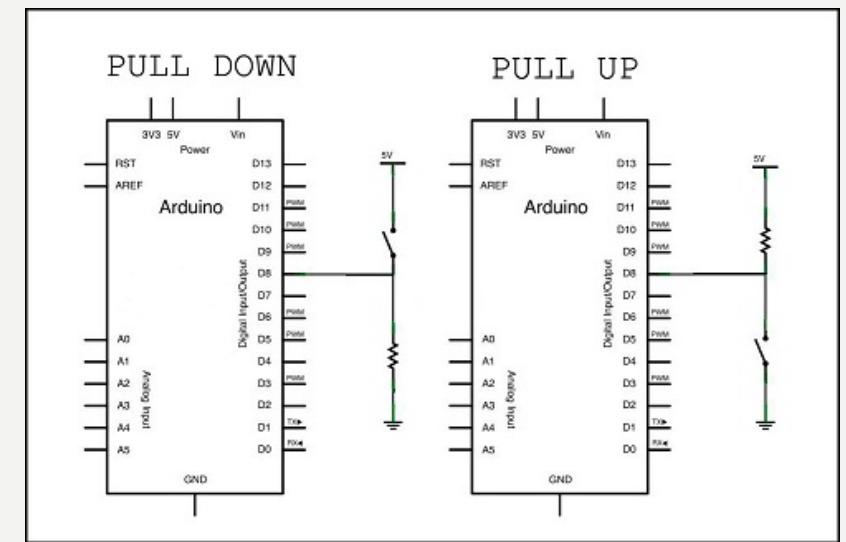


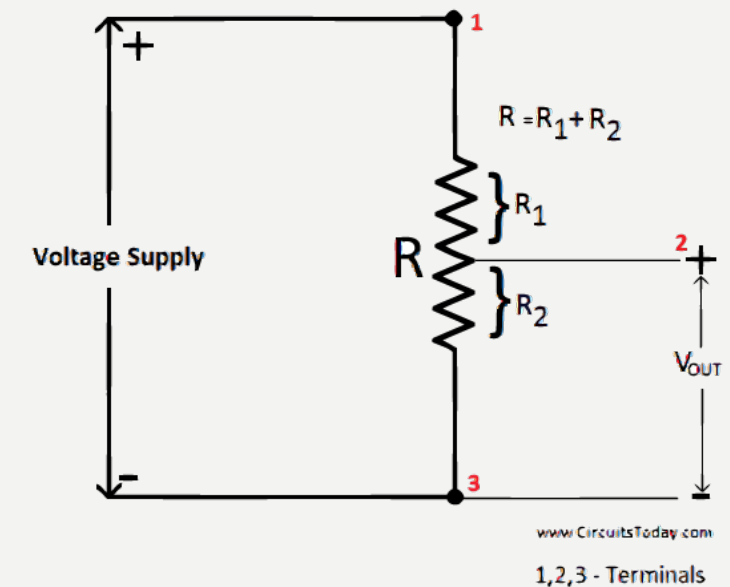
INTERFACING WITH THE ARDUINO

**ZIED JARRAYA
SAIFEDDINE BARKIA
MOHAMED AZIZ TOUSLI**

ELECTRIC CIRCUITS



- Terminal = Pôle / Soldering = Soudure / Short circuit = Court circuit
- Diode:
 - Forward Bias :
 - \geq Threshold: PASS
 - \leq Threshold: NO PASS
 - Reverse Bias: NO PASS
- Potentiometer: $R_1 + R_2 = \text{cte}$ (Top +, Middle Out, Bottom -)



