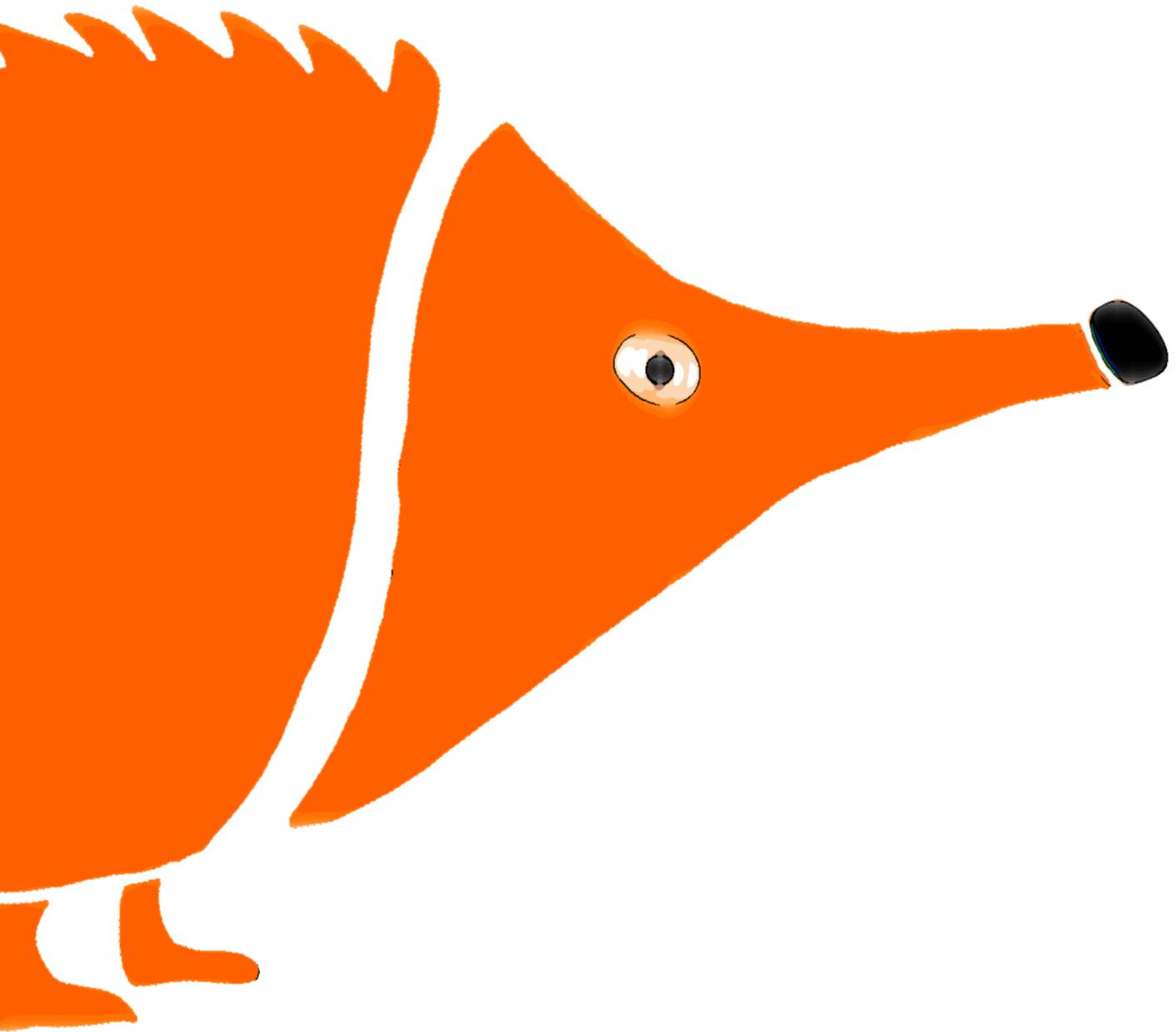


EchidnaBlack

Xabier Rosas



Puesta en marcha y proyectos



Índice

| | |
|--|----|
| Conociendo la placa | 8 |
| Selector alimentación 5V:..... | 12 |
| Selector alimentación Vin (Alimentación externa):..... | 12 |
| Modo sensores..... | 13 |
| Modo MkMk..... | 14 |
| Descripción de los componentes | 15 |
| LED RGB..... | 16 |
| Sonido..... | 17 |
| Pulsadores..... | 18 |
| Palanca de juegos “Joystick”..... | 19 |
| Sensor de Luz: LDR..... | 20 |
| Sensor de temperatura LM35..... | 21 |
| Micrófono..... | 22 |
| Acelerómetro..... | 23 |
| Entradas (Modo MkMk)..... | 24 |
| Pines de libre disposición..... | 25 |
| Conector ISCP “In-Circuit Serial Programming”..... | 26 |
| Programación (hablemos con EchidnaBlack) | 27 |
| IDE de Arduino..... | 28 |
| Estructura de un programa “Sketch” de Arduino:..... | 31 |
| Sketch “Titila_1”..... | 34 |
| Subir el programa..... | 35 |
| LED_PWM..... | 36 |
| Operadores | 37 |
| For..... | 40 |
| LED_PWM_2..... | 40 |
| Colores_RGB..... | 41 |
| Tonos..... | 43 |
| Pulsador_1..... | 46 |
| if...else..... | 47 |
| While..... | 48 |
| Lectura analógica..... | 49 |
| JoyStick_analog..... | 50 |
| Tipos de datos:..... | 52 |
| Propuestas A,B:..... | 53 |
| LDR..... | 54 |
| Theremín LDR..... | 55 |
| Temperatura..... | 56 |
| Vúmetro..... | 59 |
| Nivel eje Y..... | 62 |
| Toma la EchidnaBlack sin activar a Alarma..... | 64 |
| Piano MkMk..... | 68 |
| Piano_2..... | 71 |
| Encender LEDes desde el ordenador..... | 73 |



| | |
|--|------------|
| Ajusta la iluminación del LED desde el ordenador..... | 74 |
| Enciende LEDs desde el Smartphone..... | 75 |
| Comandos AT para HC-06..... | 77 |
| Usando elementos externos..... | 80 |
| Servomotor..... | 80 |
| Sensor de proximidad infrarrojos Sharp 0A41SK..... | 83 |
| Sensor distancia con ultrasonidos HC-SR04..... | 86 |
| Características:..... | 86 |
| Conexiones:..... | 86 |
| Instalar librerías..... | 88 |
| Medida del % de humedad y temperatura con el DHT-11..... | 90 |
| Sensor capacitivo de humedad del terreno..... | 92 |
| EEPROM..... | 98 |
| Sensor capacitivo de humedad con Limites en EEPROM..... | 99 |
| Receptor de infrarrojos..... | 104 |
| IR_Receptor..... | 106 |
| Emisor de Infrarrojos..... | 108 |
| Detectando campos magnéticos..... | 110 |
| Medida de corriente con ACS712..... | 113 |
| Visualizador de cuatro dígitos. TM1637..... | 115 |
| TM1637_Temperatura..... | 115 |
| NeoPixel WS2812B..... | 119 |
| Vumetro con tira de NeoPixel WS2812B..... | 120 |
| Pantalla 5 x 5 NeoPixel WS2812B..... | 123 |
| Analizador de espectro 5 x 5 NeoPixel WS2812B..... | 127 |
| Pantalla Oled bus I ² C..... | 130 |
| Ejemplos..... | 132 |
| Oled_Cuadro_JOY..... | 133 |
| Dibuja un cuadro límite del joystick..... | 133 |
| Oled_Medidor_Analog..... | 135 |
| Anexos | |
| Calculo resistencia serie "Rs"..... | 139 |
| Desenladrillar..... | 140 |
| Usted es libre de..... | 143 |





Tres docentes decidimos crear nuestra propia herramienta para trabajar con robótica en el aula, usando hardware e software libre. Creamos materiales y contenidos didácticos bajo licencia [CERN-OHL-W](#) e [CC BY 4.0](#).

Creamos Echidna Educación como una asociación sin ánimo de lucro que tiene como objetivos: Promover el aprendizaje de programación y robótica mediante el uso de herramientas de código abierto. Facilitar el acceso a la programación de dispositivos físicos a través de creación de electrónica de acceso libre “hardware open source”. Diseñado a partir de las necesidades y experiencias del aula. Un tiempo después llegaron dos docentes más, configurando el actual equipo @EchidnaSteam



EchidnaBlack es una placa autónoma compatible con Arduino Nano/ Arduino UNO...

- Sensores integrados: Pulsadores, joystick, acelerómetro, luz, micrófono, temperatura.
- Actuadores integrados: LEDs, LED RGB, Sonido
- Flexible: entradas e salidas para complementos
- Ocho (8) entradas tipo Makey Makey
- Conexión para BlueTooth

Cuenta con los certificados:



Certificado CE: ISETC.000520201231

Electromagnetic Compatibility

2014/30/EU

Directiva CE 2002/95/EC

EN 55032:2015+A11:2020, EN 55035:2017

The Open Source Hardware Association

ES000010





Conociendo la placa.

Vamos conocer las partes que componen esta placa de entrenamiento, y aprenderemos a programar los actuadores integrados y poder leer los sensores, mediante sencillo ejemplos, como controlar una luz, leer a temperatura para activar una salida en función de la misma, y otros ejemplos que iremos viendo a lo largo del manual.

En las explicaciones de los componentes encontraremos:

Texto en cursiva indicando la conexión de componente.

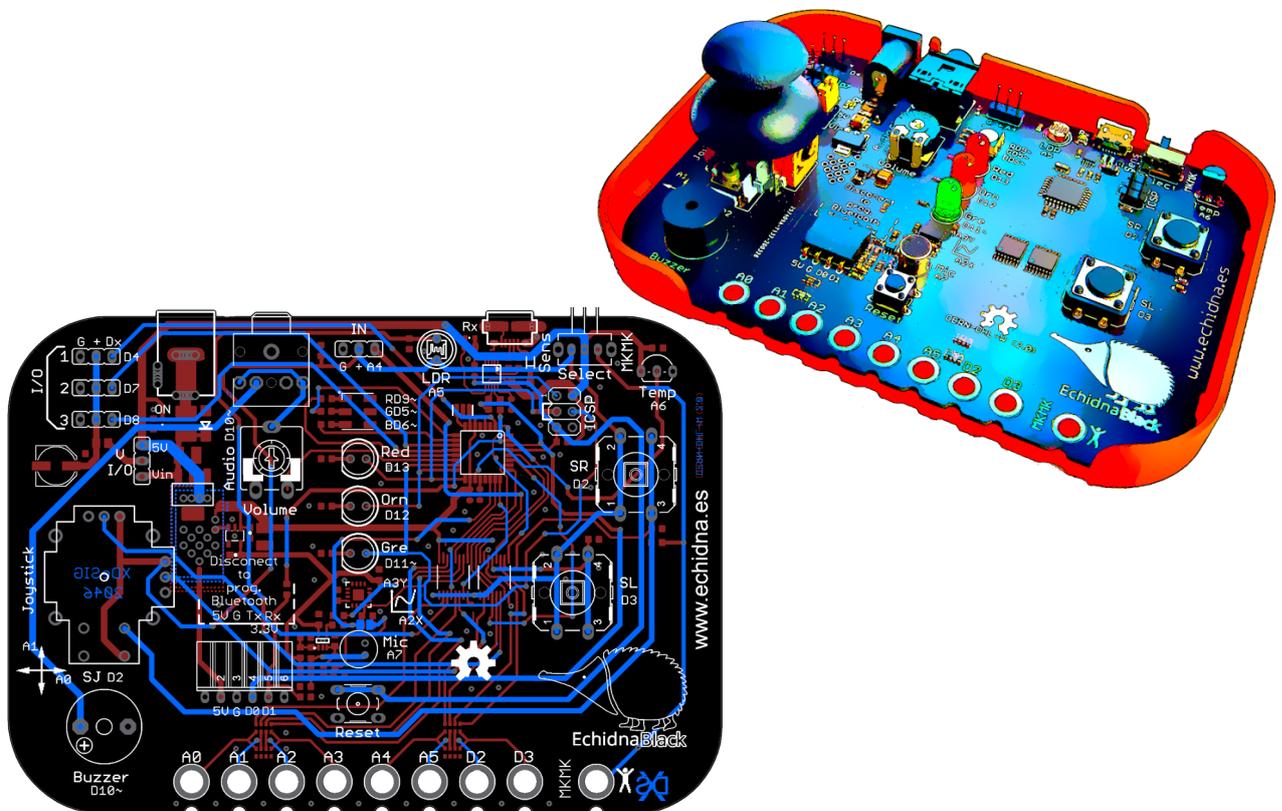
D2 "SL" : Entrada digital

1. Texto ejemplarizando la instrucción de lectura/escritura del componente.

```
boolean Pul_R = digitalRead (SR);
```

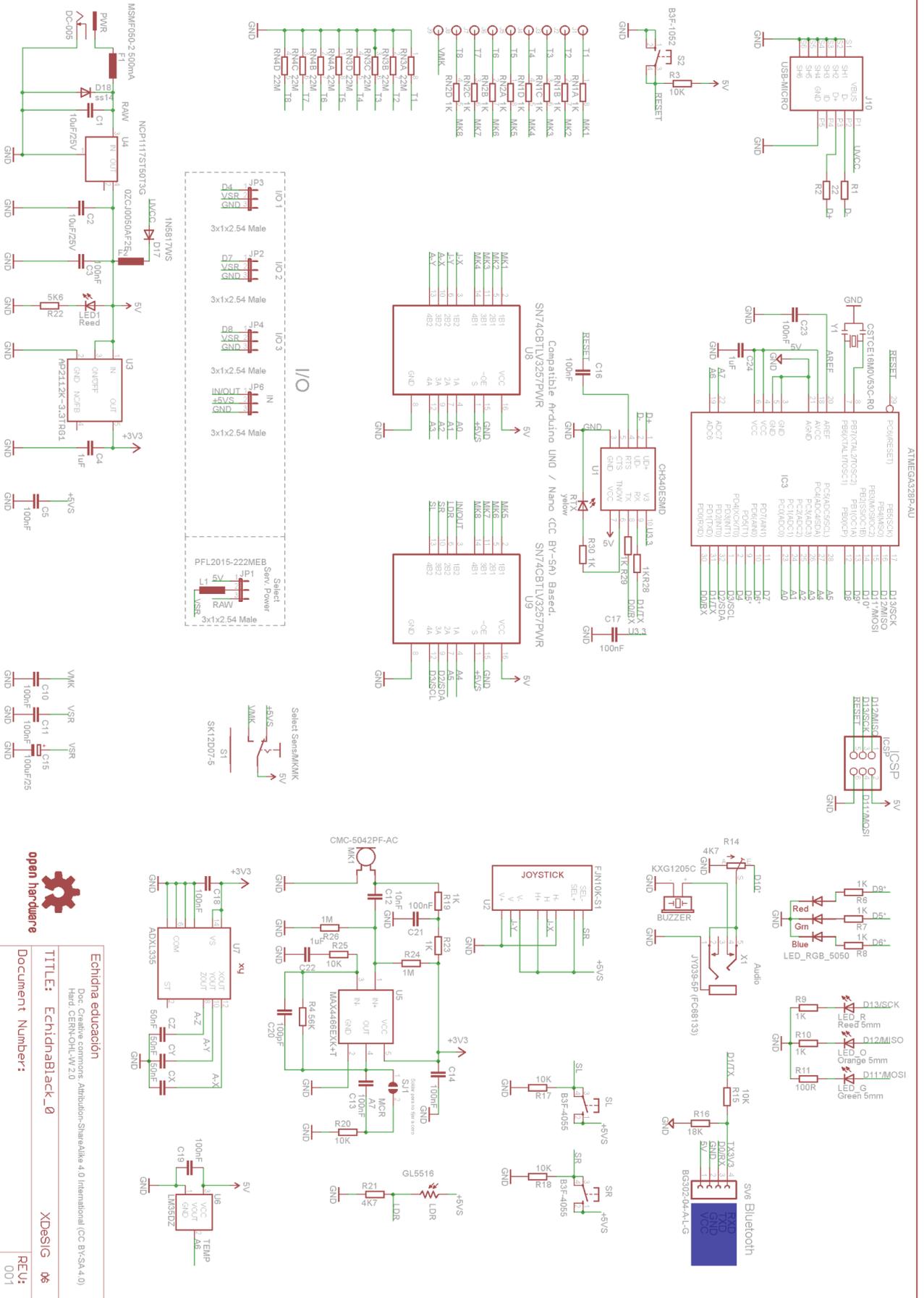
Texto subrayado, enlace a datos en internet.

<https://echidna.es/hardware/echidnablack/>





Makey Makey (CC BY-SA) variant.



