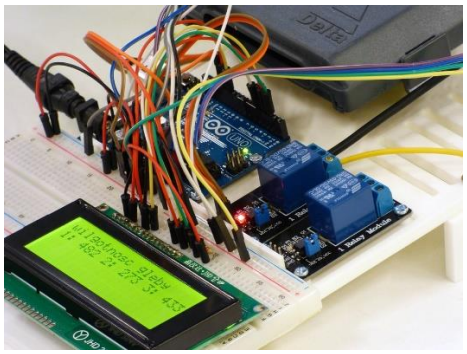
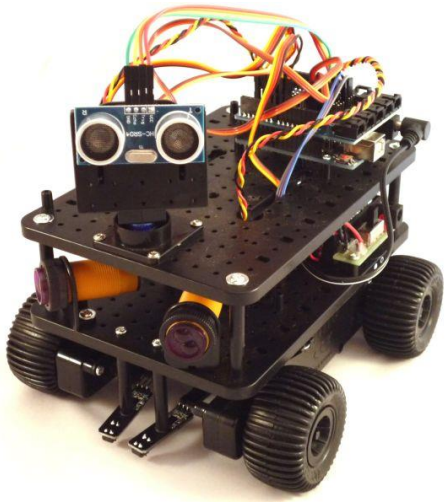
Apuntes y Prácticas de Arduino





Bloque de Sistemas de control.

Tecnología Industrial

Contenido

1.-Introducción 3

2.-Manejo de LEDs 3

3.-Uso de pulsadores 4

4.- Sensor BMP280: medida de la presión y temperatura 6

5.-Puerto Serie 10

6.- Emisor/receptor de radiofrecuencia APC220 12

7.-Emisor/recptor de radiofrecuencia HC-12 .....

8.-Servos 18

9.-Lector tarjetas microSD 21

10.-Sensor humedad DHT11 23

1.-Introducción

Canal de Youtube Bitwise Ar: https://www.youtube.com/watch?v=eBVvD85Ml2c&t=16s

Tipos de variables →https://arduino.cl/introduccion-a-los-tipos-de-dato-con-arduino/

2.-Manejo de LEDs

https://www.youtube.com/watch?v=GUuWgk3dXd0

1. Intermitente led del pin 13

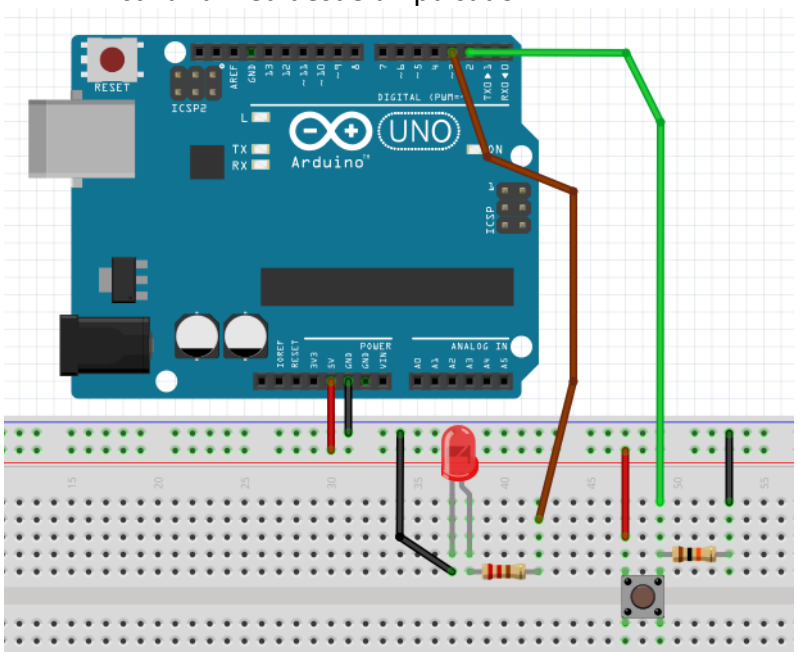
```
void setup () {
  pinMode (13, OUTPUT);   //pin 13 como salida
}

void loop(){
  digitalWrite(13,HIGH);        // pin 13 a nivel alto
(1 lógico)
  delay(1000);           //retardo de 1 segundo
  digitalWrite(13,LOW);
  delay(1000);
}
```

2. Encender un led en otro pin
3. Apagar un led
4. Encendido y apagado de dos leds alternativamente
5. Luces coche fantástico con 4 leds (Vamos encendiéndolos hasta quedar todos encendidos, y después los apagamos uno a uno hasta que queden todos apagados.)
    a. Maneja led a led
    b. Utiliza un bucle FOR
6. Aprovechando el montaje vamos a crear dos funciones: *encender()*, que enciende los leds y *apagar()* que los apaga.

```
int i=0;

/***   Function declaration   ***/
void encender (); //anuncia las funciones que usará
void apagar ();

void setup() {
        for (i = 10; i <= 13; i++) {
                pinMode(i,OUTPUT);
                }
}
void loop() {
        encender(); //llamamos a la función
        delay(200);
        apagar();  //llamamos a la función
        delay(200); }

/***   definición de funciones   ***/
void encender () {
        for (i = 10; i <= 13; i++) {
                digitalWrite(i,LOW);
                }
                 }
void apagar () {
        for (i = 10; i <= 13; i++) {
                digitalWrite(i,HIGH);
        }
}
```

3.-Uso de pulsadores

1. Activar un led desde un pulsador:



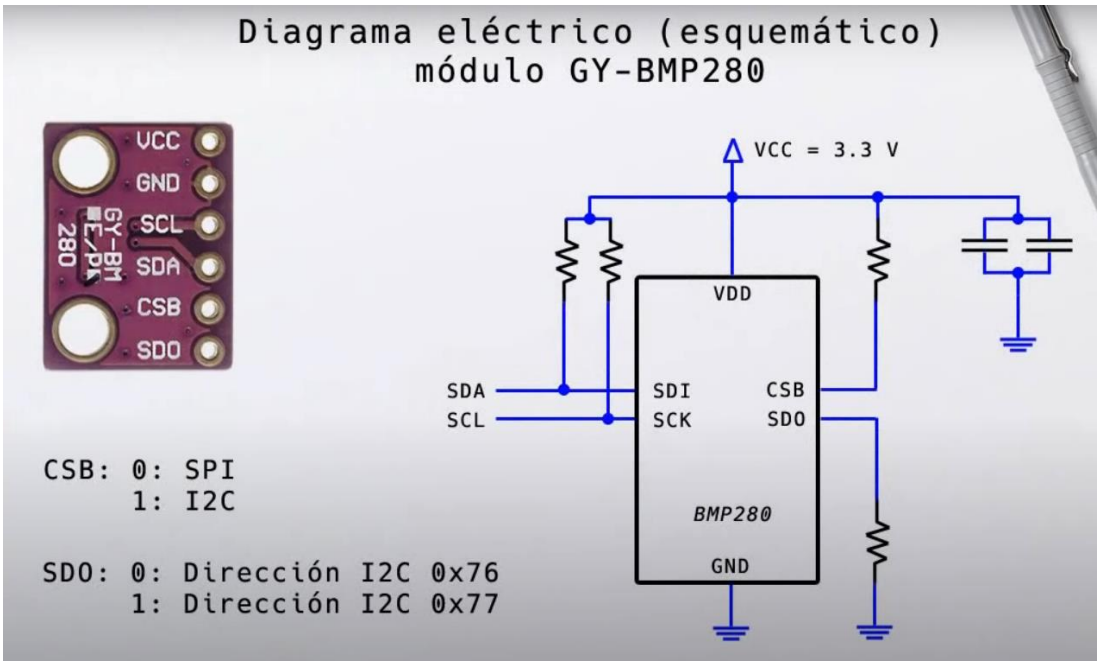
```
void setup() {

  pinMode (2,INPUT);              //pin 2 como entrada
  pinMode (3, OUTPUT);//pin3 como salida
}

void loop() {
  if (digitalRead(2)==HIGH){      //comprobamos si el el
  pulsador está accionado
      digitalWrite (3, HIGH);     //activamos el LED si el
      pulsador está accionado
  }
  else{
  digitalWrite (3,LOW);
  }
}
```

2. Haz un programa en el que se apague un LED al accionar un pulsador
3. (VERSIÓN 2) Encender un LED con botón pulsador y luego apagarlo con el mismo botón.