SENSORS

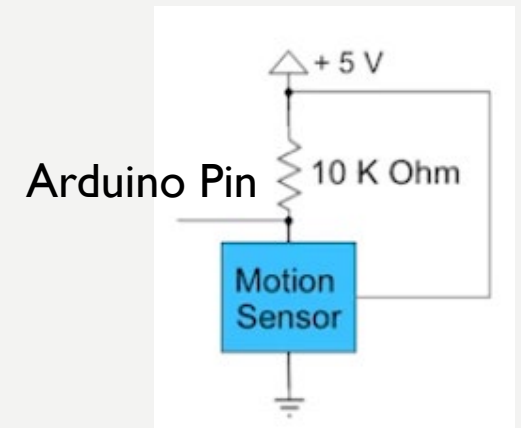
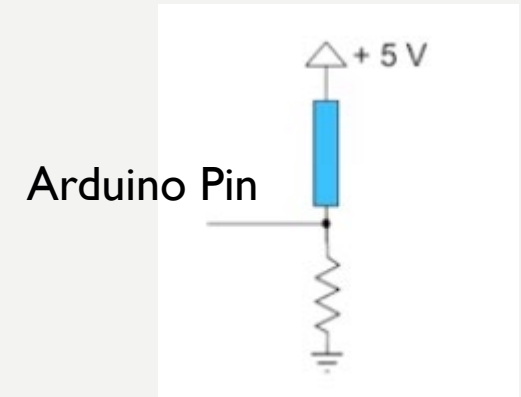
- Get information from environment
- Resistance $\sim \rightarrow$ Voltage $\sim \rightarrow$ Arduino pin receives a digital value

☐ Resistive sensors (Resistance)

EXP: Photoresistor

☐ Voltage controlling sensors (Pull Up Path)

EXP: Accelerometer (2D) ; Gyroscope (3D)



ACTUATORS

- Send information to environment

☐ On-off actuators (digital)

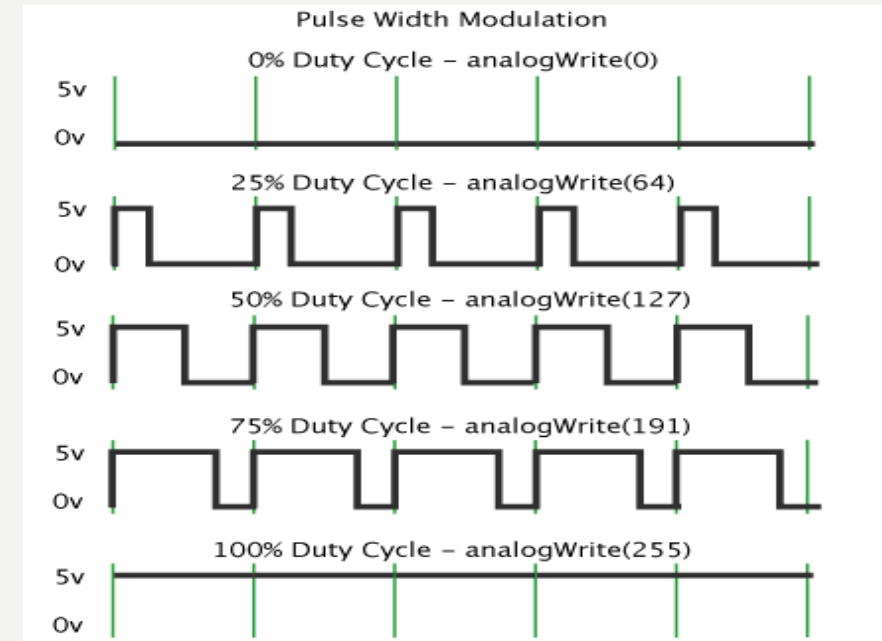
☐ Analog voltage control actuators

⊖ DAC ⇔ Pulse Width Modulation PWM

○ Duty cycle – **AnalogWrite(Pin PWM ~, Value 0 to 255 [0% to 100%])**

○ Example: 50% – Duty cycle: Half the time 1, Half the time 0 → Give 2,5V

✓ EXP: Buzzer – **tone(pin, frequency in Hz, duration in ms)** #Generate square wave; Fixed duty cycle 50% → **notone()**