Echidna education
 Doc: Creative commons: Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)
 Hard: CERN-OHL-W/2.0

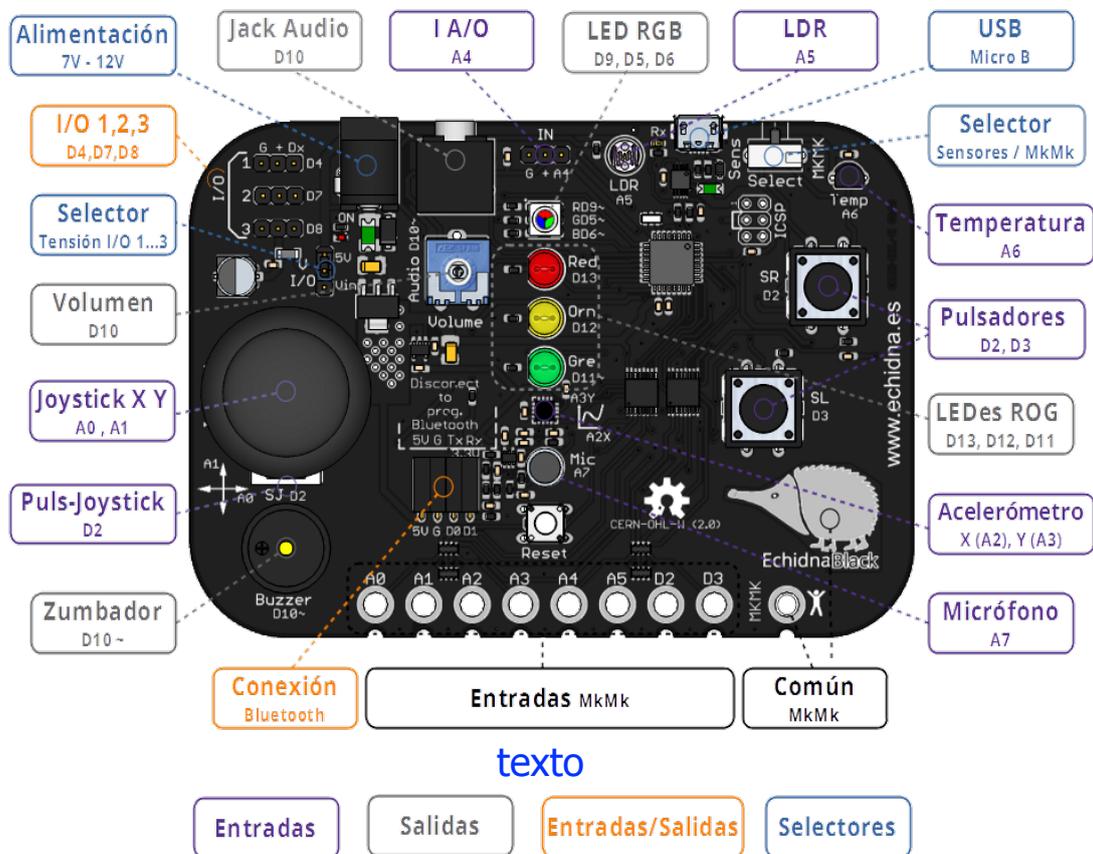
Document Number: XDeSIG_06
REU: 001

Date: 16/06/2022 21:36:58

Sheet: 1/1

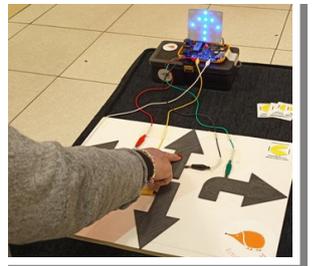


Entradas/ salidas



Cuenta con múltiples entradas e salidas, algunas son solo entradas o salidas, pero también contamos con cuatro conexiones que pueden funcionar como entradas y salidas, dependiendo de la configuración que hagamos en nuestro programa.

Tenemos entradas tipo MkMk, que se activan cuando pasa una pequeña corriente entre el común MkMk (+5V) y cualquiera de las entradas MkMk; tocando con los dedos, conectando una gominola, una fruta, plastilina conductora o pintando teclas con lápiz...



El cerebro es el microcontrolador ATmega328P, cuenta con arquitectura RISC, pudiendo realizar un millón de operaciones cada segundo, tiene memoria flash de 32KB, memoria RAM de 2Kb y 1KB de EEPROM. Disponemos de 23 **líneas** de entrada/salida de las cuales 6 también son entradas analógicas de 10 bits. Un canal de comunicación serie Tx/RxD. **Non** te preocupes de memorizar a que pin del microcontrolador está conectada cada cosa, la "chuleta" está impresa en la propia placa ;-).

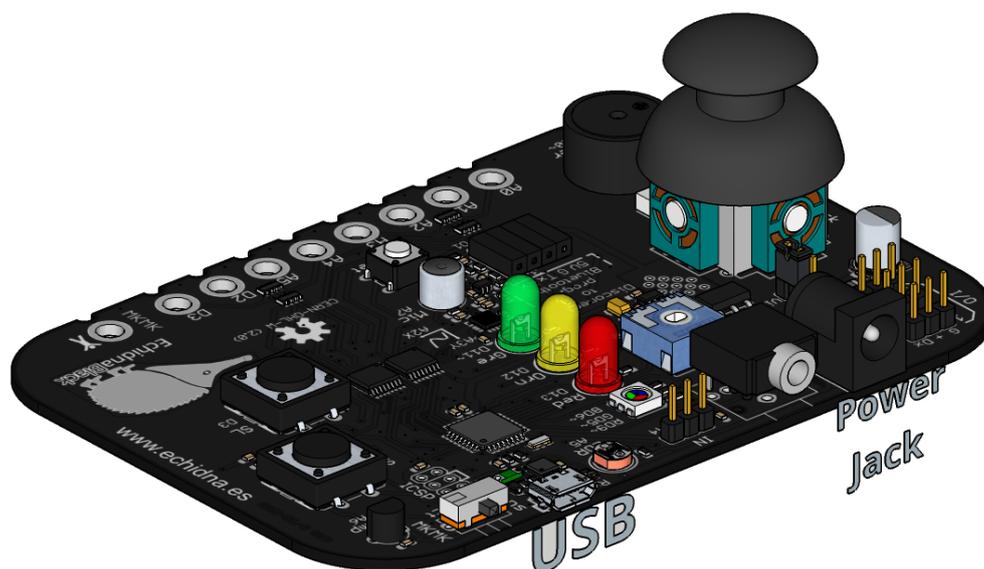


Alimentación de la placa

Disponemos de dos formas de alimentación:

1. Mediante la conexión USB. Esta forma es válida para la mayoría de las funciones, toda la placa recibe tensión regulada de 5V proporcionada por el cable USB, generalmente con un límite de 500mA EchidnaBlack cuenta con fusible rearmable protegiendo nuestro ordenador de los posibles sobre-consumos de nuestros experimentos.
2. Mediante el Jack de alimentación. Todas las partes reciben tensión regulada internamente de 5V e 1A máx. Nos da la posibilidad de seleccionar V_{in} para alimentar los **pinos I/O con la tensión** y potencia que proporciona el **alimentador externo**, esta conexión cuenta con fusible rearmable de protección (ojo no superar 1A de corriente constante).

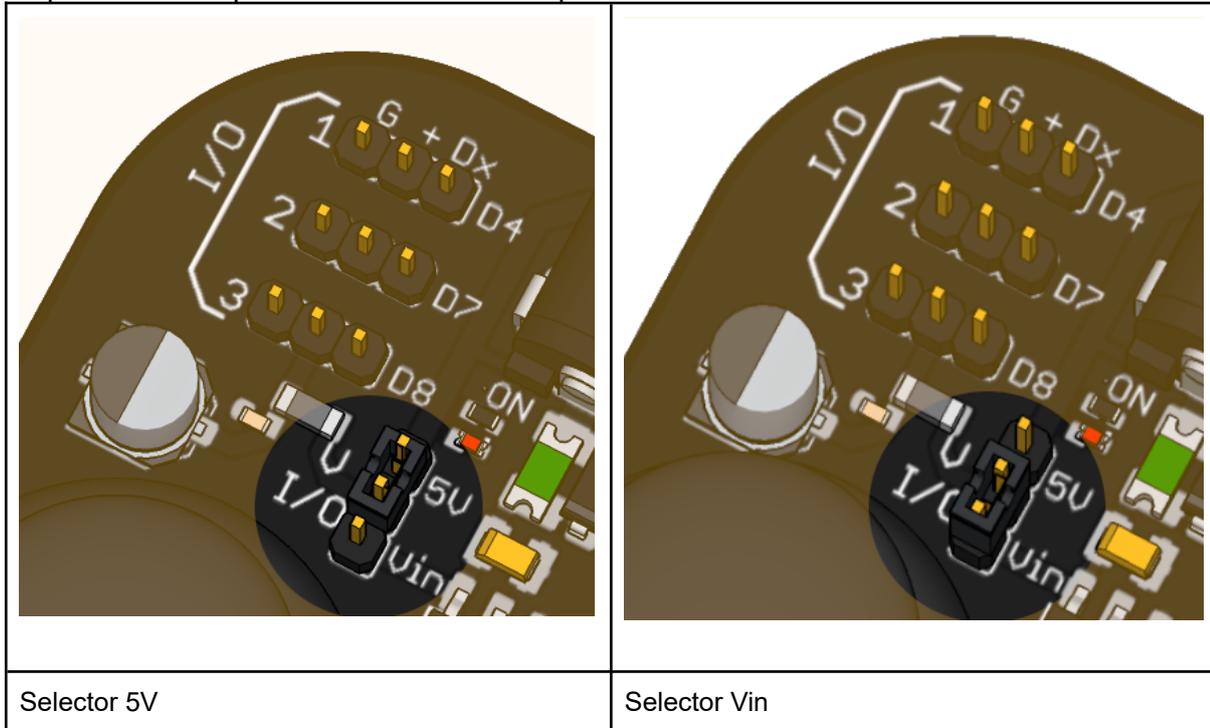
Interlineado





Selector de Alimentación I/O

El puente “Jumper” de alimentación permite seleccionar la alimentación de las I/O.



Selector alimentación 5V:

En el caso de querer alimentar las I/O desde los 5 Volts procedentes del regulador integrado, el selector tiene que estar colocado como indica la imagen de la izquierda. Aconsejado para sensores externos que necesiten una tensión estabilizada.

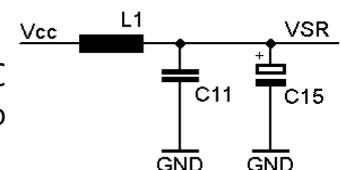
¡No utilizar la alimentación 5V cuando los dispositivos conectados consuman más de 500 mA!. De lo contrario sobrepasaríamos la capacidad del regulador.

Si es necesario podemos alimentarlo mediante una alimentación suplementaria, manteniendo el GND común entre ambos equipos.

Selector alimentación Vin (Alimentación externa):

Aconsejado para alimentar servos o otros dispositivos conectados a I/O que consuman más de 500 mA con alimentación externa en el Jack de alimentación.

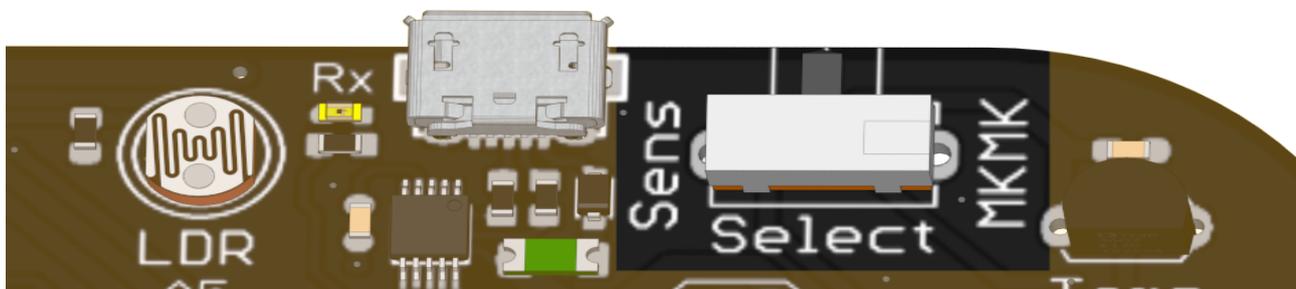
La alimentación en los pines I/O “V_{SR}” cuenta con filtro L-C para evitar que lleguen parásitos eléctricos de los motores o del resto de componentes.





Selector Modo Sensores/Modo MkMk

Mediante el conmutador cambiamos entre **Modo Sensores** y **Modo Mk Mk**.



Modo sensores

En el modo sensores tenemos activos todos los componentes de la placa exceptuando las conexiones MkMk.





Modo MkMk

En este modo tenemos activos:

Conexiones MkMk.

Sensor de temperatura.

Micrófono.

Salidas:

LED RGB, LEDes ROG, I/O 1.. I/O3

Aqui faltan
guiones..?

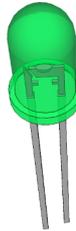




Descripción de los componentes:

Salidas:

LED.



Acrónimo de “Light Emitting Diode”, basado en el fenómeno de la electroluminiscencia. Se usan como pilotos (indicadores) y como fuente de iluminación.

Al aplicar tensión en los pines del LED, este emite luz, podemos controlar el encendido y apagado programando  pin como salida digital:

“1” enciende, “0” apaga, si está conectado a una salida PWM “~” podemos ajustar el brillo.

Todos los LEDes se conectan con una resistencia serie “Rs” para controlar la tensión/corriente aplicada. ([ejemplo de calculo de la resistencia para un LED pág. 138](#)).

Tenemos tres LEDes conectados en las siguientes salidas del microcontrolador:

D11~ “Gre” (Verde): Salida Digital e PWM.

D12 “Orn” (Naranja): Salida Digital.

D13 “Red” (Rojo): Salida Digital.



Para manejar los LEDes podemos hacerlo de dos formas, dependiendo de la salida en la que esta conectado:

Digital: Podemos modificar el estado enviando “0” apagado y “1” encendido.

```
digitalWrite(Red, 1); o digitalWrite(Red, HIGH); //Encendido
```

```
digitalWrite(Red, 0); o digitalWrite(Red, LOW); //Apagado
```

PWM: En la salida D11~ “Gre” podemos enviar el valor de intensidad luminosa entre 0 apagado y 255 encendido completo.

```
analogWrite(Gre,128); //Mitad de iluminación
```

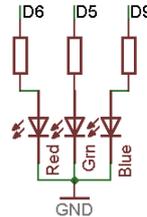
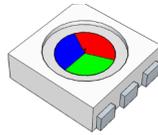
Hoja de datos:

<http://www.sparkfun.com/datasheets/Components/LED/COM-09590-YSL-R531R3D-D2.pdf>



LED RGB.

Acrónimo de Light Emitting Diode y Red Green Blue, son tres LEDs: rojo, verde y azul en la misma cápsula.



Podemos controlar el LED RGB digitalmente, encendiendo o apagando cada LED y ajustar la luminosidad mediante PWM.

Control digital: podemos encender y apagar cada **un** de los LEDs que forman el RGB, esta forma permite formar $2^3=8$ colores diferentes.

Control “analógico” PWM: podemos controlar el brillo de cada LED, variando el tiempo que está encendido y apagado, lo que nos da la sensación de variación luminosa desde 0 a 255 (8 bits), esto permite realizar $2^8*2^8*2^8= 16,7$ millones de colores.

Como los LEDs anteriores, los conectamos con una resistencia serie “Rs” para controlar a tensión/corriente aplicada.

Este RGB esta conectado a salidas “~” que nos permiten ter control PWM:

D9~ “RGB_R” (rojo): Digital y PWM

D5~ “RGB_G” (verde): Digital y PWM

D6~ “RGB_B” (azul): Digital y PWM

La programación es la misma que usamos en los otros LEDs:

Digitalmente (máximo 8 colores):

```
digitalWrite(RGB_R, 1); o digitalWrite(RGB_G, HIGH);  
digitalWrite(RGB_G, 1); o digitalWrite(RGB_G, HIGH);  
digitalWrite(RGB_B, 0); o digitalWrite(RGB_G, LOW);
```

Analógicamente (máximo 16 millones de colores):

```
analogWrite(RGB_R, 128);  
analogWrite(RGB_G, 128);  
analogWrite(RGB_B, 0);
```

Hoja de datos:

http://www.farnell.com/datasheets/2003905.pdf?_ga=2.248743815.1083922617.1497294090-1303058009.1492504873



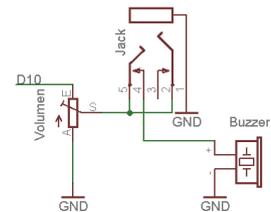
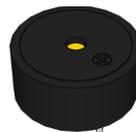
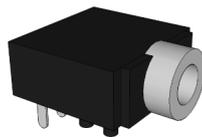
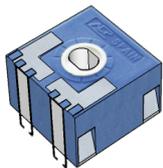


Sonido.