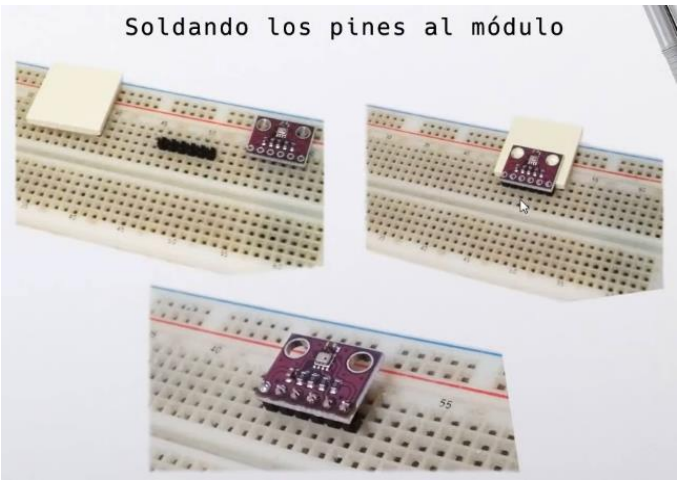
```
boolean enciende = LOW; //almacenamos el estado del
pulsador
void setup() {
          pinMode(8, INPUT); //pin 8 para el pulsador
          pinMode(7, OUTPUT); //pin 7 para el LED
          Serial.begin(9600);
}

void loop() {
  if (digitalRead(8) == 1) {
    enciende = !enciende;          //cada vez que pulsamos
cambia de estado
    Serial.println(enciende);      //para ver el estado de esa
variable
    //delay(200);      // Observa que ocurre con esta línea y sin
ella. Trata de explicarlo
          if (enciende) {
                digitalWrite(7, HIGH);
          } else {
                digitalWrite(7, LOW);
                }
          }
}
```

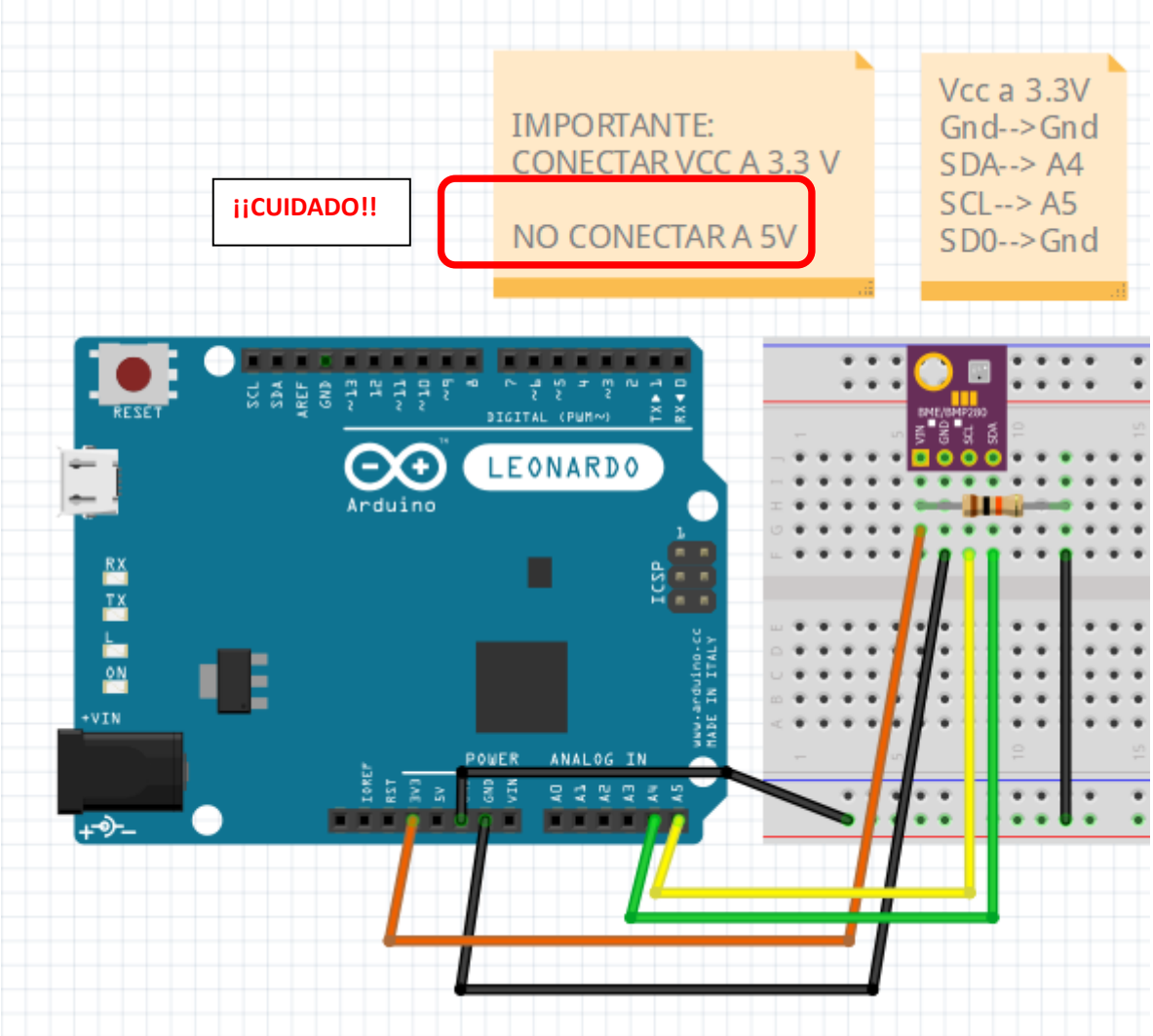
4.- Sensor BMP280: medida de la presión y temperatura

100 Pa= 1hPa (hectopascal)
1hPa=1mb (milibar)



Por defecto, este módulo está configurado para funcionar con la interfaz I2C en la dirección 0x76





- *Abrimos el IDE de Arduino y vamos a Programa>Incluir Librería>Administrar Bibliotecas*
- *Se actualizarán las Librerías.*
- *En el campo de búsqueda escribir: Adafruit BMP 280. Pulsar instalar.*

*Por un olvido del programador, **hay que modificar un archivo**. En **Windows** (C:) vamos a Documentos>Arduino>Libraries>Adafruit_BMP280_Library>Botón derecho sobre Adafruit_BMP280 y seleccionar Abrir con> WordPad, Cambiar la línea:* `#define BMP280_ADDRESS (0x77)` por `#define BMP280_ADDRESS (0x76)`
Es conveniente también enviar la salida SD0 a masa (Gnd) para asegurarnos que toma la dirección (0x76).