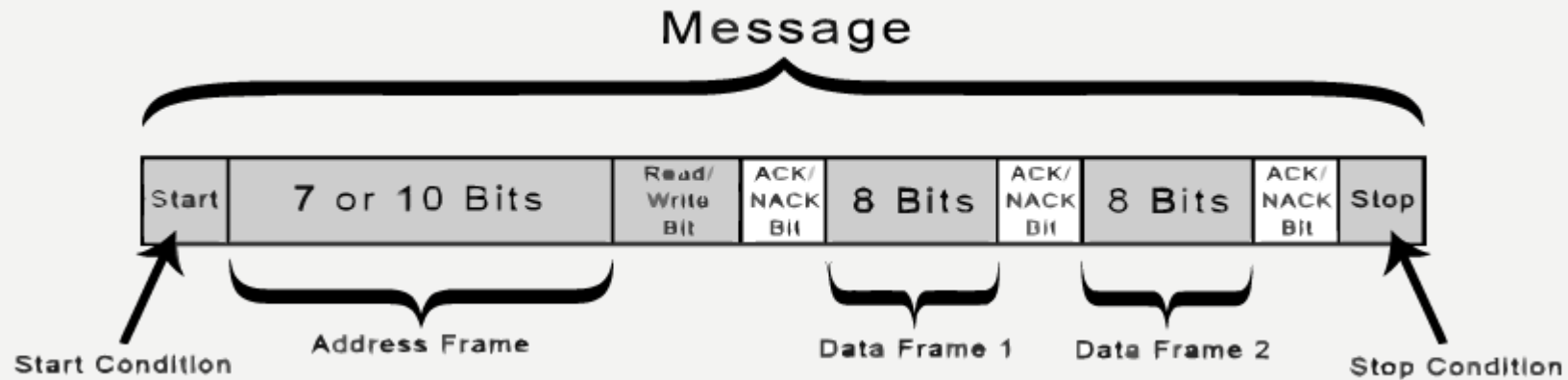
EEPROM


- EEPROM: Electronically Erasable Programmable Read Only Memory
- Non volatile memory
- Writes a single byte at a time (Similar to flash memory)
- 1k bytes for Arduino
- Masking: Process to store data of 2 bytes (1's in the bits we interested in and 0's)
- PS: 0xFF is a mask for 255
- ❖ `#include<EEPROM.h>`
- ❖ `EEPROM.read(address | byte 0->1023)`
- ❖ `EEPROM.write(address, data | byte)`

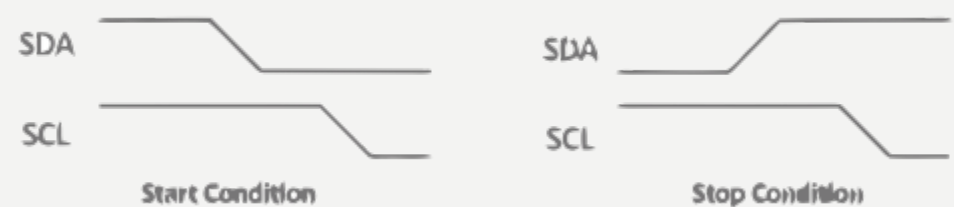
I2C COMMUNICATION

- ❖ Serial protocol → 1 wire, 1 bit at a time → Saves pins
 - ❖ Synchronous protocol → Shares same clock
 - Master: Who starts the communication
 - Slave: Who receives the communication
 - Bit width fixed: 2 bits
 - 1 bit for data signal
 - 1 bit for clock signal
 - 2 Wires:
 - SDA: Serial Data
 - SCL: Serial Clock
 - ◻ Transmitter: Places data on the bus
 - ◻ Receiver: Read data from the bus
- Transaction:
 - Write
 - Read

I2C COMMUNICATION



- SCL – Example of 1 clock pulse 
- SDA – Can be both 0 and 1
 - Cross in SDA – SDA changes value
 - The sender has to apply a value of SDA before SCL goes HIGH and hold it while SCL is HIGH and that is the value that is going to be on the bus.
 - Acknowledge bit: After every byte the receiver sends an ack bit: SDA = 0 : Message received



`#include<Wire.h>`

`Wire.begin()` #No argument = Master ; #Addr(0->127) = Slave

I2C COMMUNICATION

Master communication (Write):

1. `Wire.beginTransmission(@Slave)` #Start the transmission
2. `Wire.write(data)`
3. `Wire.endTransmission()` #Returns 0 for success

PS: Whole data is put into a buffer before being sent.