Disponemos de dos salidas para reproducir sonido: el zumbador y el jack, donde podemos conectar auriculares o altavoces autoamplificados.

Al conectar un conector (3,6 mm) en el jack, se desconecta automáticamente el zumbador.

Podemos ajustar la amplitud del sonido mediante el potenciómetro de volumen.



En la salida D10~ podemos reproducir señales entre 31 Hz y 20 KHz.

El zumbador, al ser excitado por una señal con una frecuencia determinada, vibra y reproduce un sonido.

Es importante tener en cuenta que su frecuencia central es de 2,3 KHz, por lo que si nos alejamos mucho de esta frecuencia, no sonará con calidad.

Si conectamos un equipo de sonido externo en la salida del jack, podremos reproducir las señales con mayor calidad.

 Si conectamos unos auriculares a alto volumen, se puede alcanzar el umbral del dolor en los oídos y dañar los auriculares.

 Si reducimos demasiado el nivel de volumen del zumbador, puede no resultar audible.

D10~ “Buzz”: Salida PWM (modulación por ancho de pulso).

Se activa como una salida PWM con un valor entre 0 y 255 que modula la frecuencia del sonido.

Disponemos de dos instrucciones que pueden realizar esta tarea.

```
analogWrite(Buzzer, 128);
tone(Buzzer, 2000, 500);
```

Hoja de datos:

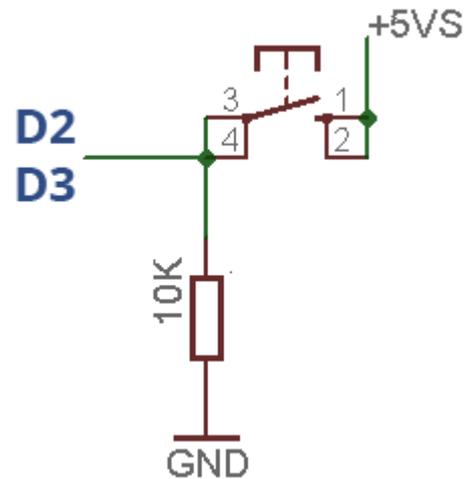
<https://docs.google.com/file/d/0B1T3xR6vq4KRAG0yTjBBXzB3Y2M/edit>



Entradas (Sensores)

Pulsadores.

Un "pulsador" es un componente electromecánico que permite abrir o cerrar un circuito y cuenta con un único estado estable.



Conectado con una resistencia a masa, forma un circuito denominado pull-down, que proporciona un "0" (0V) cuando no se pulsa y un "1" (5V) al pulsarlo.

Están conectados a los siguientes pines:

D2 "SL": Entrada digital

D3 "SR": Entrada digital

Para leer los "pulsadores", utilizamos la instrucción de lectura digital

```
boolean Pul_R = digitalRead (SR);
```

```
boolean Pul_L = digitalRead (SL);
```

Hoja de datos:

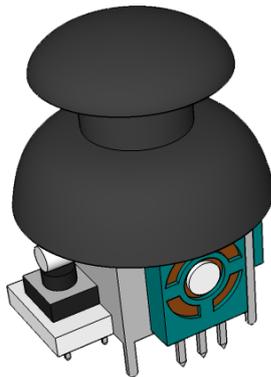
https://omronfs.omron.com/en_US/ecb/products/pdf/en-b3f.pdf

Nota: `boolean`: Ver página 52 "Tipos de datos"

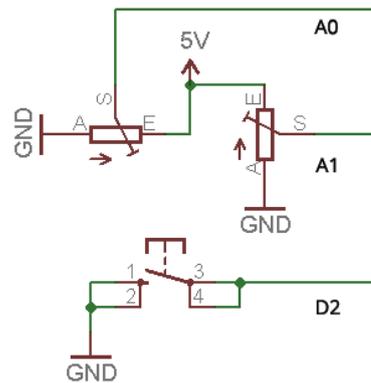


Palanca de juegos “Joystick”

Es un mando formado por dos potenciómetros, uno para el eje X y otro para el eje Y. Este modelo incluye un "pulsador" adicional.



Imaxe Alfredo B.



Permite convertir el movimiento del mando en un valor de resistencia proporcional. Al conectarlo en modo divisor de tensión, obtenemos una tensión proporcional de 0 a 5V en la salida de cada eje (X, Y).

En la posición de reposo, los valores rondan los 2,5V (520 en lectura analógica).

El "pulsador" del Joystick está conectado al mismo pin que el "pulsador" SR.

A0 “Joy_X”: Entrada analógica

A1 “Joy_Y”: Entrada analógica

D2 “SJ” = “SR”: Entrada digital

Lectura y almacenamiento de los valores del Joystick:

```
int Valor_X = analogRead(A0);
int Valor_Y = analogRead(A1);
boolean Pul_R = digitalRead (D2);
```

Hoja de datos:

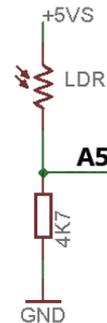
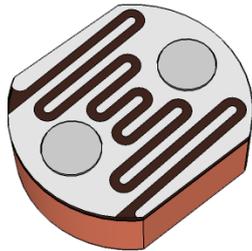
<http://www.polyshine.cn/uploadFile/FJN10K-2014-01-03-07-24-37.pdf>

Nota: [int](#): Ver página 52 “Tipos de datos”



Sensor de Luz: LDR.

LDR, acrónimo de "Light Dependent Resistor" (Resistencia Dependiente de la Luz), es una resistencia cuyo valor depende de la cantidad de luz que incide sobre ella. Generalmente, se fabrica con sulfuro de cadmio.



La LDR proporciona valores de resistencia altos (algunos $M\Omega$) con poca luz y valores bajos con mucha luz. Al conectarla con una resistencia en serie, se forma un divisor de tensión que invierte la lógica de funcionamiento. De este modo, con mucha luz proporciona valores altos de tensión ($4,9V = 999$ en lectura analógica)* y valores bajos con poca luz ($0,0V = 000$).

No solo es sensible a la luz visible, su respuesta óptica abarca desde menos de 400nm (ultravioleta) hasta 800nm (infrarrojos), siendo más sensible en torno a los 550nm (verde-amarillo)

* Debido a las tolerancias de los componentes estos valores pueden ser distintos.

A5 "LDR": Entrada analógica

Lectura y almacenamiento de los valores de la LDR:

```
int Valor_LDR = analogRead(A5);
```

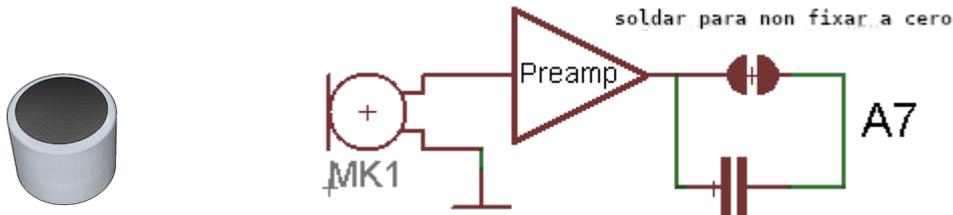
Hoja de datos:

<http://akizukidenshi.com/download/ds/senba/GL55%20Series%20Photoresistor.pdf>



Micrófono.

Es un transductor acústico-eléctrico piezoeléctrico que entrega una señal eléctrica con características similares al sonido que recibe.



Varía desde cero hasta un valor de aproximadamente 3V, como se muestra en la figura 1. Si realizamos una pequeña soldadura en el selector de la cara inferior (entre las patas delanteras de la fig. Echidna), el señal que nos proporciona ahora está centrada en 1,75V = 360 en la señal analógica, como se muestra en la figura 3.

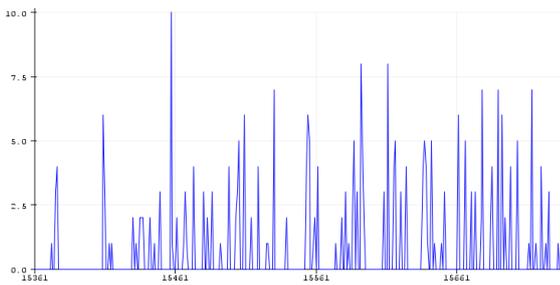


Fig. 1

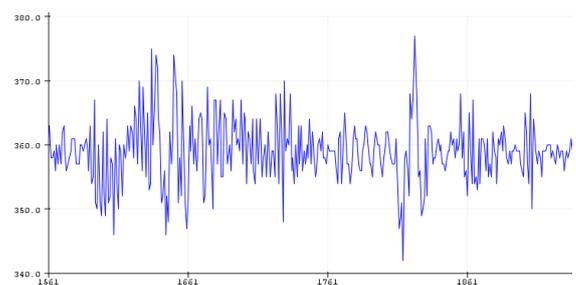


Fig. 3

A7 "Mic": Entrada Analógica

Al igual que en el caso anterior, podemos usar la referencia interna de 1,1V para mejorar la sensibilidad.

```
analogReference(INTERNAL);  
int Valor_Mic = analogRead(A7);
```

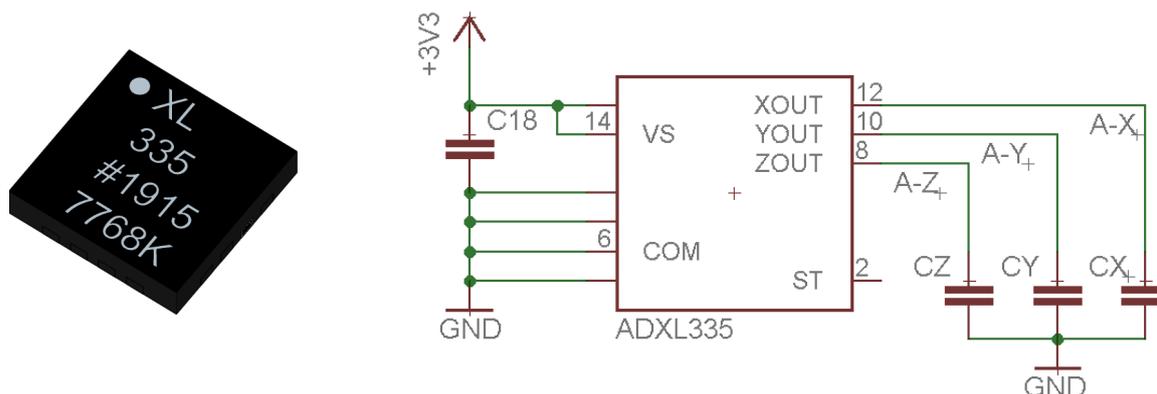
Hoja de datos:

<https://www.mouser.es/datasheet/2/670/cmc-5042pf-ac-1776337.pdf>



Acelerómetro.

Sensor de aceleraciones, está basado en condensadores diferenciales dentro de una estructura micro-mecanizada. Proporciona una tensión dependiente de la aceleración en cada eje.



Mide la aceleración en los ejes (X, Y y Z) en un rango de $\pm 3g$. En la posición de reposo proporciona un valor de $1,75V = 359$ (de lectura analógica), y en los extremos los valores son $1,39V=285$ y $2,10V = 428^*$.

En EchidnaBlack, la salida Z no está conectada.

* Debido a las tolerancias de los componentes estos valores **poden** ser distintos.

A2 "Ace_X": Entrada analógica

A3 "Ace_Y": Entrada analógica

Lectura y almacenamiento de los valores del sensor:

```
int Valor_Ace_X = analogRead(A2);
int Valor_Ace_y = analogRead(A3);
```

Hoja de datos:

<https://www.nxp.com/docs/en/data-sheet/MMA7361L.pdf>



Entradas (Modo MkMk).

Disponemos de 8 conexiones MkMk. Cada conexión es parte de un divisor de tensión con una resistencia muy elevada, por lo que al conectar un elemento que conduzca electricidad, aunque sea tenuemente, al cerrar el circuito con el punto común, la tensión de la entrada sube y puede ser detectada. La tensión umbral de detección depende de cada material/persona. (El logo Echidna también se comporta como Común).



Podemos leer las entradas MkMk como si se tratara de una entrada digital, pero también podemos leer las entradas A0..A5 de forma analógica y compararlas con un valor umbral para tomar decisiones..

A0 MkMk0: Entrada analógica

A1 MkMk1: Entrada analógica

A2 MkMk2: Entrada analógica

A3 MkMk3: Entrada analógica

A4 MkMk4: Entrada analógica

A5 MkMk5: Entrada analógica

D2 MkMk6: Entrada digital

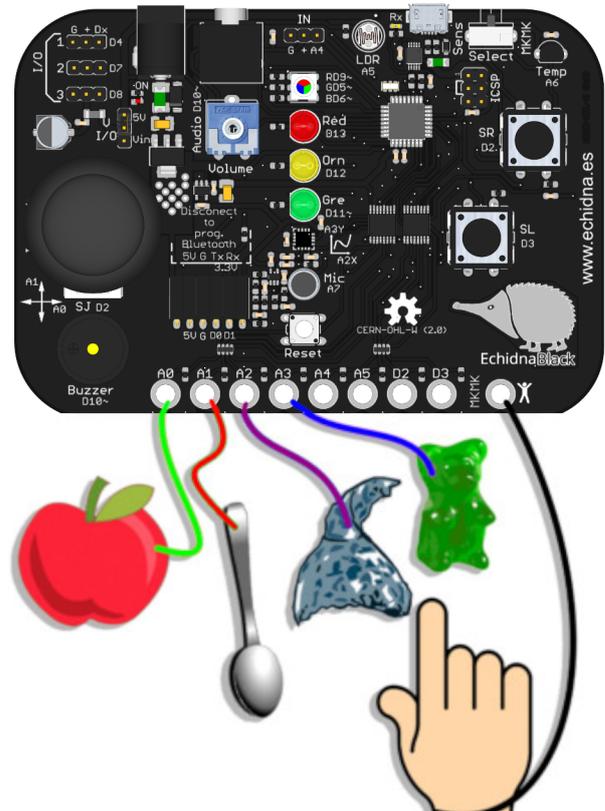
D3 MkMk7: Entrada digital

Lectura digital:

```
int valorMkMk0 = digitalRead(A0);
```

Lectura analógica:

```
int umbral = 600;  
int valorMkMk0 = analogRead(A0);  
if (valorMkMk0 >= umbral){  
  // aquí lo que quieras hacer  
}
```





Pines de libre disposición.

Contamos con cuatro entradas/salidas de libre disposición:

Tres digitales (D4, D7, D8) y una entrada analógica o digital, salida digital (A4).



Cada I/O e IN cuenta con tres pines:

G = GND (negativo, común).

+ = +V (positivo)

Para I/O ver "[Selector de Alimentación](#)" / Para IN +5Vcc

Dx = entrada / salida

A4 = Entrada, salida analógica o digital.

Podemos conectar diferentes dispositivos, como servos, magnetómetros, higrómetros, pantallas, entre otros.

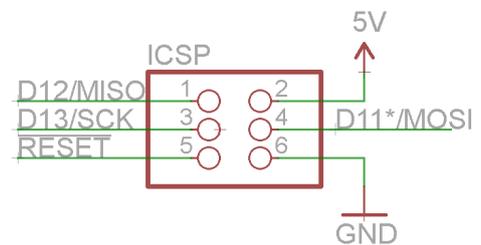
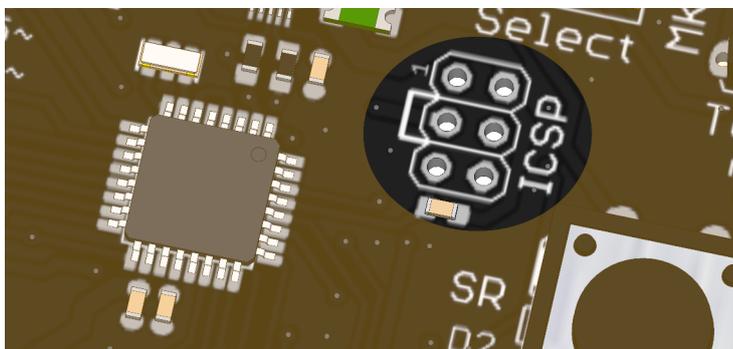
La disposición de pines se eligió para tener compatibilidad con muchos dispositivos, pero no todos son compatibles pin a pin. Es necesario comprobar la disposición de las conexiones de tus dispositivos para evitar dañarlos.



Conector ICSP “In-Circuit Serial Programming”.

Por defecto, EchidnaBlack viene preprogramada con el cargador de arranque "bootloader" OPTIBOOT (GPLv2 WRT), que nos permite enviar los programas vía USB, y es compatible con todos los IDE de Arduino (Arduino Nano).

Mediante el conector ICSP podemos reprogramar el microcontrolador, teniendo acceso a la memoria de programa (flash) sin necesidad de utilizar el bootloader. También podemos cambiar el cargador de arranque o instalar otro firmware. (Para el trabajo normal, no es necesario utilizarlo.)