El código fuente se puede bajar de: https://github.com/bitwiseAr/Curso-Arduino-desde-cero/ (Capítulo 36)

Programa 1

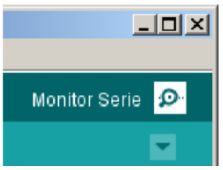
```
#include <Wire.h>  // incluye librería de bus I2C
#include <Adafruit_Sensor.h>  // incluye librerías para sensor BMP280
#include <Adafruit_BMP280.h>

Adafruit_BMP280 bmp;   // crea objeto con nombre bmp
float TEMPERATURA;   // variable para almacenar valor de temperatura
float PRESION;     // variable para almacenar valor de presion atmosferica

void setup() {
 Serial.begin(9600);      // inicializa comunicacion serie a 9600 bps
 Serial.println("Iniciando:");    // texto de inicio
 if ( !bmp.begin() ) {      // si falla la comunicacion con el sensor mostrar
  Serial.println("BMP280 no encontrado !"); // texto y detener flujo del programa
  while (1);         // mediante bucle infinito
 }
}

void loop() {
 TEMPERATURA = bmp.readTemperature();   // almacena en variable el valor de temperatura
 PRESION = bmp.readPressure()/100;   // almacena en variable el valor de presion divido
        // por 100 para covertirlo a hectopascales
 Serial.print("Temperatura: ");   // muestra texto
 Serial.print(TEMPERATURA);     // muestra valor de la variable
 Serial.print(" C ");       // muestra letra C indicando grados centigrados
 Serial.print("Presion: ");     // muestra texto
 Serial.print(PRESION);     // muestra valor de la variable
 Serial.println(" hPa");    // muestra texto hPa indicando hectopascales
 delay(5000);         // demora de 5 segundos entre lecturas
}
```

Programa 2

```
#include <Wire.h>  // incluye libreria de bus I2C
#include <Adafruit_Sensor.h>  // incluye librerias para sensor BMP280
#include <Adafruit_BMP280.h>
Adafruit_BMP280 bmp;   // crea objeto con nombre bmp
float TEMPERATURA;   // variable para almacenar valor de temperatura
float PRESION;     // variable para almacenar valor de presion atmosferica

void setup() {
 Serial.begin(9600);      // inicializa comunicacion serie a 9600 bps
 Serial.println("Iniciando:");    // texto de inicio
 if ( !bmp.begin() ) {      // si falla la comunicacion con el sensor mostrar
  Serial.println("BMP280 no encontrado !"); // texto y detener flujo del programa
  while (1);        // mediante bucle infinito
 }
}

void loop() {
 TEMPERATURA = bmp.readTemperature();   // almacena en variable el valor de
temperatura
 PRESION = bmp.readPressure()/100;  // almacena en variable el valor de presion
divido
        // por 100 para covertirlo a hectopascales
 Serial.print("Temperatura: ");   // muestra texto
 Serial.print(TEMPERATURA);     // muestra valor de la variable
 Serial.print(" C ");       // muestra letra C indicando grados centigrados

 Serial.print("Presion: ");     // muestra texto
 Serial.print(PRESION);     // muestra valor de la variable
 Serial.println(" hPa");    // muestra texto hPa indicando hectopascales

 delay(5000);        // demora de 5 segundos entre lecturas
}
```

5.-Puerto Serie

El puerto serie es una forma de comunicar Arduino con otros dispositivos, para enviar o recibir datos. El envío se lo haremos típicamente por teclado, aunque podría hacerse vía Bluetooth (mediante una app de terminal en un móvil, p.ej.) o a través de una WIFI.



Para acceder al monitor lo podemos hacer con el botón de la parte superior derecha

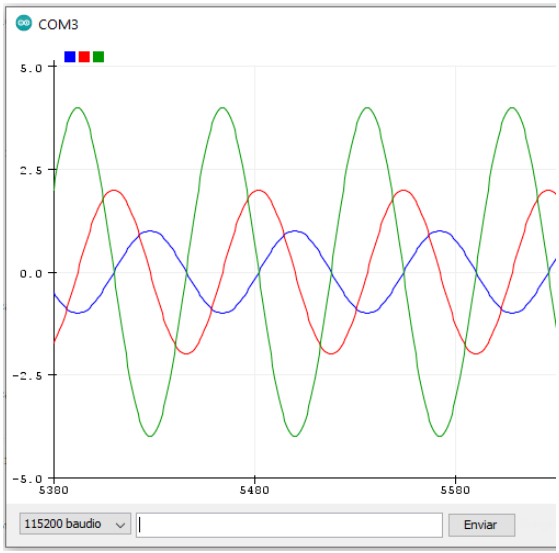
O por menú. Ahí también aparece el Serial Plotter que nos hará una representación gráfica en tiempo real si vamos mandando datos desde la placa

Esta es la apariencia que tiene en Monitor Serie

Los programas que utilizan el monitor serie deben tener estas ordenes:

```
void setup(){
  Serial.begin(9600); // Velocidad de comunicación
                      // La velocidad del puerto serial debe ser
                      // la misma que la de configuración del
modulo
}
void loop(){
}
```

Ejemplo de uso del PLOTTER

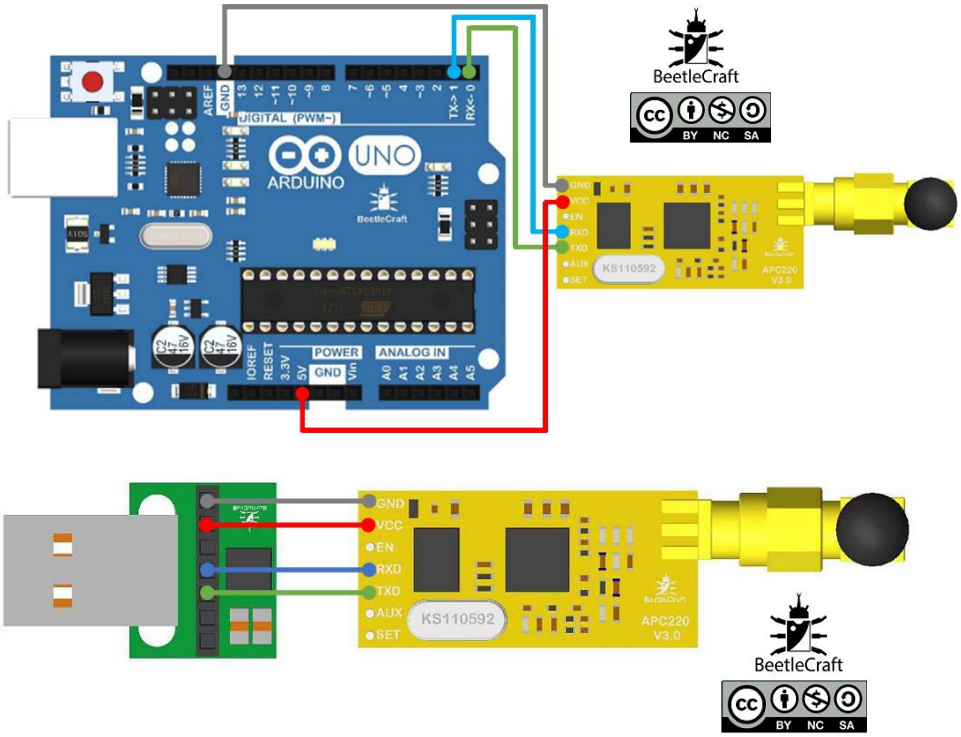


```
void setup() {
  // preparamos el puerto serie a 115200 baudios
  Serial.begin(115200);
}

/**
   Función loop: se ejecuta continuamente mientras el arduino permanece
encendido
 */
void loop()
{
  for (int i = 0; i < 360; i += 5)
  {
    // generar 3 señales senoidales con distinta amplitud y fase
    float y1 = 1 * sin(i * 2*M_PI / 360);
    float y2 = 2 * sin((i + 90) * 2*M_PI / 360);
    float y3 = 4 * sin((i + 180) * 2*M_PI / 360);

    // primer valor a graficar
    Serial.print(y1);
    // imprimimos un caracter de espacio o tabulador entre cada valor
    Serial.print("\t");
    // segundo valor a graficar
    Serial.print(y2);
    // imprimimos un caracter de espacio o tabulador entre cada valor
    Serial.print("\t");
    // imprimimos el ultimo valor y avanzamos a la siguiente linea de texto
    // se usa la función println para imprimir también el fin de linea
    Serial.println(y3);

    // esperamos 50 milisengundos antes de enviar un nuevo punto en la gráfica
    delay(50);
  }
}
```

Ahora vamos a cargar la librería "separar.zip" en el IDE de Arduino: Programa>Incluir Librería>Añadir biblioteca.zip
Seleccionamos la biblioteca "separar.zip" desde nuestro ordenador.