Master communication (Read):

- ❖ `Wire.requestFrom(@Slave, Nb of bytes to read, X)` #X=Stop arg to release the bus after(optional)
#Specify a read transaction
- ❖ `Wire.read()` #Returns a single byte from the receiver buffer
- ❖ `Wire.available()` #Returns number of bytes waiting

I2C COMMUNICATION

Slave operation:

PS: Slave cannot initiate for transmission : must wait.

- Problem → Busy wait: loops to check if there is transaction
- Solution → Call back functions: functions called when an event occurs “transmission is received”
- ❖ **Wire.onReceive(funcName)** #Master: Write Transaction
- ❖ **Wire.onRequest(funcName)** #Master: Read Request
- ✓ Exp: **void receiveFunction(int bytesReceived)** #1 argument
- ✓ Exp: **void transmitFunction(void)**

SHIELDS – ETHERNET SHIELD

Shield = Printed Circuit Board (PCB) → Adds functionality to Arduino

- Hardware: Circuit prewired (same size as Arduino)
- Software: Library associated to IDE

Exp: Ethernet shield ;

❑ MAC address : Unique – 6 bytes long

❑ IP address : Changeable – 4 bytes long

❑ Port : Application protocol – 2 bytes long ; Exp: Port 80 (for Internet)

❑ DNS: Domain Name Server → Match between IP@ and URL

❑ DHCP: Dynamic Host Connection Protocol → Gives an IP@ dynamically | Static IP@

PS: Routers – DHCP / Servers – Static IP@

❖ `#include<Ethernet.h>`

❖ `Ethernet.begin(@Mac*, @IP, DNS, Gateway, Subnet mask)`

ETHERNET SHIELD

- ❖ `EthernetClient client` #Create a client object
- ❖ `client.connect(@IP / DNS, port)` #returns 1 if ✓
- ❖ `client.stop()`
- ❖ `client.print(data)`
- ❖ `client.write(byte)`
- ❖ `client.read()`
- ❖ `client.available()`

- ❖ `EthernetServer server=EthernetServer(port)` #Create a server object; before setup
- ❖ `EthernetClient client=server.available()`

Server Receives Data

```
EthernetServer server = EthernetServer(80);  
void setup() {  
  Ethernet.begin(mac, ip, gateway, subnet);  
  server.begin();  
}  
void loop() {  
  EthernetClient client = server.available();  
  if (client) {  
    Serial.print(client.read());  
  }  
}
```

Client Sends Data

```
byte mac[]={0xDE, 0xAD, 0xBE, 0xEF, 0x12, 0x34};  
char server[] = "testdomain.edu";  
EthernetClient client;  
void setup() {  
  Ethernet.begin(mac);  
  if (client.connect(server, 80)) {  
    client.println("GET index.html HTTP/1.1");  
    client.stop();  
  }  
}
```

WIFI SHIELD

`WiFi.begin(ssid,pass°,keyindex*,key*)` #ssid=network ; pass type=WPA-2°,WEP*

- `WiFiClient client` #Create a client object
- `client.connect(ip,port)`
- `client.stop()`
- `WiFiServer server(port)` #create a server object
- `server.begin()`
- ❖ `WiFi.scanNetworks()` #returns the number of available networks
- ❖ `WiFi.SSID(i)` #returns ssid of the i'th network
- ❖ `WiFi.RSSI(i)` #returns the strength of the i'th network (-80 → 0)
- ❖ `WiFi.encryptionType()` #0 WEP ; 1 WPA-1 ; 2 WPA-2 ; 3 None