MISO (D12) - Master In Slave Out

MOSI (D11) - Master Out Slave In

SCK (D13) - Signal Clock

RESET - Reinicio del microcontrolador.

5V (VCC) - Voltaje

GND - Tierra (común)

Puede ser que alguna vez (rara) no se cargue correctamente el programa que queremos y el microcontrolador se quede "Briqueado" (bloqueado) y ya no podamos programarlo. En ese caso, podemos "Desbriquearlo" volviendo a cargar el bootloader mediante el conector ICSP.

[Ver página 139 “desenladrillar”](#)



Programación (hablemos con EchidnaBlack)

Tenemos varias formas para programarla:

Mediante entornos gráficos, es necesario cargar algún programa específico al estilo Firmata.

- Echidna-Scratch “<https://scratch.echidna.es>”



- Snap4Arduino “<https://snap4arduino.rocks>”
- ArduinoBlocks “<http://www.arduinoblocks.com>”
- Bitbloq “<https://bitbloq.cc>”
- Makeblock “<https://mblock.makeblock.com/en-us>”
- ...



- Mediante líneas de código
 - ASM
 - C
 - Wiring “<http://wiring.org.co>”
 - Arduino (C++ modificado) es el que vamos a usar en este manual.



Para escribir los programas y pasárselos a EchidnaBlack necesitamos usar una diversidad de programas: editor, compilador y programador. Sin embargo, contamos con programas que ya integran todo lo necesario para poder programar cómodamente.

Se llaman entornos de desarrollo integrado (Integrated Development Environment, IDE), y entre otros, tenemos

- [Arduino IDE 2.0](#)
- [PlatformIO](#)
- [Eclipse Arduino IDE](#)
- [Codebender](#)
- [ArduinoDroid](#)
- [Programino](#)

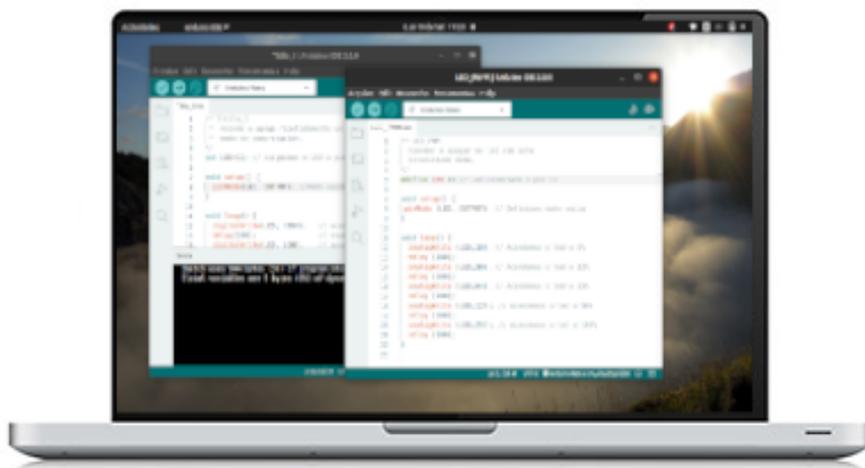


PlatformIO, en mi opinión, es la más recomendable de las IDE, pero en este manual usaremos el **Arduino IDE 2.x.x.** más extendido. Te aconsejo que pruebes otras IDEs y uses la que te resulte más cómoda (ojo, algunas son de pago).

IDE de Arduino

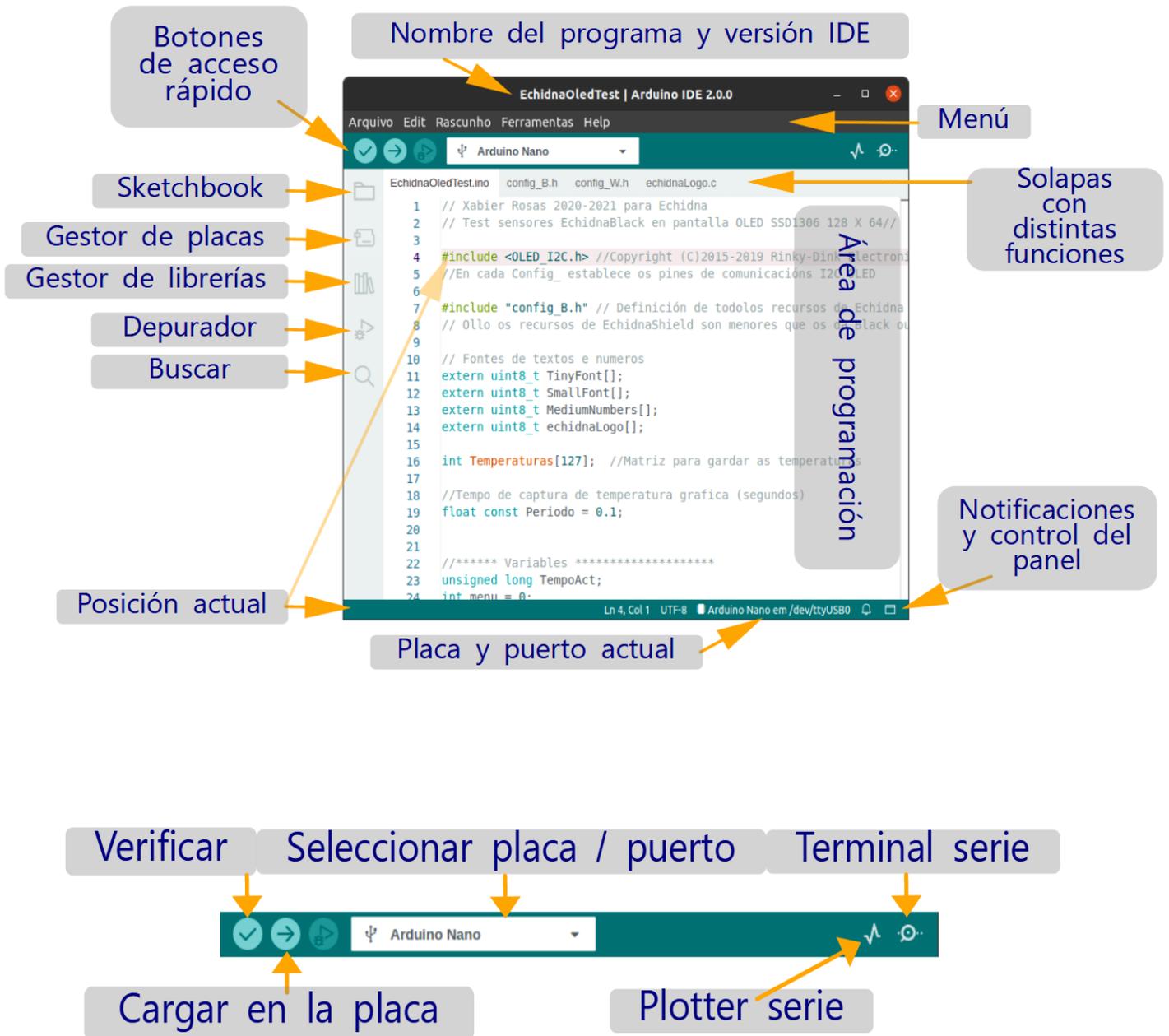
Descarga e instala en el ordenador (la instalación variará dependiendo del sistema operativo que tengas). “[Enlace a arduino](#)”

En Internet tenemos muchos manuales de instalación que puedes consultar.





Una vez instalado el IDE de Arduino, vamos a conocerlo:





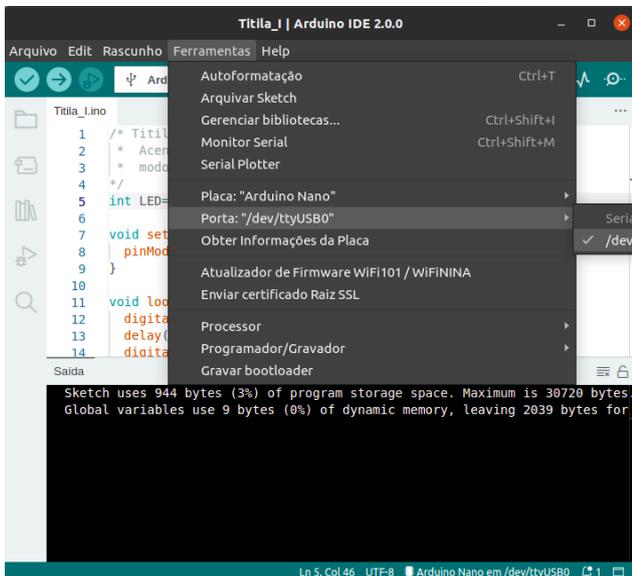
Para subir los programas a nuestra EchidnaBlack, tenemos que seleccionar la placa Arduino Nano y el puerto serie correspondiente, que puede variar según la conexión USB que utilicemos.

Usamos el chip de comunicaciones, CH341. "[Datos y driver](#)"

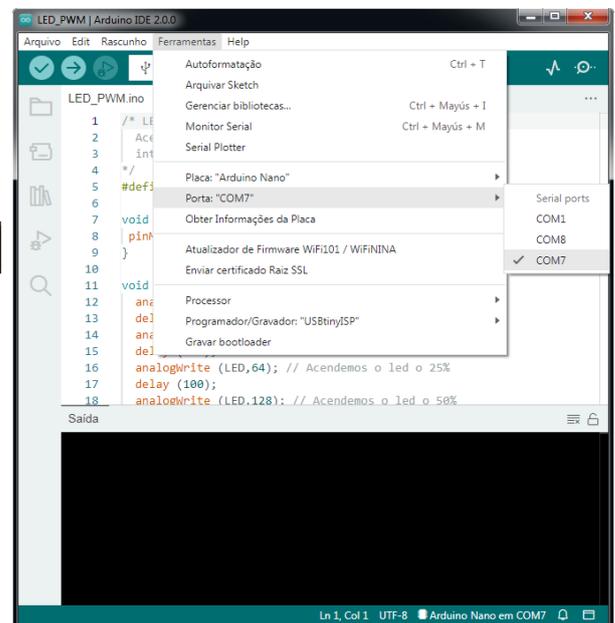
En los sistemas operativos GNU Linux, no es necesario instalar ningún controlador "Driver" USB para comunicarnos con EchidnaBlack.

En MAC OS, a veces no se reconoce el dispositivo o aparece un mensaje de **Kernel Panic**. Para solucionarlo, descargamos la versión actualizada del "driver" [CH341](#) (se pedirá reinicio) y debemos permitir su ejecución en "permitir aplicaciones descargadas".

En Windows también tenemos que instalar el controlador. [CH341](#)



GNU LINUX



Windows



Estructura de un programa “Sketch” de Arduino:

Todos los sketch de Arduino tienen dos funciones básicas:

1. void setup()

Es una función que se ejecuta una sola vez, cuando se inicia el sketch. Se utiliza para inicializar comunicaciones, ajustar el modo de los pines, iniciar el uso de dispositivos, librerías, etc.

2. void loop()

1. Hace precisamente lo que su nombre sugiere: es un bucle que ejecuta consecutivamente las instrucciones del programa.

```
Titila_I | Arduino IDE 2.3.4
Archivo Editar Sketch Herramientas Ayuda
Arduino Nano
Titila_I.ino
1  /* Titila_I
2  * Enciende y apaga repetidamente un LED,
3  * A modo de comprobación.
4  */
5  #define LED 12 // Asignamos el LED al pin D12 (Led naranja)
6
7  void setup() {
8  | pinMode(LED, OUTPUT); //Modo salida
9  }
10 void loop() {
11 | digitalWrite(LED, HIGH); // Enciende el LED HIGH = "1" = +5V
12 | delay(100); // Espera 100 milisegundos
13 | digitalWrite(LED, LOW); // Apaga el LED LOW = "0" = 0V
14 | delay(500); // Espera 500 milisegundos
15 }
16 int LED=12;
```

Comentar el código es una buena práctica. Nos ayuda a elaborar el programa y entenderlo pasado un tiempo. Los comentarios no afectarán la operación normal del programa, no se compilan, son solo para nosotros. La línea que escribamos tras



"//" es un comentario. Si necesitamos escribir más líneas, podemos establecer un bloque de comentario comenzando por `/*` y terminando por `*/`.

Cuando escribimos un programa, necesitamos empaquetar un conjunto de instrucciones (funciones, bucles, etc.), las cuales se colocan entre llaves `{}`. Se utiliza `;` para indicarle al compilador que una instrucción ha finalizado. Las entradas y salidas de los microcontroladores son configurables y, normalmente, digitales (solo entienden ceros y unos):

`"0"` Cero = $-0,5V$ a $0,3 \cdot V_{cc}$ `"1"` Uno = $0,7 \cdot V_{cc}$ a $V_{cc} + 0,5V$

Algunas entradas y salidas pueden ser analógicas (entienden valores que van de $0V$ a V_{cc}). En el ATmega328P que tiene la EchidnaBlack, disponemos de ocho (8) entradas analógicas y ninguna salida analógica, pero sí contamos con seis (6) salidas PWM de 8 bits.

Para ajustar cómo se van a comportar las salidas usamos:

```
pinMode(pin, modo);
```

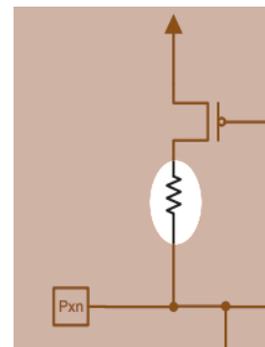
pin: especifica el número de pin (0..13) (A0..A7)

modo: configura el modo de funcionamiento, que en Arduino es:

`OUTPUT`: Salida.

`INPUT`: Entrada.

`INPUT_PULLUP`: Entrada con una resistencia de $10K\Omega$ conectada a $+V_{CC}$, así no tenemos que conectar una resistencia externa.



Para leer una entrada:

```
digitalRead(pin);
```

Lectura digital, devuelve un "1" o "0", pines de 0..13 y A0..A5.

```
analogRead(Ax);
```

Lectura analógica, devuelve un valor de 10 bits (0..1023) pines A0..A7



Escribir en una salida:

```
digitalWrite(pin, valor);
```

Salida digital pines 0..13 y A0..A5, valor puede ser un 0=LOW o 1=HIGH.

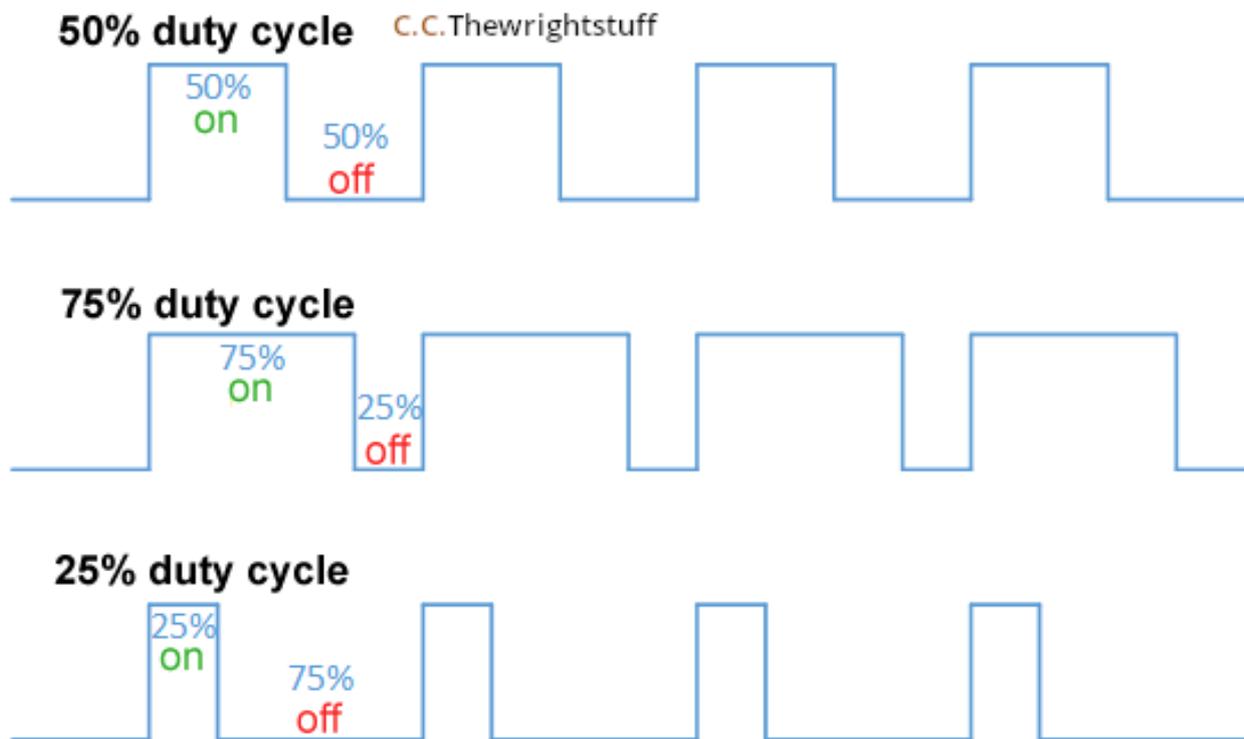
```
analogWrite(pin, valor);
```

Salida PWM pines 5, 6, 9, 10, 11, valor de 8 bits ($2^8 = 256$) 0 a 255.