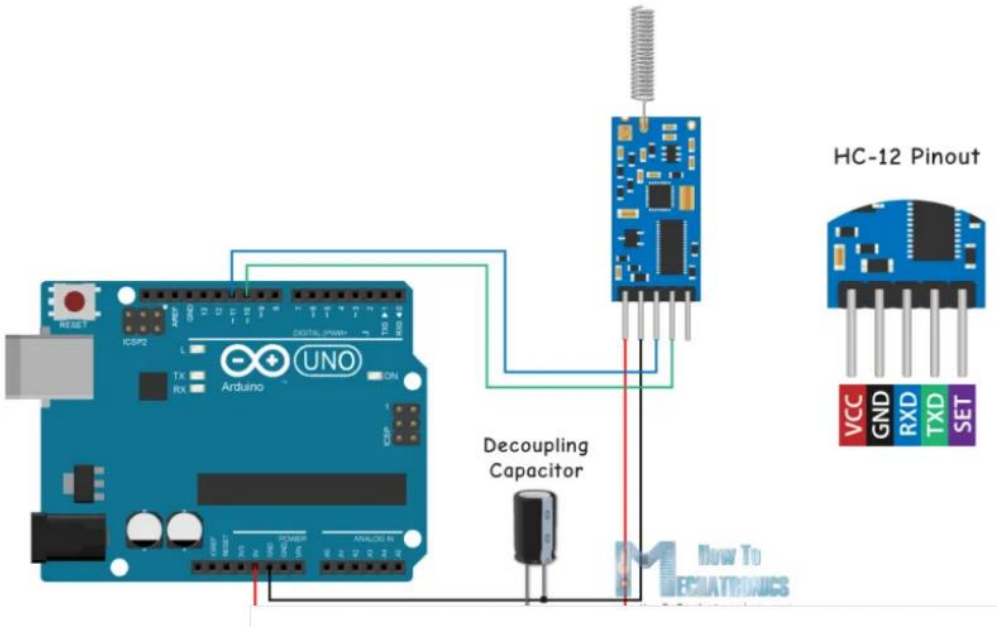
```
#include <Separador.h> //esta línea se puede insertar clicando el Programa>Incluir Libreria y seleccionando la librería correspondiente
Separador s;
void setup() {
  Serial.begin(9600);
  Serial.println("Ingresa un texto con 3 cadenas separadas por una coma. Ej: Hola, cómo, estas. El programa las separará");
}

void loop(){
  if(Serial.available(){
    serial.Event();
  }
void serial.Event(){
  String datosRecibidos=Serial.readString();
  //separamos los datos recibidos
  String dato1=s.separa(datosRecibidos,",",0);
  String dato2=s.separa(datosRecibidos,",",1);
  String dato3=s.separa(datosRecibidos,",",2);

  Serial.println("El dato 1 es:"+dato1);
  Serial.println("El dato 2 es:"+dato2);
  Serial.println("El dato 3 es:"+dato3);
  Serial.println();
  Serial.println();
  Serial.println("Ingresa un texto con 3 cadenas separadas por una coma. Ej: Hola, cómo, estas. El programa las separará");

}

}
```

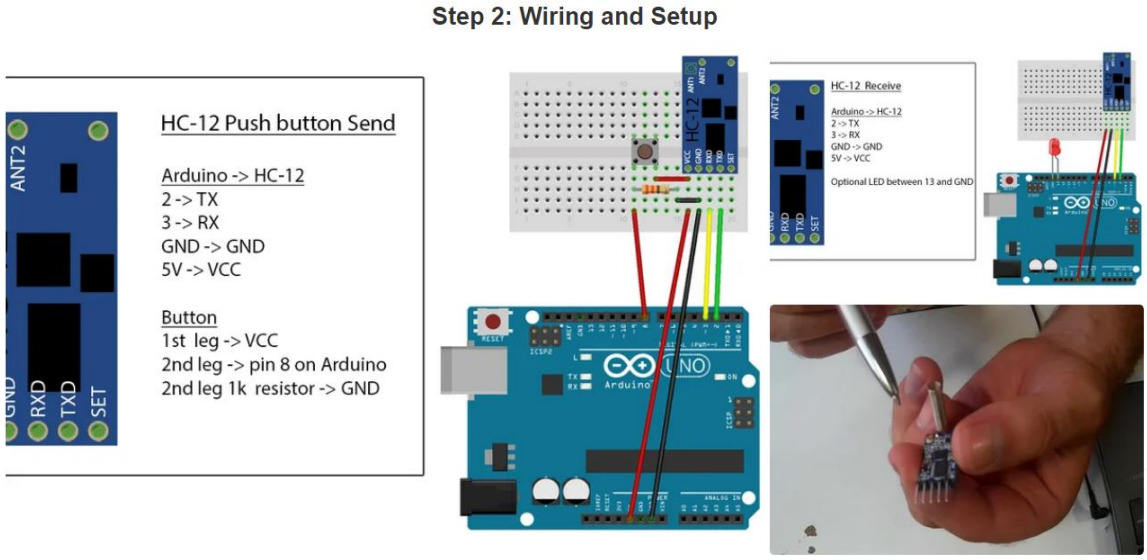
6.- Emisor/receptor de radiofrecuencia APC220



Para enviar y recibir datos con este módulo se utiliza el puerto serie.

7.- Emisor/receptor de radiofrecuencia HC-12





Resistencia de 10KOhm para el divisor de tensión del pulsador de entrada.

Quizá sea necesario un cable adaptador de conector IPEX a conector SMA y una antena SMA.

Descripción
El Transceptor HC-12 de 433 MHz es un módulo de comunicación de puerto serie inalámbrico, el cual se basa en el chip RF SI4463, tiene un microcontrolador incorporado y se pueden configurar mediante comandos AT, la potencia de salida máxima es de 100 mW (20 dBm) y la sensibilidad del receptor difiere de: 117dBm a -100dBm, dependiendo de la velocidad de transmisión. Acepta 3.2V-5.5V y se puede usar con dispositivos de voltaje UART de 3.3V y 5V.

Características

- Transmisión inalámbrica de larga distancia (1,000 m en espacio abierto / velocidad de transmisión de 5,000bps en el aire)
- Rango de frecuencia de trabajo (433.4-473.0MHz, hasta 100 canales de comunicación)
- Máxima potencia de transmisión de 100mW (20dBm) (se pueden configurar 8 engranajes de potencia)
- Tres modos de trabajo, adaptándose a diferentes situaciones de aplicación.
- MCU incorporado, que realiza la comunicación con un dispositivo externo a través del puerto serie
- El número de bytes transmitidos ilimitado a una vez

Especificaciones:

- Frecuencia de trabajo: 433.4MHz a 473.0MHz
- Voltaje de alimentación: 3.2V a 5.5VDC
- Distancia de comunicación: 1.000m en espacio abierto.
- Velocidad de transmisión en serie: 1.2Kbps a 115.2Kbps 9. 9.6Kbps por defecto）
- Sensibilidad de recepción: -117dBm a -100dBm
- Potencia de transmisión: -1dBm a 20dBm
- Protocolo de interfaz: UART / TTL
- Temperatura de funcionamiento: -40 ℃ a + 85
- Dimensiones: 27.8mm x 14.4mm x 4mm

Modos de Funcionamiento:
Cada HC-12 puede funcionar en los siguientes modos:

1. FU1: modo de ahorro de energía moderado con una velocidad de transmisión de 250000bps “por aire”. La velocidad en baudios del puerto serie se puede establecer en cualquier valor admitido
2. FU2: modo de ahorro de energía extremo con 250000bps de velocidad “por aire”. La velocidad del puerto serie está limitada a 1200bps, 2400bps, 4800bps
3. FU3 – predeterminado, modo de propósito general. La velocidad “por aire” varía según la velocidad del puerto en serie. Lo mismo ocurre con el rango máximo:
    - 1200bps ~ 1000m
    - 2400bps ~ 1000m
    - 4800bps ~ 500m
    - 9600 bps ~ 500m
    - 19200bps ~ 250m
    - 38400bps ~ 250m
    - 57600bps ~ 100m

- 115200bps ~ 100m

4. FU4 (disponible en la versión 2.3 o posterior): modo de largo alcance. La velocidad “en el aire” está limitada a 500bps y la velocidad del puerto serie a 1200bps. Debido a que la velocidad del aire es inferior a la velocidad del puerto, solo se pueden enviar paquetes pequeños: 60 bytes como máximo con un intervalo de 2 segundos. En este modo el rango se incrementa a 1800m.

El HC-12 que crea un enlace inalámbrico tiene que funcionar en el mismo modo (FU1, FU2, FU3, FU4) y con la misma velocidad.

Configuración:

El HC-12 se puede configurar usando el comando AT. La mejor manera de hacerlo, es usar un convertidor de USB a serie como CP2102. Para poner el HC-12 en modo AT, presione el pin SET a GND.

Los comandos más importantes:

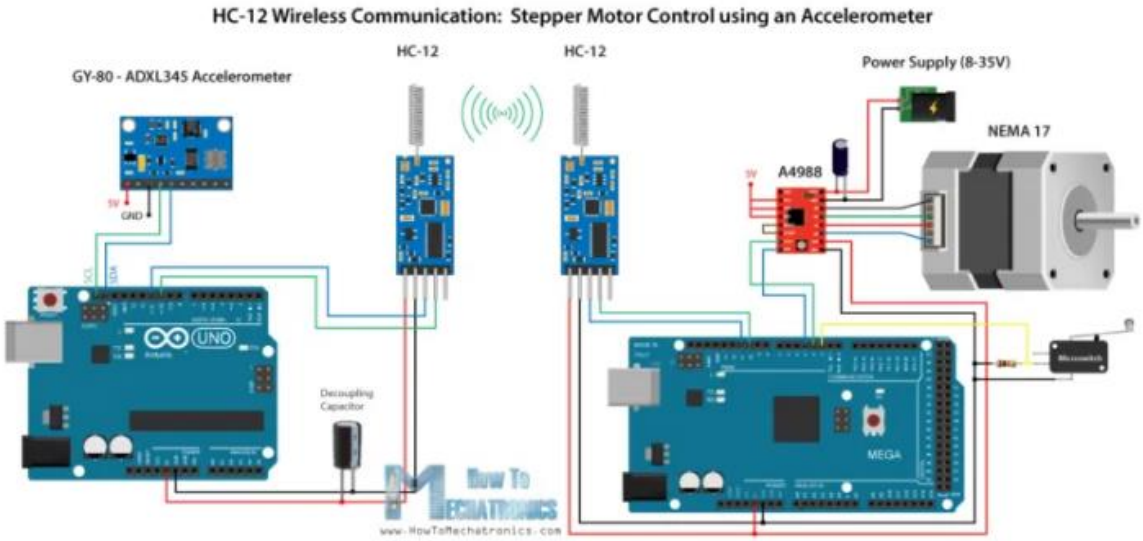
1. 1. AT – comando de prueba. Retornará OK si la interfaz AT está habilitada
2. 2. AT + Bxxxx: establece la velocidad en baudios del puerto serie. Por ejemplo, AT + B57600 establece la velocidad en baudios a 57600bps
3. 3. AT + Cxxx – establece el canal de radio. Los canales comienzan desde 001 a 433,4MHz. Cada siguiente canal añade 400kHz. El canal 100 es 473,0MHz. AT + C002 ajustará la frecuencia a 433,8MHz. Dos dispositivos HC-12 que crean un enlace inalámbrico tienen que operar en la misma frecuencia
4. 4. AT + FUx – configura el modo de dispositivo: FU1, FU2, FU3 o FU4. Dos dispositivos HC-12 que crean un enlace inalámbrico tienen que usar el mismo modo
5. 5. AT + Px: configura la potencia de transmisión del dispositivo. Por ejemplo, AT + P2 ajusta la potencia a 2dBm (1.6mW)
   - -1dBm (0.8mW)
   - 2dBm (1.6mW)
   - 5dBm (3.2mw)
   - 8dBm (6.3mW)
   - 11dBm (12mW)
   - 14dBm (25mW)
   - 17dBm (50mW)
   - 20dBm (100mW)
6. 6. AT + RX – recupera todos los parámetros: modo, canal, velocidad de transmisión, potencia
7. 7. AT + V – recuperar la versión del módulo
8. 8. AT + DEFAULT: restablece los parámetros del módulo a la configuración predeterminada

Ver tutorial:

https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-12-long-range-wireless-communication-module/

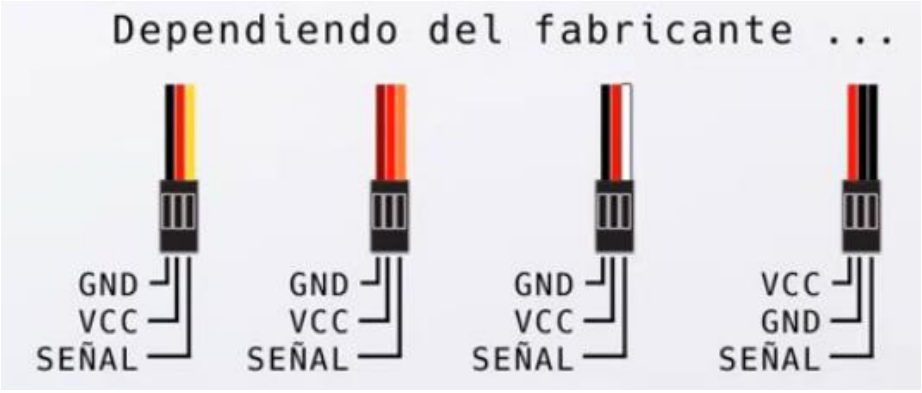
```
/*   Arduino Long Range Wireless Communication using HC-12
                    Example 01
   by Dejan Nedelkovski, www.HowToMechatronics.com
*/

#include <SoftwareSerial.h>

SoftwareSerial HC12(10, 11); // HC-12 TX Pin, HC-12 RX Pin

void setup() {
  Serial.begin(9600);           // Serial port to computer
  HC12.begin(9600);             // Serial port to HC12

}

void loop() {
  while (HC12.available()) {        // If HC-12 has data
    Serial.write(HC12.read());      // Send the data to Serial monitor
  }
  while (Serial.available()) {      // If Serial monitor has data
    HC12.write(Serial.read());      // Send that data to HC-12
  }
}
```