Si escribimos en la salida PWM = 0 es lo mismo que apagado.

Si el valor es 128, la salida tendrá la mitad del tiempo en 0 y la otra mitad en 1.

La relación entre el impulso y el intervalo la llamamos ciclo de trabajo “duty cycle”, permite controlar la iluminación de un LED, la velocidad de giro de un motor...



En este manual, vamos a aprender a programar paso a paso, comenzando con ejemplos simples que nos permitirán comprender los conceptos básicos. A lo largo del proceso, practicaremos con los programas más sencillos y luego avanzaremos a tareas un poco más complejas.

Cada ejemplo será una oportunidad para experimentar y mejorar nuestras habilidades.

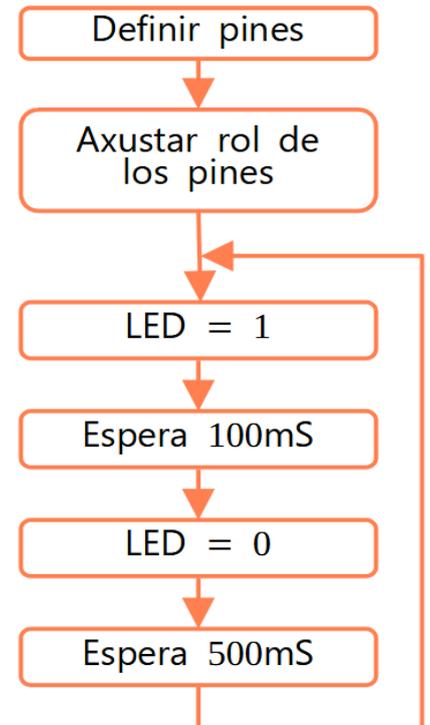


Sketch "Titila_1".

Ya podemos hacer un pequeño programa que encienda y apague un LED. Iniciamos el IDE de Arduino y escribimos el siguiente programa:

```
/* Titila_1
 * Enciende y apaga repetidamente un LED,
 * A modo de comprobación.
 */
#define LED 12 // Asignamos el LED al pin D12 (Led
naranja)

void setup() {
  pinMode(LED, OUTPUT); //Modo salida
}
void loop() {
  digitalWrite(LED, HIGH); // Enciende el LED HIGH = "1" =
+5V
  delay(100); // Espera 100 milisegundos
  digitalWrite(LED, LOW); // Apaga el LED LOW = "0" = 0V
  delay(500); // Espera 500 milisegundos
}
```



Análisis:

La línea `#define LED 12`, asigna el valor constante 12 a LED. Las constantes definidas de esta manera no ocupan espacio en memoria, ya que el compilador sustituirá la referencia LED por el valor 12 en el momento de la compilación. Así, si quisiéramos cambiar el pin asignado al LED, simplemente tendríamos que modificar el valor en la cabecera del programa, y no sería necesario hacerlo en cada línea donde se use, lo que facilita la gestión del código y lo hace más legible.

`delay(500);`

Esta es una declaración de retraso, lo que significa que el LED puede encenderse o apagarse durante 100 milisegundos. Fíjate que hemos puesto un retraso de **500 milisegundos** después de apagar el LED, lo que hace que el apagado dure más que el encendido. Esto es útil si queremos que el LED esté encendido solo un breve momento y apagado durante un tiempo más largo.



Subir el programa.

Ahora podemos verificar que la sintaxis es correcta, haz clic en "Verificar". También puedes hacer clic en el botón "Subir", que también verificará la sintaxis. Si no tenemos errores, el programa se subirá a EchidnaBlack.

Asegúrate de tener EchidnaBlack conectada al ordenador, selecciona la placa "Arduino Nano", el procesador "ATmega328" y el puerto de comunicación asignado.

```
Titila_1.ino
2  * Enciende y apaga repetidamente un LED,
3  * A modo de comprobación.
4  */
5  #define LED 12 // Asignamos el LED al pin D12 (Led naranja)
6
7  void setup() {
8    pinMode(LED, OUTPUT); //Modo salida
9  }
10 void loop() {
11   digitalWrite(LED, HIGH); // Enciende el LED HIGH = "1" = +5V
12   delay(100); // Espera 100 milisegundos
13   digitalWrite(LED, LOW); // Apaga el LED LOW = "0" = 0V
14   delay(500); // Espera 500 milisegundos
15 }
```

Salida

El Sketch usa 932 bytes (3%) del espacio de almacenamiento de programa. El máximo de memoria dinámica disponible es de 2048 bytes. Las variables Globales usan 9 bytes (0%) de la memoria dinámica, dejando 2039 bytes disponibles.

Carga completada.

Lín. 15, col. 2 Arduino Nano en COM7

Podemos ver en la imagen que el programa se subió correctamente y utiliza muy poca memoria: 932 bytes de memoria de programa y 9 bytes de memoria RAM.

Nota: Encontraremos en muchos ejemplos la asignación de entradas/salidas como si fuera un valor, por ejemplo "`int LED=12;`". Esto ocupará memoria. En este programa no importa, ya que tenemos mucho espacio, pero no es necesario sacrificar espacio de memoria para una simple sustitución en tiempo de compilación.

- Prueba a acender el resto de los LEDes que tiene EchidnaBlack.



LED_PWM.

Ahora toca manejar la intensidad de algunos LEDs conectados a salidas que permiten PWM, que son los pines marcados con “~”: RD9, GD5, BD6 y Gre D11.

```
/* LED\_PWM
   Encender y apagar un led con una
   intensidad dada.
*/
#define LED 11 // Led conectado al pin 11

void setup() {
  pinMode (LED, OUTPUT); // Definimos modo salida
}

void loop() {
  analogWrite (LED,16); // Encendemos el led al 6%
  delay (100);
  analogWrite (LED,30); // Encendemos el led al 12%
  delay (100);
  analogWrite (LED,64); // Encendemos el led al 25%
  delay (100);
  analogWrite (LED,128); // Encendemos el led al 50%
  delay (100);
  analogWrite (LED,256); // Encendemos el led al 100%
  delay (100);
}
```

Análisis:

Este ejemplo no es el más adecuado para manejar la intensidad de los LEDs, ya que tendríamos que escribir cada uno de los valores. Para mejorar esto, podremos usar la estructura de programación de repetición **for**, que permite repetir un conjunto de instrucciones de manera automática, ajustando la intensidad de los LEDs de forma más dinámica y eficiente.



Operadores comunes:

Operadores aritméticos (A =2, B = 4)

| Operador | Nombre | Descripción | Ejemplo |
|----------|----------------|---|------------------|
| " = " | Asignación | Almacena el valor en la variable De la izquierda. | A = B dará A = 4 |
| + | Suma | Agrega dos operandos. | A + B dará 6 |
| - | Resta | Resta el segundo operando del primero. | A - B dará -2 |
| * | Multiplicación | Multiplica dos operandos. | A* B dará 8 |
| / | División | Divide numerador por denominador. | B / A dará 2 |
| % | Módulo | Resto después de la división. | B % A dará 0 |

Operadores de comparación (A =2, B = 4)

| Operador | Nombre | Descripción | Ejemplo |
|----------|---------------|---|----------------------|
| == | Igual a | Comprueba si los operandos son iguales | (A == B) dará falso |
| != | Non igual | Comprueba si los operandos no son iguales | (A != B) dará cierto |
| < | Menor que | Comprueba si el operando de la izquierda es menor que el de la derecha. | (A < B) dará cierto |
| > | Mayor que | Comprueba si el operando de la izquierda es mayor que el de la derecha. | (A > B) dará falso |
| <= | Menor o igual | Comprueba si el operando de la izquierda es menor o igual que el de la derecha. | (A <= B) dará cierto |
| >= | Mayor o igual | Comprueba si el operando de la izquierda es mayor o igual que el de la derecha. | (A >= B) dará falso |



Operadores lógicos (A =2, B = 4)

| Operador | Nombre | Descripción | Ejemplo |
|----------|--------|---|----------------------|
| && | E | Operador lógico Y (And) si ambos son distintos de cero devuelve verdadero. . | (A && B) dará cierto |
| | OU | Operador lógico O (Or) si cualquiera de los operandos es distinto de cero devuelve verdadero. | (A B) dará cierto |
| ! | Non | Operador lógico No (Not) se usa para invertir el estado lógico. | !(A && B) dará falso |

Operadores booleanos (A =2, B = 3)

| Operador | Nombre | Descripción | Ejemplo |
|----------|----------------------|---|--------------------------------------|
| & | E | El operador Y (And) binario copia un bit en el resultado si existe en ambos operandos. | (A & B) dará 2 que é 10 en binario |
| | O | El operador O (Or) binario copia un bit en el resultado si existe en cualquiera de los operandos. | (A B) dará 3, 11 en binario |
| ^ | XO | El operador binario XOR (XO) copia el bit si está configurado en un operando pero no en ambos. | (A ^ B) dará 1, 1 en binario |
| ~ | NO | El operador complementario cambia los unos por ceros. | (~A) dará 253 , en binario 1111 1101 |
| << | Desplaza a izquierda | Desplaza a la izquierda el número de bits indicado por el operando de la derecha. | A << 2 dará 8, en binario 1000 |
| >> | Desplaza a derecha | Desplaza a la derecha el número de bits indicado por el operando de la derecha. | A >>2 dará 0, en binario 0000 |



Operadores compuestos (A =2, B = 3)

| Operador | Nombre | Descripción | Ejemplo |
|----------|--------------------------|---|---|
| ++ | Incremento | El operador incrementará el valor entero en una unidad. | A++ dará 3 |
| -- | Decremento | El operador decrementa el valor entero en una unidad. | A-- dará 1 |
| += | Suma compuesta | Suma el operando derecho al operando izquierdo y asigna el resultado al operando izquierdo. | B += A es equivalente a B = B + A |
| -= | Resta compuesta | Resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo. | B -= A es equivalente a B = B - A |
| *= | Multiplicación compuesta | Multiplica el operando derecho por el operando izquierdo y asigna el resultado al operando izquierdo. | B *= A es equivalente a B = B * A |
| /= | División compuesta | Divide el operando derecho por el operando izquierdo y asigna el resultado al operando izquierdo. | B /= A es equivalente a B = B / A |
| %= | Módulo compuesto | Obtiene el módulo usando los operandos y asigna el resultado al operando izquierdo. | B %= A es equivalente a B = B % A |
| = | Operación O compuesto | Realiza la operación OR (O) bit a bit y deja el resultado en el operando izquierdo. | A = 2 es lo mismo que A = A 2 |
| &= | Operación Y compuesto | Realiza la operación AND (Y) bit a bit y deja el resultado en el operando izquierdo. | A &= 2 es lo mismo que A = A & 2 |

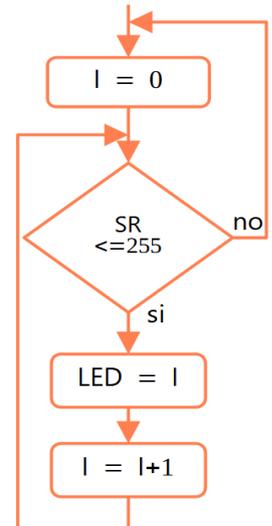
Más información: <https://www.arduino.cc/reference/es/>



For.

La declaración **for** se utiliza para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Tiene tres partes separadas por (;). La inicialización de la variable local ocurre una sola vez y la condición se verifica cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue siendo verdadera, las instrucciones del bucle se ejecutan nuevamente. Cuando la condición ya no se cumple, el bucle termina.

Cualquiera de los tres elementos de la cabecera puede omitirse, aunque el punto y coma es obligatorio. Además, las declaraciones de inicialización, condición y expresión pueden ser cualquier instrucción válida en lenguaje "C" sin relación con las variables declaradas.



Sintaxis:

```
for (inicialización; condición; cambio){instrucciones}
```

LED_PWM_2

Incremento del brillo de uno de los LEDs RD9, GD5, BD6, y Gre D11. Mediante estructura de repetición **for**.

```
/* LED_PWM, Variar la intensidad de un led con modulación de la anchura de pulso  
usando un bucle for.  
*/
```

```
#define LED 11 // Definimos Led conectado al pin 11
```

```
void setup() {  
  pinMode (LED, OUTPUT); // Definimos LED modo salida  
}  
void loop() {  
  // Incrementamos 0 desde 0 hasta 255 en pasos de 1 en 1  
  for (int i = 0; i <= 255; i++)  
  {  
    analogWrite(LED, i); // Escribimos o valor i "PWM" en la salida  
    delay(10); // Hacemos una pausa de 10ms entre incrementos  
  }  
}
```



Colores_RGB.

Vamos a encender el LED RGB con colores aleatorios, podemos tener 16777216 colores.

```
/* Cores_RGB
   colores aleatorios
*/
#define RGB_R 9 //Led RGB rojo
#define RGB_G 5 //Led RGB verde
#define RGB_B 6 //Led RGB azul

void setup() {
  pinMode (RGB_R, OUTPUT); // Definimos como salida
  pinMode (RGB_G, OUTPUT); // "
  pinMode (RGB_B, OUTPUT); // "
}
void loop() {
  analogWrite(RGB_R, random(255)); // PWM aleatorio rojo
  analogWrite(RGB_G, random(255)); // " " verde
  analogWrite(RGB_B, random(255)); // " " azul
  delay(500);
}
```

Análisis:

En este programa se utiliza la instrucción `random()`, que devuelve un valor pseudoaleatorio.

Sintaxis:

`random(max);` Devuelve un valor que va de 0 a máximo.
`random(min , max);` Devuelve un valor que va de min a max,

- Probamos a pulsar el botón de "Reset". ¿Qué sucede?

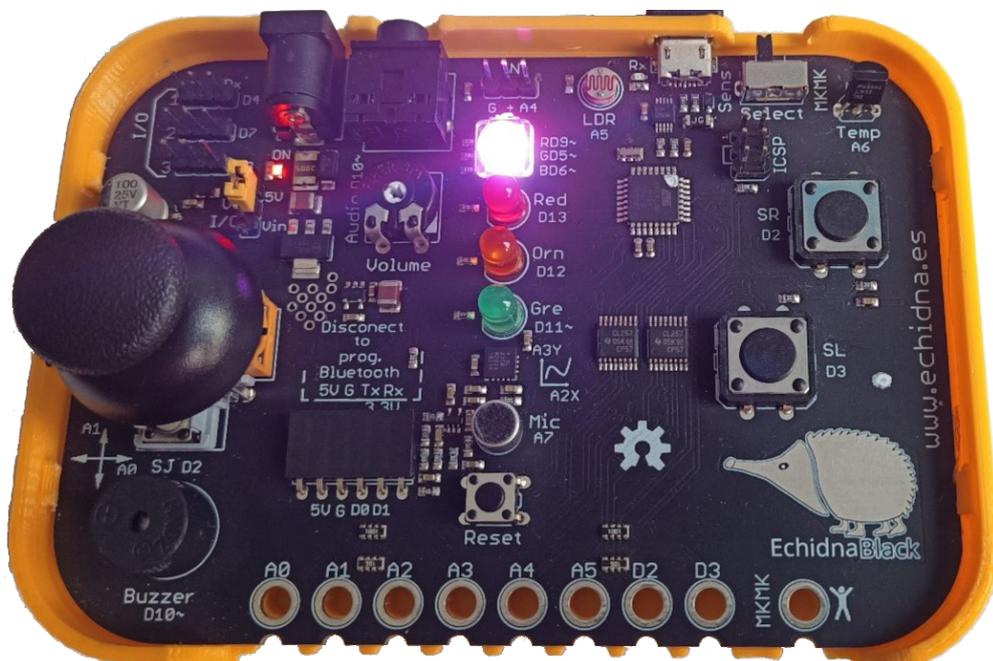


Veremos que siempre hace la misma secuencia de colores, no es aleatorio real. Para corregir eso, tenemos que usar una semilla que podamos tomar del "mundo real", por ejemplo, de una entrada analógica no usada `randomSeed(anaLogRead(A4));`; en esa entrada no tenemos nada conectado (¿o sí?) y usar ese valor desconocido como semilla:

Incluimos la siguiente línea después de la definición del modo de los pines:

```
void setup() {  
  pinMode (RGB_R, OUTPUT); // Definimos modo saída  
  pinMode (RGB_G, OUTPUT); // "  
  pinMode (RGB_B, OUTPUT); // "  
  randomSeed(anaLogRead(A4)); //Semilla aleatoria basada en A4  
}
```

Comprobamos ahora pulsando el botón de Reset, que ya no comienza siempre igual.