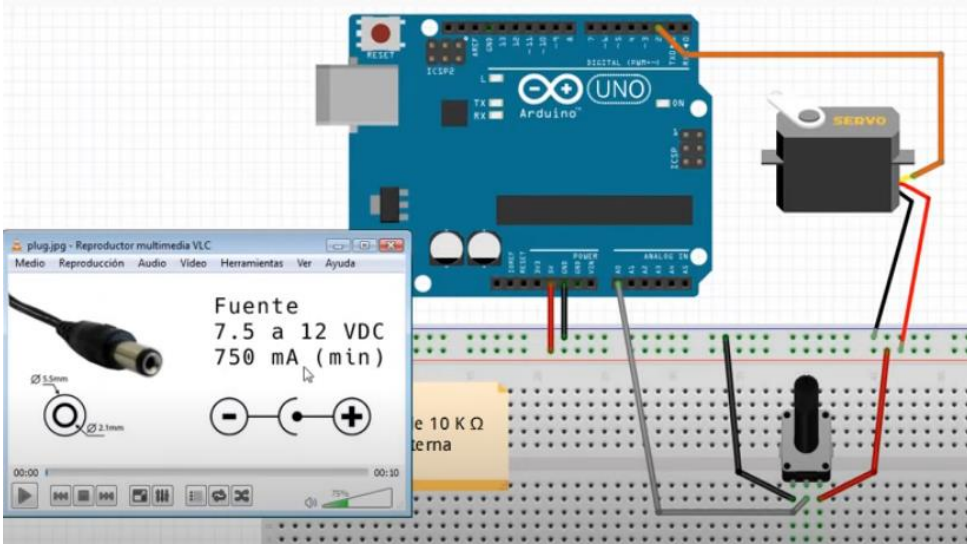
Ejemplo de montaje de un emisor y un receptor (Véase que el emisor utiliza un condensador para mejorar la transmisión):



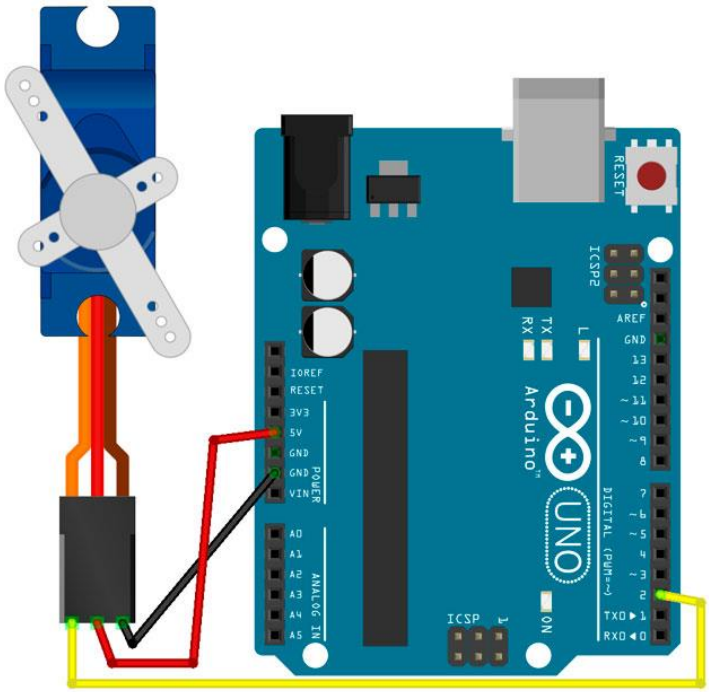
8.-Servos

https://www.youtube.com/watch?v=6bPVZg17vKc&t=24s





Esquema para un micro-servomotor:



Para manejar servos con facilidad conviene utilizar la librería Servo.h

- En la zona de creación de variables declaramos un objeto de la clase Servo
- En el setup debemos inicializarla con *nombre_objeto_creado.attach(PinServo,pulso_max,pulso_min)*
- Para que gire: nombre_objeto_creado.write(ángulo).

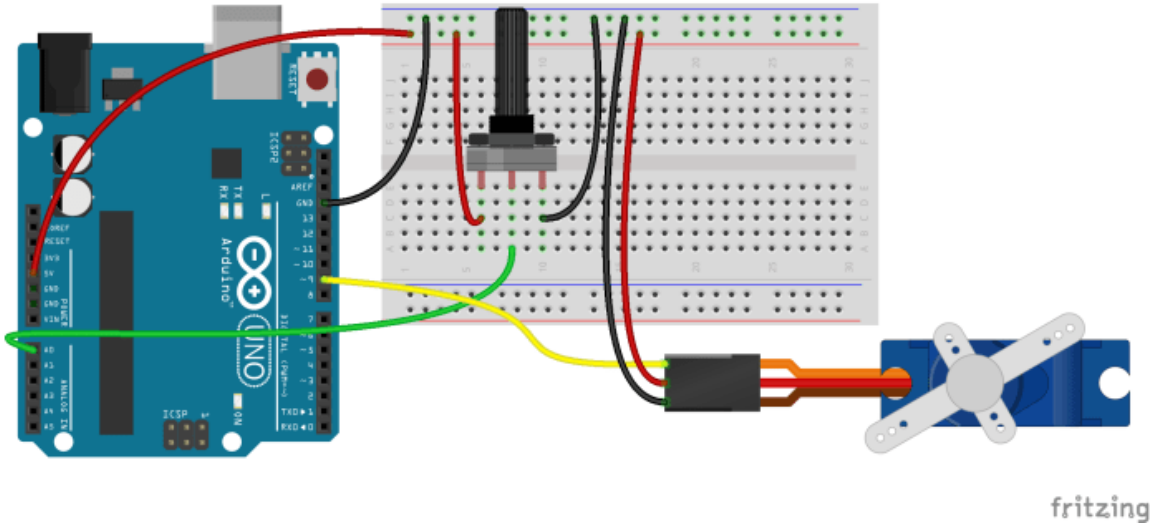
Calibramos nuestro servo:

```
#include <Servo.h>    //Biblioteca con instrucciones de control de servos

Servo servo1;

int PINSERVO = 2;
int PULSOMIN = 750;
int PULSOMAX = 2550;

void setup(){
  servo1.attach (PINSERVO,PULSOMIN,PULSOMAX);
}

void loop(){
  servo1.write(0);        //lo llevamos a ángulo 0
  delay(5000);
  servo1.write(180);     //lo llevamos a ángulo 180
  delay(5000);
}
```

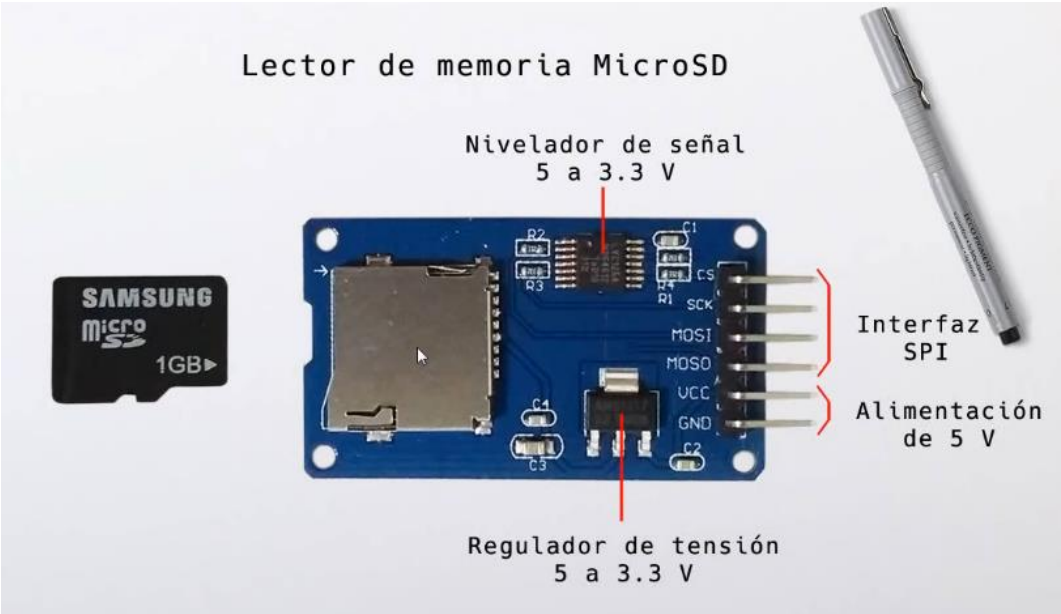
Los servos tienen un valor máximo y mínimo en los pulsos que modulan su giro medido en microsegundos (en nuestro servo PULSOMIN=750 μs y PULSOMAX=2550μs)