Tonos.

Podemos usar la técnica PWM para hacer sonidos (tonos), pero tenemos una instrucción dedicada al sonido "tone()" que genera una onda cuadrada de una frecuencia ajustable con un ciclo de trabajo del 50%, también tenemos una instrucción para parar "noTone()".

Sintaxis:

```
tone(Pin, frecuencia);
```

```
tone(Pin, frecuencia, Duración);
```

```
noTone(Pin);
```

```
/* Buzzer\_previo, Ejemplo de sonidos con el buzzer
*/
#define Buzzer 10 //Zumbador D10~
void setup() {
  pinMode (Buzzer, OUTPUT); //Definimos como salida
  tone(Buzzer, 493, 70.7); //Frecuencia 493Hz durante 70,7mS
  delay(70.7); //Retardo de 70,7mS
  delay(70.7);
  tone(Buzzer, 987, 70.7);
  delay(70.7);
  delay(70.7);
  tone(Buzzer, 739, 70.7);
  delay(70.7);
  delay(70.7);
  tone(Buzzer, 622, 70.7);
  delay(70.7);
  delay(70.7);
  tone(Buzzer, 987, 70.7);
  delay(70.7);
  tone(Buzzer, 739, 70.7);
  delay(70.7);
  delay(141.5);
  tone(Buzzer, 622, 212.2);
  noTone(10); // Paramos el sonido en el pin D10
}
void loop() { // No es necesario poner nada, no necesitamos repetir
}
```



🔍 Análisis:

Podemos ver en el código que las instrucciones de tono están en la función `setup()`, esto se hace para que solo ejecute los tonos una vez.

En la función `loop()` no tenemos instrucciones ya que realmente no es necesario hacer nada.

Esto puede escribirse usando una llamada a una nueva función `void loop()` que no necesita parámetros.

```
/* Buzzer
   Sonido con Buzzer
*/
#define Buzzer 10 //Zumbador D10~
void setup() {
  pinMode (Buzzer, OUTPUT); //Definimos como salida
  son(); //llamamos a la función son
}
void loop() {
  //nada, no necesitamos repetir
}
void son(){
  tone(Buzzer, 493, 70.7); // Frecuencia 493Hz durante 70,7mS
  delay(70.7); // Retardo de 70,7mS
  delay(70.7);
  tone(Buzzer, 987, 70.7);
  delay(70.7);
  delay(70.7);
  tone(Buzzer, 739, 70.7);
  delay(70.7);
  delay(70.7);
  tone(Buzzer, 622, 70.7);
  delay(70.7);
  delay(70.7);
  tone(Buzzer, 987, 70.7);
  delay(70.7);
  tone(Buzzer, 739, 70.7);
  delay(70.7);
  delay(141.5);
  tone(Buzzer, 622, 212.2);
  noTone(10); // paramos el sonido en el pin D10
```



}

🔍 Análisis:

En la función `setup()` llamamos a `tone()`, y una vez terminada sigue a `loop()`, donde se queda haciendo nada "-:-").

Por cierto, ¿te suenan esas notas?

En los ejemplos que trae Arduino, tenemos un ejemplo de melodía

"Archivo/ejemplos/02.Digital/toneMelody", solo tenemos que cambiar el pin de salida de 8 por el **10** (líneas 37 y 44).

Nota: Para mostrar los números das líneas en el IDE [Archivo] [Preferencias]

| | | |
|---|--|---|
|  | <pre> 33 34 // to calculate the note duration, 35 //e.g. quarter note = 1000 / 4, ei 36 int noteDuration = 1000 / noteDura 37 tone(8, melody[thisNote], noteDura 38 39 // to distinguish the notes, set a 40 // the note's duration + 30% seem 41 int pauseBetweenNotes = noteDura 42 delay(pauseBetweenNotes); 43 // stop the tone playing: 44 noTone(8); 45 } </pre> | <pre> 33 34 // to calculat by the 35 //e.g. quarter 36 int noteDurati 37 tone(10, meloc 38 39 // to distinguem. 40 // the note's 41 int pauseBetwe 42 delay(pauseBet 43 // stop the tc 44 noTone(10); 45 } </pre> |
|---|--|---|



Pulsador_I.

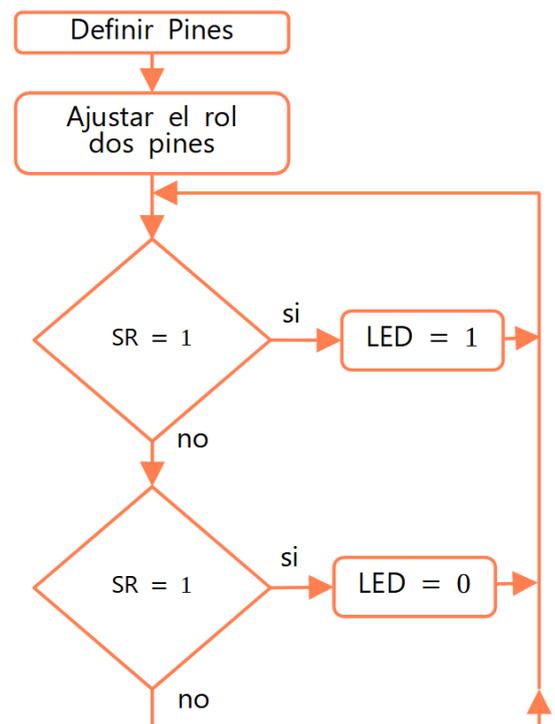
Programa que enciende un LED cuando presionamos el pulsador (sin variables).

Lectura digital 0/1.

Para este ejemplo usamos una estructura de control If:

Es una sentencia que se utiliza para probar si una determinada condición se ha alcanzado, por ejemplo, comprobar si una igualdad es cierta, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves **{}**. Si es falso (la condición no se cumple), no ejecuta las operaciones que están dentro de las llaves.

```
/* Pulsador_I
Encender un LED cuando se active el pulsador
sin usar variables
*/
#define SR 2 // Pin del pulsador
#define LED 11 // pin del led
void setup() {
  pinMode(LED, OUTPUT); //modo salida
  pinMode(SR, INPUT); //modo entrada
}
void loop() {
  if (digitalRead (SR) == HIGH) { //comprueba si
SR es igual a "1"
  digitalWrite(LED, HIGH); //enciende el LED
}
  if (digitalRead (SR) == LOW) { //comprueba si
SR es igual a "0"
  digitalWrite(LED, LOW); //apaga el LED
}
}
```



Análisis:

En este ejemplo hacemos dos comprobaciones sobre dos lecturas consecutivas de la misma entrada. No es lo más recomendable, pero podemos usar la siguiente estructura.



if...else.

Viene siendo una estructura que se ejecuta en respuesta a la idea:

Si se cumple la condición, haz esto; si no, haz esto otro. Por ejemplo, se desea probar una entrada digital y hacer una cosa si la entrada es alta, o hacer otra cosa si la entrada no es alta.

Como hicimos en el caso anterior, vamos a encender el LED cuando lo pulsemos SR = 1, y apagar en el caso contrario.

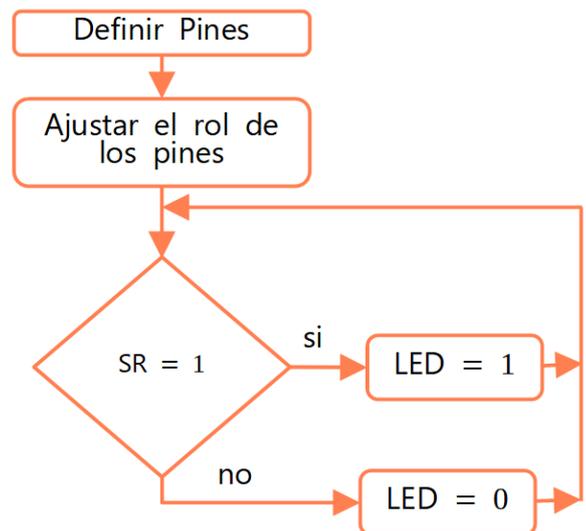
```

/* Pulsador_2
  Encender un LED cuando se active el pulsador
  Apagar en caso contrario
*/
#define BTN 2 // Pin del pulsador
#define LED 11 // Pin del LED

void setup() {
  pinMode(LED, OUTPUT); // Configurar el pin como salida
  pinMode(BTN, INPUT); // Configurar el pin como entrada
}

void loop() {
  // Si el pulsador está en "1", enciende el LED
  if (digitalRead(BTN) == HIGH) {
    digitalWrite(LED, HIGH);
  }
  // En caso contrario, apaga el LED
  else {
    digitalWrite(LED, LOW);
  }
}

```





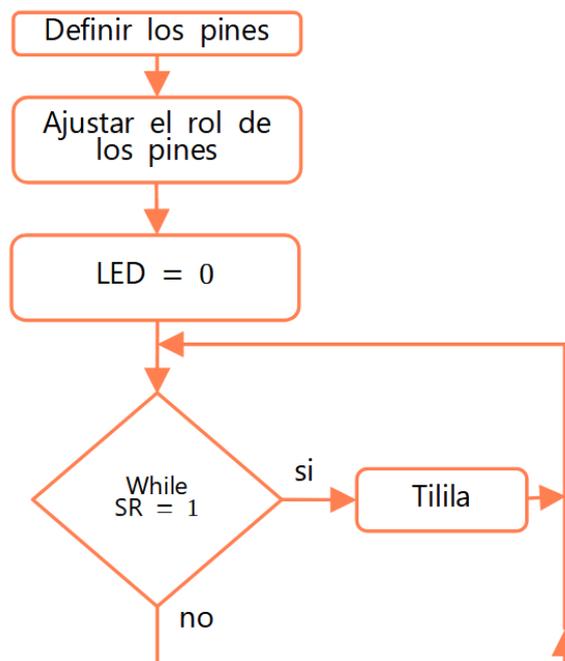
While

Vamos a hacer parpadear el LED cuando se pulse SR, usando la estructura While. Este bucle se ejecuta de forma continua e infinita, siempre que la condición dentro del paréntesis sea verdadera. Cuando la expresión dentro del paréntesis se convierte en falsa, deja de ejecutarse.

```
/* Titila\_II
   Titila cando se pulse SR usando la estructura While
*/
#define L_Orn 12 // Led naranja
#define SR 2 // Pulsador SR

void setup() {
  pinMode(SR, INPUT); // Definimos modo entrada para SR
  pinMode(L_Orn, OUTPUT); // Definimos modo salida para LED naranja
  digitalWrite(L_Orn, LOW); // Apaga el LED
}

void loop() {
  // Mientras el pulsador SR valga 1 hacemos parpadear el LED
  while (digitalRead(SR) == 1) {
    digitalWrite(L_Orn, HIGH); // Enciende el LED
    delay(50); // Espera 50 milisegundos
    digitalWrite(L_Orn, LOW); // Apaga el LED
    delay(100); // Espera 100 milisegundos
  }
}
```





Lectura analógica.

Vamos a comenzar a leer los sensores analógicos. Para hacer más cómoda la medición, vamos a enviar los datos por la conexión USB y verlos en el ordenador.

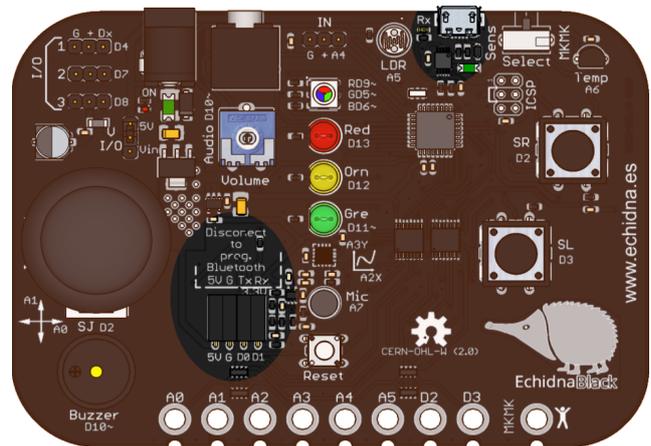
EchidnaBlack cuenta con comunicación serie mediante conexión USB y con un zócalo para adaptador Bluetooth.

Recuerda que para subir los sketches es recomendable no tener conectado el adaptador Bluetooth, ya que comparten las mismas conexiones.

Para usar la comunicación serie, veremos las instrucciones:

```
Serial.begin(); Serial.print(); Serial.println();
```

Existen más instrucciones relacionadas con la comunicación serie que iremos viendo en otros ejemplos. Ahora nos centraremos en las fundamentales para enviar datos y poder visualizarlos en el ordenador.



Sintaxis:

```
Serial.begin(Velocidad); //configura y inicializa el canal serie  
Velocidad en bits por segundo (Baudios).
```

Velocidades más usadas, 2400, 4800, **9600**, 19200, 38400 o 115200

Podemos usar otras velocidades para adaptar los terminales de comunicación. EchidnaBlack está probada desde 300 hasta 2,000,000 baudios.

También podemos ajustar el resto de los parámetros de comunicación:

```
Serial.begin(Velocidad, configuración);
```

Configuración: SERIAL_ y número de datos por palabra, paridad par o impar y bits de parada uno o dos. Por defecto, envía 8 bits, sin paridad y un bit de parada.

```
Serial.begin(Velocidad, SERIAL_8N1); es el mismo que Serial.begin(9600);
```

Más info: <https://www.arduino.cc/reference/es/language/functions/communication/serial/begin/>

 Tendremos cuidado al enviar datos en formato decimal sin separadores, ya que si el sistema operativo o una aplicación no los espera en ese formato, podrían generar errores.



Una vez configurada la comunicación, ya podemos enviar datos, estos pueden ser numéricos o cadenas de texto.

Sintaxis:

```
Serial.print(ValorSensor); // envía el ValorSensor
Serial.print("\t"); // envía una separación tabulada entre valores
// envía el ValorSensor y un linea nueva de separación
Serial.println(ValorSensor);
```

JoyStick_analog.

```
/* Joystick_analog
 * Lectura analógica de los ejes X,Y del Joystick
 */

#define Joy_X A0 // Eje X conectado -> A0
#define Joy_Y A1 // Eje Y conectado -> A1

void setup() {
  // Inicia comunicación serie a 9600 Bps
  Serial.begin(9600);

  // No es preciso configurar los pines como entrada,
  // Pero es una buena práctica para recordar el rol de cada pin.
  pinMode(Joy_X, INPUT); // Definimos como entrada
  pinMode(Joy_Y, INPUT); // "
}

void loop() {
  // Leemos las entradas
  int Valor_X = analogRead(Joy_X);
  int Valor_Y = analogRead(Joy_Y);
  // Envía los valores vía serie
  Serial.print(Valor_X);
  Serial.print("\t"); // Envía un tabulador para separar los valores
  Serial.println(Valor_Y);
  // Un pequeño retardo para estabilizar las medidas.
  delay(1);
}
```



🔑 Análisis:

En el ejemplo vemos que aparecen unas variables “Valor_X y Valor_Y” precedidas de “int”. Las variables son espacios de memoria donde vamos a almacenar datos que pueden variar con el tiempo. Debemos declarar cómo serán esos datos, en este ejemplo:

“int” (entero) usa 16 bits (2 bytes), los valores van de -32,767 a 32,767.

La variable puede declararse de forma local, solo para la función en la que se define, o de forma global si se declara antes del `setup()`. También se le puede asignar un valor inicial, por ejemplo, “`int Valor_X = 0`”.

Los datos transmitidos se pueden verificar en el monitor serie integrado en el IDE Arduino.