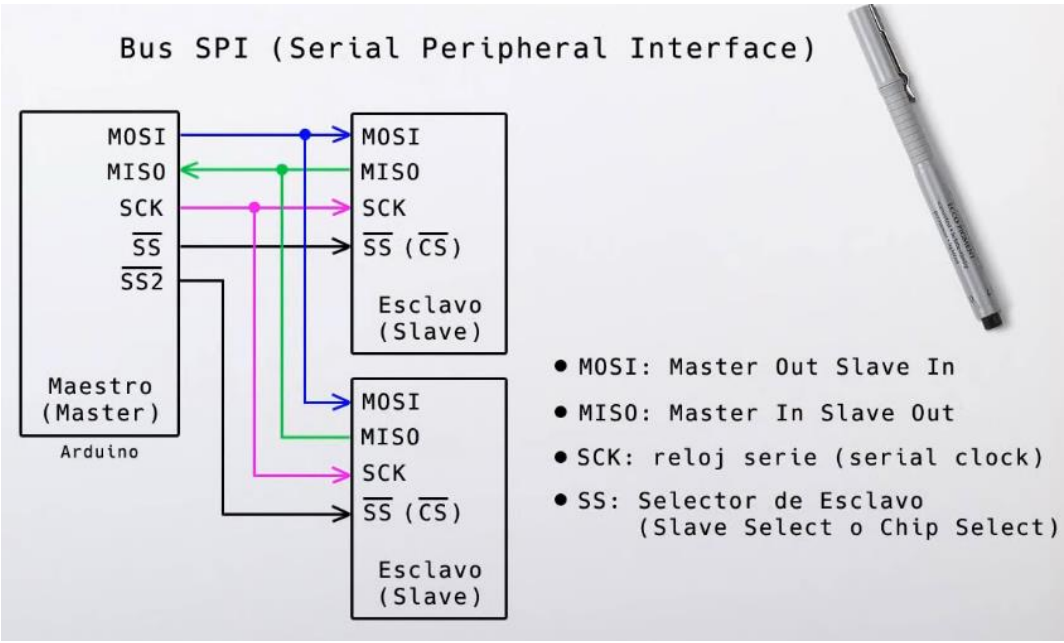
1. Control de la posición de un servo con un potenciómetro

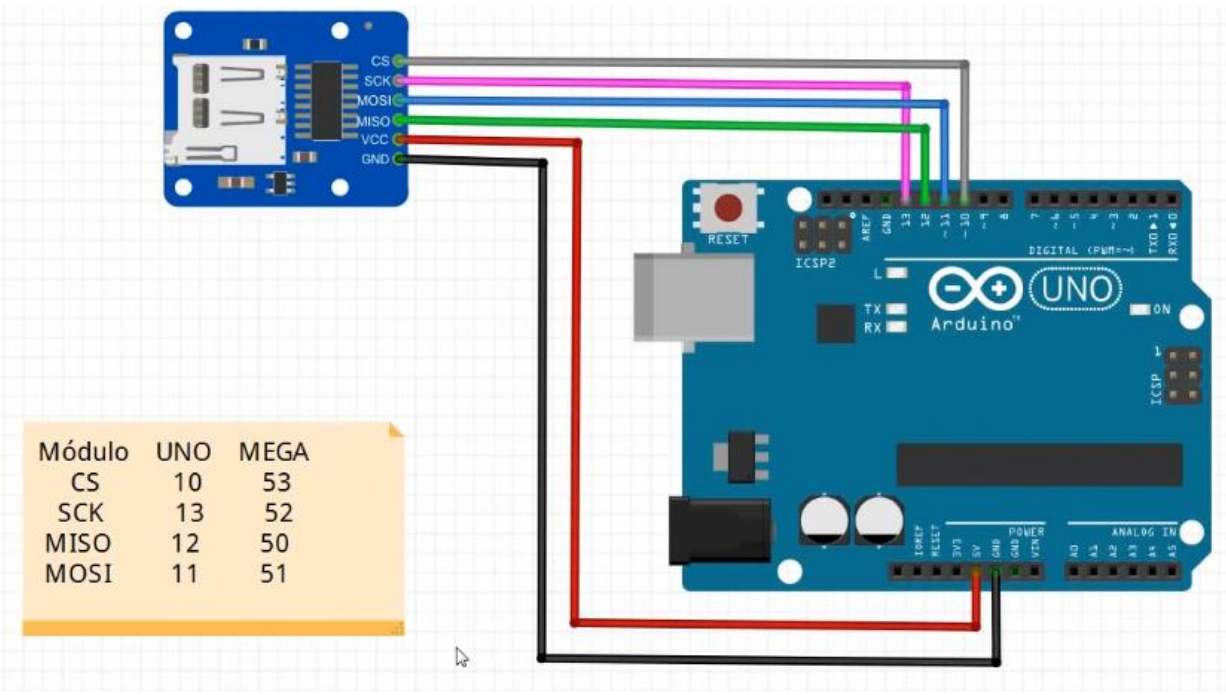
```
#include <Servo.h>

Servo servo1;

int PINSERVO = 2;
int PULSOMIN = 750;
int PULSOMAX = 2550;
int VALORPOT;
int ANGULO;
int POT=0;

void setup(){
 servo1.attach (PINSERVO,PULSOMIN,PULSOMAX);
 //las entradas analógicas no requieren inicialización
}

void loop(){
 VALORPOT = analogRead(POT);
 ANGULO=map(VALORPOT, 0, 1023,0,180);  //cambia el rango de 0-1023 a 0-180
 servo1.write (ANGULO);
 delay (20);                          //le damos tiempo al servo de llegar a su posición
}
```



9.-Lector tarjetas microSD







- La memoria micro SD debe estar formateada en formato FAT (hasta 1Gb) o FAT32.
- No es necesario borrar el contenido de la tarjeta para poder usarlas.

```
#include <SPI.h>              // incluye libreria interfaz SPI
#include <SD.h>                       // incluye libreria para tarjetas SD
#define SSpin 10              // Slave Select en pin digital 10
File archivo;                         // objeto archivo del tipo File

void setup() {
 Serial.begin(9600);                                  // inicializa monitor serie a 9600 bps
 Serial.println("Inicializando tarjeta ...");         // texto en ventana de monitor
 if (!SD.begin(SSpin)) {                              // inicializacion de tarjeta SD
  Serial.println("fallo en inicializacion !");        // si falla se muestra texto correspondiente y
  return;                                             // se sale del setup() para finalizar el programa
 }

 Serial.println("inicializacion correcta");           // texto de inicializacion correcta
 archivo = SD.open("prueba.txt", FILE_WRITE);  // apertura para lectura/escritura de archivo prueba.txt

 if (archivo) {
  archivo.println("Probando 1, 2, 3");  // escritura de una línea de texto en archivo
  Serial.println("Escribiendo en archivo prueba.txt...");     // texto en monitor serie
  archivo.close();                                    // cierre del archivo
  Serial.println("escritura correcta");   // texto de escritura correcta en monitor serie
 } else {
  Serial.println("error en apertura de prueba.txt");          // texto de falla en apertura de archivo
 }

 archivo = SD.open("prueba.txt");                     // apertura de archivo prueba.txt
 if (archivo) {
  Serial.println("Contenido de prueba.txt:");      // texto en monitor serie

  while (archivo.available()) {                       // mientras exista contenido en el archivo

   Serial.write(archivo.read());                      // lectura de a un carácter por vez

  }

  archivo.close();                                    // cierre de archivo

 } else {

  Serial.println("error en la apertura de prueba.txt");// texto de falla en apertura de archivo

 }

}



void loop() {                         // funcion loop() obligatoria de declarar pero no utilizada

 // nada por aqui

}
```

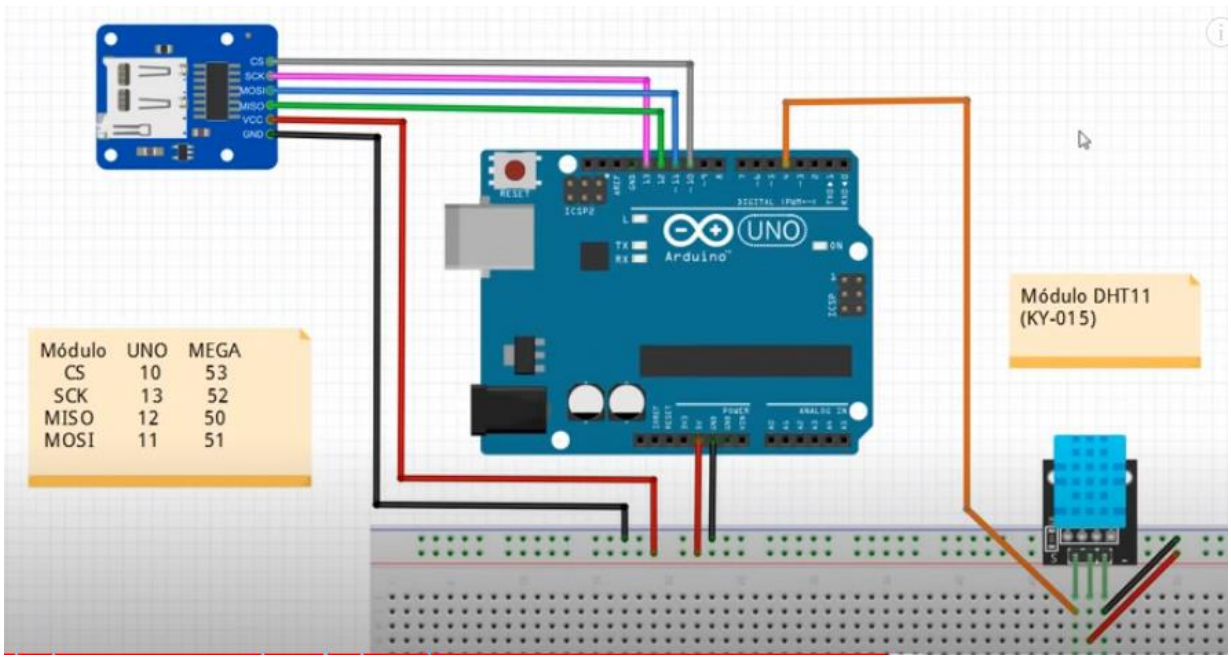
10.-Sensor humedad DHT11

Vídeo https://www.youtube.com/watch?v=2tdsg_K-oQQ

Para manejar el sensor conviene cargar previamente las librerías que indica el vídeo para facilitar su programación.

```
#include <DHT.h>    //Se debe  importar la Librerias DHT
#include <DHT_U.h> //Se debe Importar la librería Adafruit Unified sensor

int Pin_SENSOR = 2;     // pin DATA de DHT11 a pin digital 2
int TEMPERATURA;
int HUMEDAD;

DHT dht(Pin_SENSOR, DHT11);   // creacion del objeto, cambiar segundo
parametro
      // por DHT22 si se utiliza en lugar del DHT11
void setup(){
  Serial.begin(9600);   // inicializacion de monitor serial
  dht.begin();      // inicializacion de sensor
}

void loop(){
  TEMPERATURA = dht.readTemperature();  // obtencion de valor de
temperatura
  HUMEDAD = dht.readHumidity();   // obtencion de valor de humedad
  Serial.print("Temperatura: ");  // escritura en monitor serial de los valores
  Serial.print(TEMPERATURA);
  Serial.print(" Humedad: ");
  Serial.println(HUMEDAD);
  delay(100);
}
```

UTILIZACIÓN DEL LECTOR DE TARJETA SD Y EL SENSOR DE HUMEDAD Y TEMPERATURA DHT11:



```
#include <SPI.h>            // incluye libreria interfaz SPI
#include <SD.h>             // incluye libreria para tarjetas SD
#include <DHT.h>            // incluye libreria DHT de Adafruit
#include <DHT_U.h>          // incluye libreria Adafruit Unified Sensor

#define SENSOR 4            // constante SENSOR en pin digital 4 (señal de DHT11)
int TEMPERATURA;            // variable para almacenar valor de temperatura
int HUMEDAD;                // variable para almacenar valor de humedad

#define SSpin 10            // Slave Select en pin digital 10

File archivo;               // objeto archivo del tipo File
DHT dht(SENSOR, DHT11);             // objeto dht del tipo DHT en pin 4 y modelo DHT11

void setup() {
 Serial.begin(9600);        // inicializa monitor serie a 9600 bps
 dht.begin();               // inicializacion de sensor

 Serial.println("Inicializando tarjeta ..."); // texto en ventana de monitor
 if (!SD.begin(SSpin)) {                    // inicializacion de tarjeta SD
  Serial.println("fallo en inicializacion !");// si falla se muestra texto correspondiente y
  return;                                   // se sale del setup() para finalizar el programa
 }
 Serial.println("inicializacion correcta");  // texto de inicializacion correcta
 archivo = SD.open("datos.txt", FILE_WRITE);        // apertura para lectura/escritura de archivo datos.txt

 if (archivo) {
  for (int i=1; i < 31; i++){               // bucle repite 30 veces
   TEMPERATURA = dht.readTemperature();     // almacena en variable valor leido de temperatura
   HUMEDAD = dht.readHumidity();            // almacena en variable valor leido de humedad

   archivo.print(i);                        // escribe en tarjeta el valor del indice
   archivo.print(",");                      // escribe en tarjeta una coma
   archivo.print(TEMPERATURA);              // escribe en tarjeta el valor de temperatura
   archivo.print(",");                      // escribe en tarjeta una coma
   archivo.println(HUMEDAD);                // escribe en tarjeta el valor de humedad y salto de linea

   Serial.print(i);                         // escribe en monitor el valor del indice
   Serial.print(",");                       // escribe en monitor una coma
   Serial.print(TEMPERATURA);               // escribe en monitor el valor de temperatura
   Serial.print(",");                       // escribe en monitor una coma
   Serial.println(HUMEDAD);                 // escribe en monitor el valor de humedad y salto de
linea

   delay(1000);                             // demora de 1 segundo
   }
  archivo.close();                          // cierre de archivo
  Serial.println("escritura correcta");     // texto de escritura correcta en monitor serie
 } else {
  Serial.println("error en apertura de datos.txt");     // texto de falla en apertura de archivo
 }
}

void loop() {                       // funcion loop() obligatoria de declarar pero no utilizada
 // nada por aquí
}
```