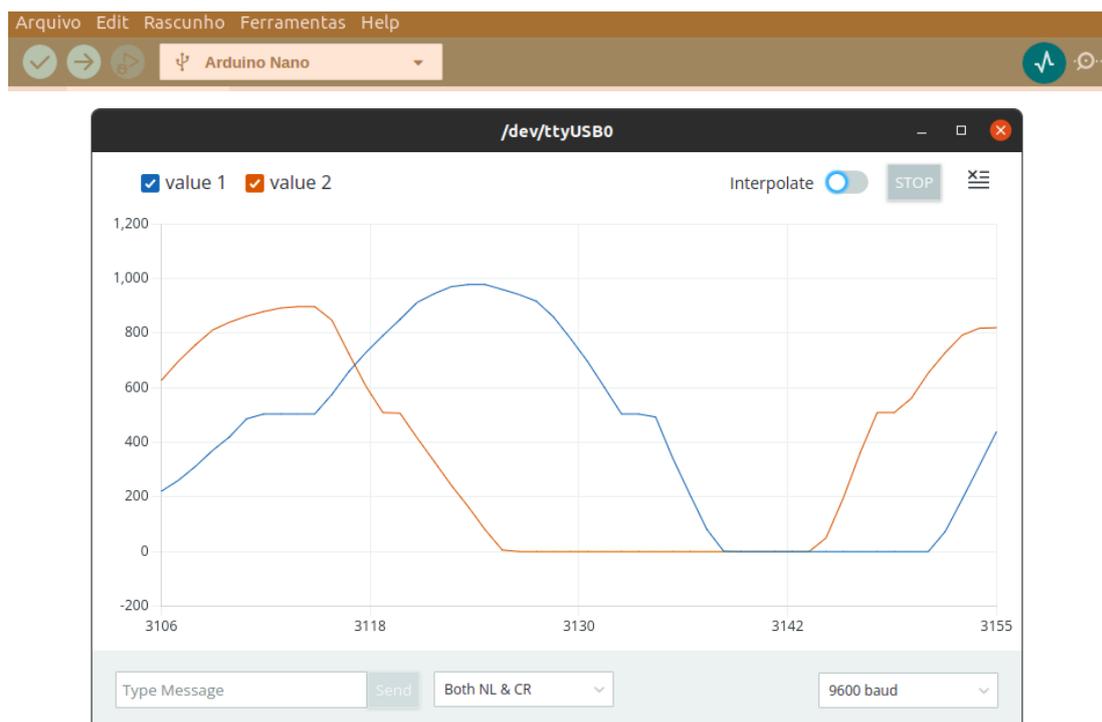
Recuerda ajustar los parámetros de comunicación del terminal, como se ha establecido en el programa con:

```
Serial.begin(9600);
```

The screenshot shows the Arduino IDE interface. The main window displays the code for `Joystick_analog.ino`. The code includes comments in Portuguese and defines two joystick axes, Joy_X and Joy_Y, connected to pins A0. The `setup()` function initializes the serial communication at 9600 Bps. The serial monitor at the bottom shows the output: "504 509" repeated on multiple lines, indicating the joystick's position.

Vemos los datos numéricos separados por tabuladores.

Otra forma de ver los datos que nos ofrece el IDE es el gráfico “plotter” serie.





Tipos de datos:

(Ojo, no están todos) (Var es el nombre de la variable, val es el valor que asignamos a la variable).

- `bool` `Var = val; //un bit, binario 0 o 1, falso (false) o verdadero (true).`
- `byte` `Var = val; //Almacena 8 bits (1 byte), 0 a 255`
- `uint8_t` `Var = val; //Almacena 8 bits (1 byte), 0 a 255`
- `int` `Var = val; //Entero usa 16 bits (2 Bytes) los valores van de -32767 a 32767`
- `word` `Var = val; //Ocupa 16 bits sin signo de 0 a 65535`
- `uint16_t` `Var r = val; //Almacena 16 bits, 0 a 65535`
- `unsigned int` `Var = val; //Entero sen signo, 2 bytes, 0 a 65535`
- `short` `Var = val; //Ocupa 16 bits (2 bytes) igual que "int"`
- `long` `Var = val; //Ocupa 32 bits (4 bytes)2,147,483,648 a 2,147,483,647`
- `double` `Var = val; //Ocupa 64 bits (4 bytes) igual que "float"`
- `float` `Var = val; //Ocupa 32 bits (4 bytes)-3.4028235E+38 a 3.4028235E+38`
- `unsigned long` `Var = val; //Largo sien signo, 32 bits(4 bytes),0 a 4,294,967,295`
- `char` `Var = val; // Almacena un carácter, los caracteres literales // se escriben entre comillas simples 'A' // para varios caracteres se usan comillas dobles "AEI"`



Propuestas A,B:

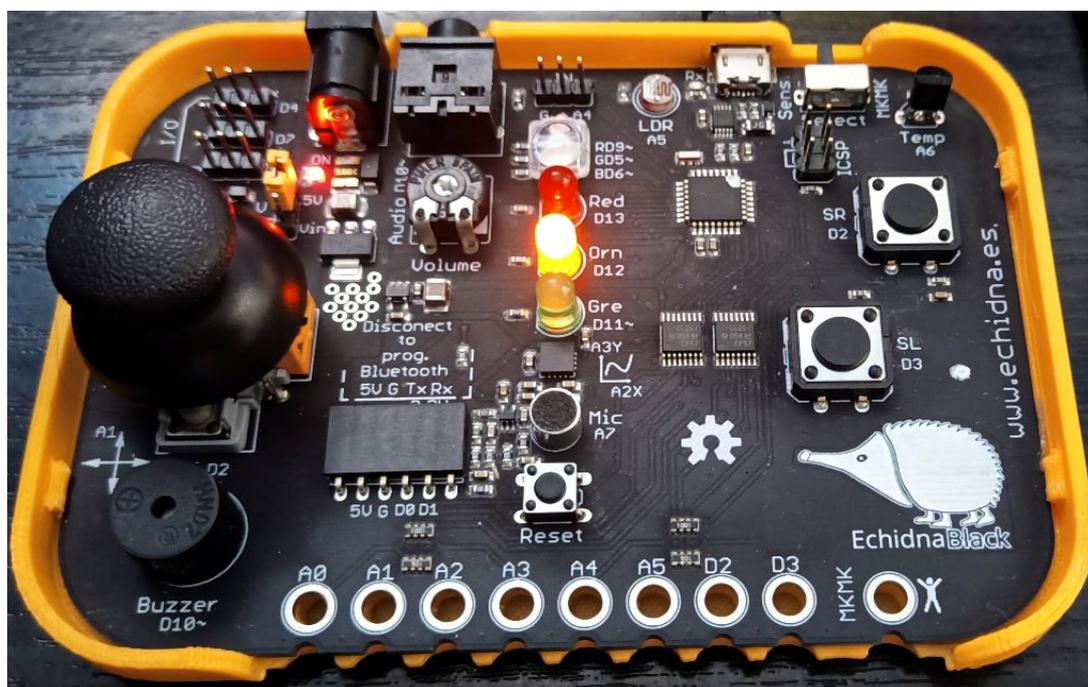
A) Escribe un programa que active un LED cuando se supere un valor en una entrada analógica,

Por ejemplo, enciende el LED rojo cuando $A1 \geq 550$.

```
...
void loop() {
  if (analogRead (A1) >= 550) {
    digitalWrite(L_Red, HIGH);
  }
  else {
    digitalWrite(L_Red, LOW);
  }
}
```

B) Escribe un programa de una baliza, que funcione según la siguiente lógica:

| Condición/LEDes | Rojo | Naranja | Verde |
|------------------|------|---------|-------|
| $A1 \geq 600$ | 1 | 0 | 0 |
| $400 < A1 < 600$ | 0 | 1 | 0 |
| $A1 \leq 400$ | 0 | 0 | 1 |





LDR.

En este ejemplo vamos a hacer un interruptor crepuscular, que encenderá un LED cuando la luz baje de un valor dado.

```
/*LDR_crepuscular
Lectura analógica da LDR
Interruptor crepuscular
*/
#define LDR A5 //LDR conectada a A5
#define L_Orn 12 //Led naranja a D12
int umbral = 600; //valor activación

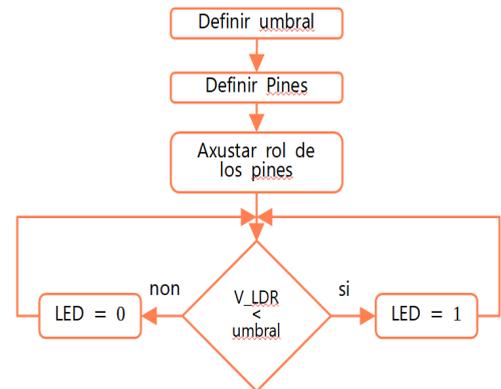
void setup() {
  pinMode (LDR, INPUT); // Definimos como entrada
  pinMode (L_Orn, OUTPUT); // Definimos modo salida
}

void loop() {
  // Leemos la entrada
  int Valor_LDR = analogRead(LDR);
  // Comprueba si el valor está por debajo
  if ( Valor_LDR < umbral) {
    digitalWrite (L_Orn, HIGH); // Enciende el LED
  }
  else {
    digitalWrite (L_Orn, LOW); // Apaga el LED
  }
  // Un pequeño retardo para estabilizar las medidas.
  delay(1);
}
```

🔍 Análisis:

Una vez que tenemos el valor de la LDR, lo comparamos con el valor de activación "umbral". Si el valor de la luz es inferior a este valor, encendemos el LED; en caso contrario, permanece apagado. Este sketch realiza la función de un Interruptor crepuscular.

✂ ES recomendable usar valores de histéresis, veremos un ejemplo en el programa [Temperatura](#)





Theremín LDR.

Vamos a manejar el zumbador manipulando la luz que incide en la LDR, emulando el famoso "Theremin". ;-)

```
/*TheremLDR
  Un pequeño programa, con la LDR manejamos el zumbador y un par de LEDes
*/
// Entradas y salidas
#define LDR A5
#define Buz 10
#define RGB_R 9
#define RGB_G 5
int Valor_LDR; // Variables
int Frecuencia;
int BrilloR;
int BrilloG;
void setup() {
  pinMode (LDR ,INPUT); // Ajuste del modo
  pinMode (Buz ,OUTPUT);
  pinMode (RGB_R ,OUTPUT);
}
void loop() {
  Valor_LDR = analogRead(LDR); // Leemos a LDR
  // Escala el Valor de la LDR a la frecuencia
  Frecuencia = map(Valor_LDR, 0, 1023, 400, 2500);
  BrilloR = map(Valor_LDR, 0, 1023, 0, 128); //Valor_LDR al brillo del LED
  BrilloG = map(Valor_LDR, 0, 1023,16, 64); //escala para colorear
  // Si se oscurece apaga el zumbador
  if (Valor_LDR<50){ Frecuencia= 0;}
  tone(Buz, Frecuencia, 10); // Emite la frecuencia dependiente de la LDR
  analogWrite(RGB_R, BrilloR); // Luz roja
  analogWrite(RGB_G, BrilloG); // Luz verde
  delay(1); // Espera para estabilizar la conversión AD
}
```



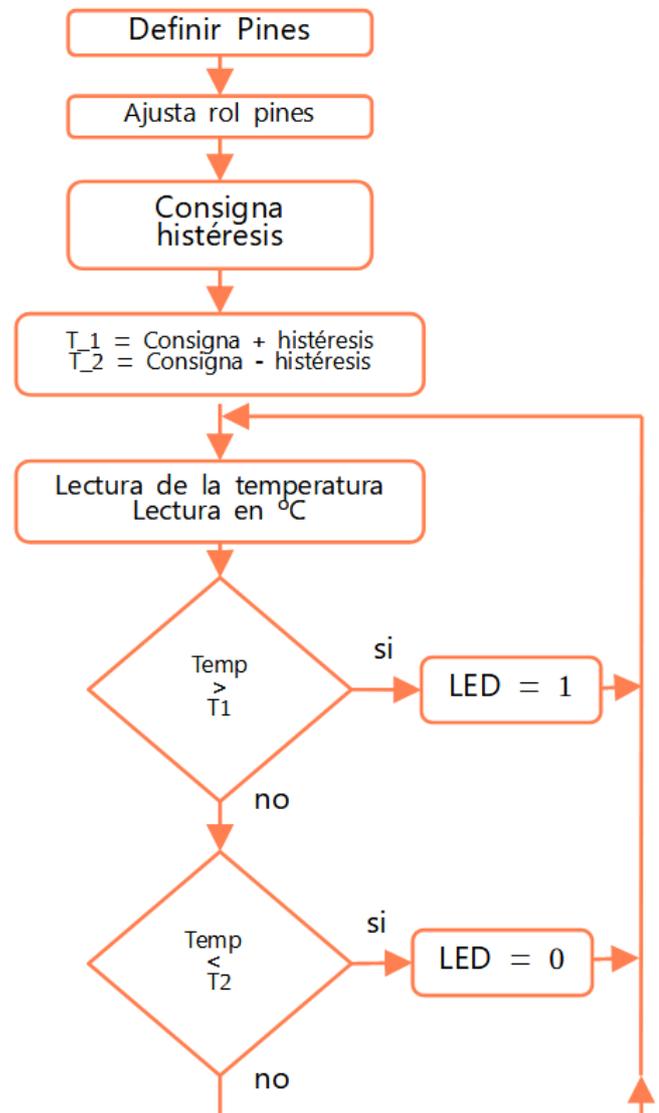
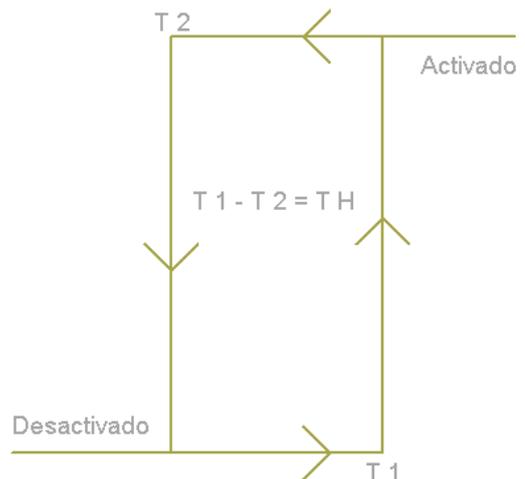
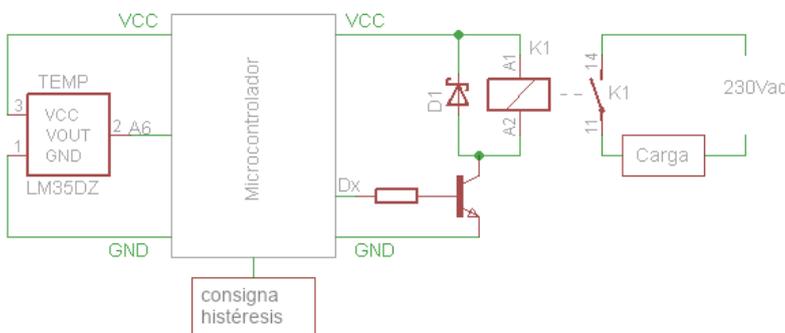
Temperatura.

Usando el sensor de temperatura LM35 (LM35_Pin), encenderemos un LED cuando la temperatura supere un valor, también enviaremos el valor a través de la comunicación serie con el ordenador.

Se tendrá en cuenta una pequeña histéresis para evitar activar repetidamente la salida cuando la temperatura esté cerca del punto de conmutación, lo que podría quemar el posible relé o contactor asociado (que activará la salida).

Así, se establecerán dos temperaturas: T_1 , la temperatura de activación, y T_2 , la temperatura de desactivación. La diferencia entre ellas se denominará histéresis de temperatura.

El esquema representa un circuito de control de una carga (como un refrigerador) controlada por nuestro programa, que envía señales a un contactor encargado de gestionar la potencia necesaria para la carga.





```
*/ Termóstato LM35
  Comparación de la temperatura con una dada como Consigna
  para encender un led (activar una salida)
*/
#define L_Gre 11 // Pin del LED verde -> D11
#define LM35 A6  // Pin del LM35 -> A6

int Consigna = 27; // Temperatura elegida °C
// Diferencia en grados sobre activación y desactivación
int Diferencia = 2;
int histéresis = (Diferencia / 2);
int T_1 = Consigna + histéresis; // Temperatura de activación
int T_2 = Consigna - histéresis; // Temperatura de desactivación
int temperatura; // Temperatura medida
char *Estado = "Desactivado"; // Cadena de texto del estado

void setup() {
  analogReference(INTERNAL); // Referencia analógica a 1.1V
  Serial.begin(9600); // Configuración del puerto serie
  pinMode(LM35, INPUT); // Pin LM35 como entrada
  pinMode(L_Gre, OUTPUT); // Salida para LED
  digitalWrite(L_Gre, LOW); // Apaga el LED, estado inicial
}

void loop() {
  int lectura = analogRead(LM35); // Devuelve un valor entre 0 y 1023

  // Temperatura en ° Celsius
  temperatura = (lectura * 1.1 * 100) / 1024;
  if (temperatura > T_1) { // Si supera el valor
    digitalWrite(L_Gre, HIGH); // Enciende el LED
    Estado = "Activado";
  }

  if (temperatura < T_2) { // Si baja del valor
    digitalWrite(L_Gre, LOW); // Apaga el LED
    Estado = "Desactivado";
  }
}
```



```
// Envía mensajes vía serie
Serial.print("Temp Actual: ");
Serial.print(temperatura);
Serial.print("°C");
Serial.print("\t");
Serial.print("Consigna: ");
Serial.print(Consigna);
Serial.print("°C");
Serial.print("\t");
Serial.print("Histéresis: ");
Serial.print(histéresis);
Serial.print("°C");
Serial.print("\t");
Serial.print("Estado: ");
Serial.println(Estado);

delay(1000); // Tiempo de espera de 1s
}
```

🔑 Análisis:

- **Consigna**, dónde depositamos el valor de la temperatura, alrededor del cual queremos establecer la conmutación.
- **Histéresis**, tiene el valor que incrementa y decrementa el valor de consigna para establecer los dos puntos de conmutación.
 - $Diferencia = 1; histéresis = (Diferencia/2);$
 - $T_{1} = Consigna + histéresis;$
 - $T_{2} = Consigna - histéresis;$
- ***Estado**, es una variable que contendrá los caracteres para enviar vía serie el estado de activado o desactivado.
- Como explicamos antes, con la instrucción `analogReference(INTERNAL)`, ajustamos la referencia de conversión analógica a digital a 1,1 V para obtener una mayor precisión.
- El cálculo de la temperatura en °C es $temperatura = (lectura * 1.1 * 100) / 1024;$, donde tenemos en cuenta la referencia de tensión de 1,1 V y



la resolución de 10 bits de la conversión analógica a digital ($2^{10} = 1024$).

- Realiza dos comparaciones para establecer el estado.
 - Una con el valor T_1 -> Activar y otra con T_2 -> Desactivar.
- Ya solo queda enviar los datos de funcionamiento vía serie.
 - `Serial.print("Temp Actual: ");`
 - `Serial.print(temperatura);`

```

Mensagem (Ctrl + Enter para enviar mensagem para 'Arduino Nano' em '/dev/ttyUSB0' Nova linha 9600 baud
Temp Actual: 27°C      Consigna: 27°C      Histérese: 2°C      Estado: Desactivado
Temp Actual: 27°C      Consigna: 27°C      Histérese: 2°C      Estado: Desactivado
Temp Actual: 27°C      Consigna: 27°C      Histérese: 2°C      Estado: Desactivado
Temp Actual: 27°C      Consigna: 27°C      Histérese: 2°C      Estado: Desactivado
Temp Actual: 27°C      Consigna: 27°C      Histérese: 2°C      Estado: Desactivado
Temp Actual: 32°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 32°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 31°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 30°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 30°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 29°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 30°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 30°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 30°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
Temp Actual: 29°C      Consigna: 27°C      Histérese: 2°C      Estado: Activado
  
```

Vúmetro.

Vamos a simular un vúmetro el medidor de volumen, usando todos los LEDes como indicador e o micrófono como captador de sonido.

/* [Vumetro](#)

Usa los LEDes L_Gre, L_Orn, L_Red, RGB_R, RGB_G e RGB_B como indicadores de nivel.

```

#define RGB_B 6 // LED RGB azul
#define RGB_G 5 // LED RGB verde
#define RGB_R 9 // LED RGB rojo
#define L_Red 13 // LED rojo
#define L_Orn 12 // LED naranja
#define L_Gre 11 // LED verde
#define Mic A7 // Micrófono

// LEDes en una matriz para simplificar el manejo
int ledPins[] = {255, L_Gre, L_Orn, L_Red, RGB_R, RGB_G, RGB_B};
int V_Max = 20; // Valor máximo que queremos medir

void setup() {
  
```



```
// Definimos los LEDs como salidas
for (int i = 0; i < 7; i++) { // Recorremos la matriz
    pinMode(ledPins[i], OUTPUT); // Y definimos cada LED como salida
}
pinMode(Mic, INPUT);
// Cambiamos la referencia analógica a 1.1V
analogReference(INTERNAL);
}

void loop() {
    // Captura el valor analógico del micrófono
    int sinal = analogRead(Mic);
    // Comprueba que no nos pasamos de V_Max
    if (sinal > V_Max) {
        sinal = V_Max;
    }
    // Convierte los valores del MIC a un número de 0 a 7 (LEDs)
    int LED_mayor = map(sinal, 0, V_Max, 0, 7);
    // Enciende los LEDs hasta el valor más alto
    for (int i = 0; i < LED_mayor; i++) {
        digitalWrite(ledPins[i], HIGH);
    }
    // Apaga hasta el LED más bajo
    for (int i = 7; i > LED_mayor; i--) {
        digitalWrite(ledPins[i], LOW);
    }
    delay(1);
}
```





🔍 Análisis:

Usamos una matriz donde guardamos los LED en orden, para poder recorrer las salidas de manera secuencial.

Las matrices se pueden definir de varias maneras:

`int ledPins[7]`; los valores no están definidos, pero se reserva el espacio para siete datos enteros.

`int ledPins[] = {0, 1, 2, 3, 4, 5, 6}`; no se indica la cantidad de datos, el compilador cuenta el número de datos y reserva el espacio con los datos entre llaves `{}` separados por comas.

`int ledPins[7] = {0, 1, 2, 3, 4, 5, 6}`; aquí decidimos cuántos datos tiene la matriz.

Al principio del programa definimos la matriz de los LEDes

```
"int ledPins[] = {255, L_Gre, L_Orn, L_Red, RGB_R, RGB_G, RGB_B};"
```

comenzando con 255, que es un espacio de memoria no usado; en el programa tiene la finalidad de ser como una salida “fantasma”, para que no se vea ningún LED encendido cuando no tengamos señal de entrada `“analogWrite(ledPins[0], 16);”`

Recorrer una matriz unidimensional usando un bucle “for” o con cualquier otro método es fácil.

En el `setup()` usamos `for (int i = 0; i < 7; i++) { pinMode(ledPins[i], OUTPUT); }`, para configurar los pines conectados a los LEDs como salidas.

Una vez que tenemos el valor de la señal de sonido, escalamos el número de LEDes disponibles (0-7) usando la instrucción `map`:

```
map(Valor, de bajo, de alto, a bajo, a alto);
```

Valor es la variable de entrada que tiene los datos a escalar, “de bajo” o “de alto” son los valores límites que queremos como entrada, y “a bajo” o “a alto” son los valores que queremos como salida.

La función `map()` no devuelve números fraccionarios, aunque el cálculo lo indique. Se puede usar para invertir valores como `x = map(y, 1, 50, 50, 1)`

Encendemos los LEDes con:

```
for (int i = 0; i < LED_mayor; i++) { digitalWrite(ledPins[i], HIGH); }
desde el 0 hasta el LED mayor.
```

Apagamos el resto (desde el LED mayor hasta el LED 6) con:

```
for (int i = 7; i > LED_mayor; i--) { digitalWrite(ledPins[i], LOW);}
```



Nivel eje Y.

Simulamos un nivel con la ayuda de la salida Y del acelerómetro. Cuando está en nivel, el LED Naranja está iluminado. Los LEDes Rojo o Verde se encenderán dependiendo de la inclinación.

```
/* Nivel_Y
 Nivel en el eje Y e indicadores LEDes
 */
#define Red 13 // LED rojo
#define Orn 12 // LED naranja
#define Gre 11 // LED verde
#define Ace_Y A3 // Entrada acelerómetro Y
word Valor_Y; // Variables de lectura del acelerómetro
// Estos valores pueden cambiar dependiendo de cada EchidnaBlack
word Reposo_Y = 359; // Valor de reposo, A nivel

void setup() {
  Serial.begin(9600);
  pinMode(Red, OUTPUT); // Modo salida para LED rojo
  pinMode(Orn, OUTPUT); // Modo salida para LED naranja
  pinMode(Gre, OUTPUT); // Modo salida para LED verde
  // Definimos entradas
  pinMode(Ace_Y, INPUT); // Acelerómetro Y
}

void loop() {
  Valor_Y = analogRead(Ace_Y);
  Serie();
  // Si el valor Y supera el reposo + 2
  if (Valor_Y > (Reposo_Y + 2)) {
    digitalWrite(Red, 1); // Encendemos el LED rojo
    digitalWrite(Orn, 0); // Apagamos el resto
    digitalWrite(Gre, 0);
  }
  // Si el valor Y está a nivel
  if (Valor_Y < (Reposo_Y + 2) && Valor_Y > (Reposo_Y - 2)) {
    digitalWrite(Red, 0);
  }
}
```



```

digitalWrite(Orn, 1); // Encendemos el LED naranja
digitalWrite(Gre, 0); // Apagamos el resto
}
if (Valor_Y < (Reposo_Y - 2)) {
digitalWrite(Red, 0);
digitalWrite(Orn, 0);
analogWrite(Gre, 32); // Encendemos el LED verde, apagamos el resto
} // Escribimos análogicamente 32, porque este LED
} // se ilumina mucho, así solo enciende un 1/4 del total

void Serie() {
Serial.print("Valor_Y:"); // Envía los datos vía serie
Serial.print(Valor_Y);
Serial.println("");
}

```

🔍 Análisis:

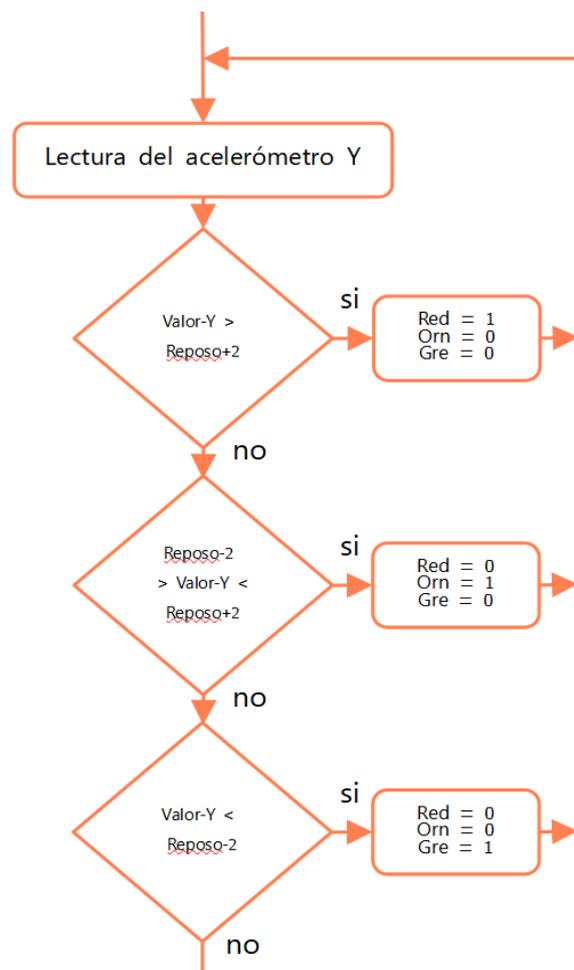
En este ejemplo, una vez realizada la lectura del valor Y del acelerómetro, realizamos tres comparaciones:

si superamos el valor de reposo +2,
activamos el LED rojo,

si el valor está entre el valor de reposo -2 y
el valor de reposo +2,
encendemos el LED naranja y

si el valor es menor que el de reposo -2,
encendemos el LED verde.

Una ampliación que podemos realizar sería memorizar el valor de reposo al presionar el pulsador SR o SL.



Esto podemos usarlo para hacer un juego en el que haya que sostener con mucho cuidado la EchidnaBlack sin inclinarla. :-)



Toma la EchidnaBlack sin activar a Alarma.

```
/* Vibraciones  
Alarma de movimiento: usamos el acelerómetro para activar la alarma  
(zumbador y LED RGB rojo) cuando intentamos moverla Echidna.  
Cuando parpadea el LED naranja, nos invita a pulsar SR  
para memorizar la posición inicial y comenzar el juego.  
Más información vía serie.  
*/
```

```
int umbral_X = 5; // Límites de alarma  
int umbral_Y = 5;  
  
#define RGB_R 9 // LED RGB rojo  
#define Orn 12 // LED naranja  
#define Gre 11 // LED verde  
#define Buz 10 // Zumbador, salida sonido  
#define SR 2 // Entrada pulsador SR  
#define Ace_X A2 // Entrada acelerómetro X  
#define Ace_Y A3 // Entrada acelerómetro Y  
  
word Valor_X; // Variables de lectura del acelerómetro  
word Valor_Y;  
word muestras = 100; // Número de lecturas consecutivas  
word media_X = 0;  
word media_Y = 0;  
  
void setup() {  
  Serial.begin(9600);  
  pinMode(RGB_R, OUTPUT); // Modo salida para LED RGB rojo  
  pinMode(Orn, OUTPUT); // Modo salida para LED naranja  
  pinMode(Gre, OUTPUT); // Modo salida para LED verde  
  pinMode(Buz, OUTPUT); // Modo salida para el zumbador  
  
  // Definimos como entradas  
  pinMode(Ace_X, INPUT); // Acelerómetro X  
  pinMode(Ace_Y, INPUT); // Acelerómetro Y  
  pinMode(SR, INPUT); // Pulsador SR
```



```
Serial.println("");
Serial.print("Pulsa SR para memorizar la posición");

while (digitalRead(SR) == 0) {
    digitalWrite(Orn, HIGH);
    delay(10);
    digitalWrite(Orn, LOW);
    delay(50);
}
memoriza_XY();
}

void loop() {
    Valor_X = analogRead(Ace_X);
    delay(1);
    Valor_Y = analogRead(Ace_Y);
    Serie(); // Llama a la función Serie(), para enviar los valores

    // Si excedemos los límites inferior o superior activamos la alarma
    if (Valor_X > (media_X + umbral_X) or Valor_X < (media_X - umbral_X)) {
        alarma();
    }
    if (Valor_Y > (media_Y + umbral_Y) or Valor_Y < (media_Y - umbral_Y)) {
        alarma();
    }
    // Si no tenemos alarma apaga el LED y zumbador
    digitalWrite(RGB_R, LOW);
    noTone(Buz);
}

// Función alarma
void alarma() {
    Serial.println("Alarma"); // Envía la palabra "Alarma"
    digitalWrite(RGB_R, HIGH); // Activa el LED rojo
    tone(Buz, 2500); // Emite una frecuencia de 2500Hz
    delay(100); // Pausa de 100 ms
}
```



```
void memoriza_XY() {
    // Medidas repetidas para hacer una media
    for (int i = 0; i < muestras; i++) {
        Valor_X = analogRead(Ace_X);
        media_X = media_X + Valor_X;
        delay(1); // Espera para la conversión analógica digital
        Valor_Y = analogRead(Ace_Y);
        media_Y = media_Y + Valor_Y;
        delay(1); // Espera para la conversión analógica digital
    }
    // Calcula la media aritmética
    media_Y = media_Y / muestras;
    media_X = media_X / muestras;
    // Envía los valores memorizados
    Serial.println("");
    Serial.print("media X:");
    Serial.print(media_X);
    Serial.print("\t");
    Serial.print("media Y:");
    Serial.print(media_Y);
    Serial.println("");
    analogWrite(Gre, 8); // Enciende el LED verde
    delay(2000); // Espera 2 segundos
    analogWrite(Gre, 32); // Enciende el LED verde con más intensidad
}

// Envía los valores memorizados y los actuales
void Serie() {
    Serial.print("media X:");
    Serial.print(media_X);
    Serial.print("\t");
    Serial.print("Valor X:");
    Serial.print(Valor_X);
    Serial.print("\t");
    Serial.print("media Y:");
    Serial.print(media_Y);
    Serial.print("\t");
}
```



```

Serial.print("Valor Y:");
Serial.print(Valor_Y);
Serial.println("");
}

```

Análisis:

En este ejemplo, además de las estructuras que ya hemos visto en los ejemplos anteriores, volvemos a usar la estructura de control `while ()`. Estos bucles se ejecutan de forma continua e infinita hasta que la expresión dentro del paréntesis (..) se vuelve falsa.

De esta forma, esperamos a que se presione SR con
`while(digitalRead (SR) == 0) { digitalWrite (Orn, HIGH); ... }`

Cuando la condición `(SR) == 0` deja de cumplirse, salimos del bucle `while` y ejecutamos la siguiente instrucción, que llama a la función `memoriza_XY()`.

Dentro de esta función, en un bucle `for`, realizamos varias mediciones de las entradas `Ace_X` y `Ace_Y` para calcular un promedio, almacenándolo en las variables `media_X` y `media_Y`. Estos valores sirven como referencia para comparar con las mediciones actuales. Si la diferencia entre ellas está dentro del margen definido por `umbral_X` y `umbral_Y`, la alarma no se activa, lo que da lugar al juego.

Si quieres hacer el juego más difícil, reduce los valores de `umbral_X` y `umbral_Y` o aumenta el número de muestras en el cálculo del promedio.



```

Saida Monitor Serial x
Mensaje (Ctrl + Enter para enviar mensajem para 'Arduino Nano' em '/dev/ttyUSB0' Nova linha 9600 baud
Pulsa SR para memorizar a posición
media X:352      media Y:348
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:353      media Y:348      Valor Y:348
media X:352      Valor X:353      media Y:348      Valor Y:349
media X:352      Valor X:354      media Y:348      Valor Y:347
media X:352      Valor X:364      media Y:348      Valor Y:350
Alarma
media X:352      Valor X:375      media Y:348      Valor Y:349
Alarma
Ln 1, Col 1 UTF-8 Arduino Nano em /dev/ttyUSB0 2

```



Piano MkMk.

Haremos sonar el zumbador usando las entradas MkMk, como un Mini-Piano ;-)

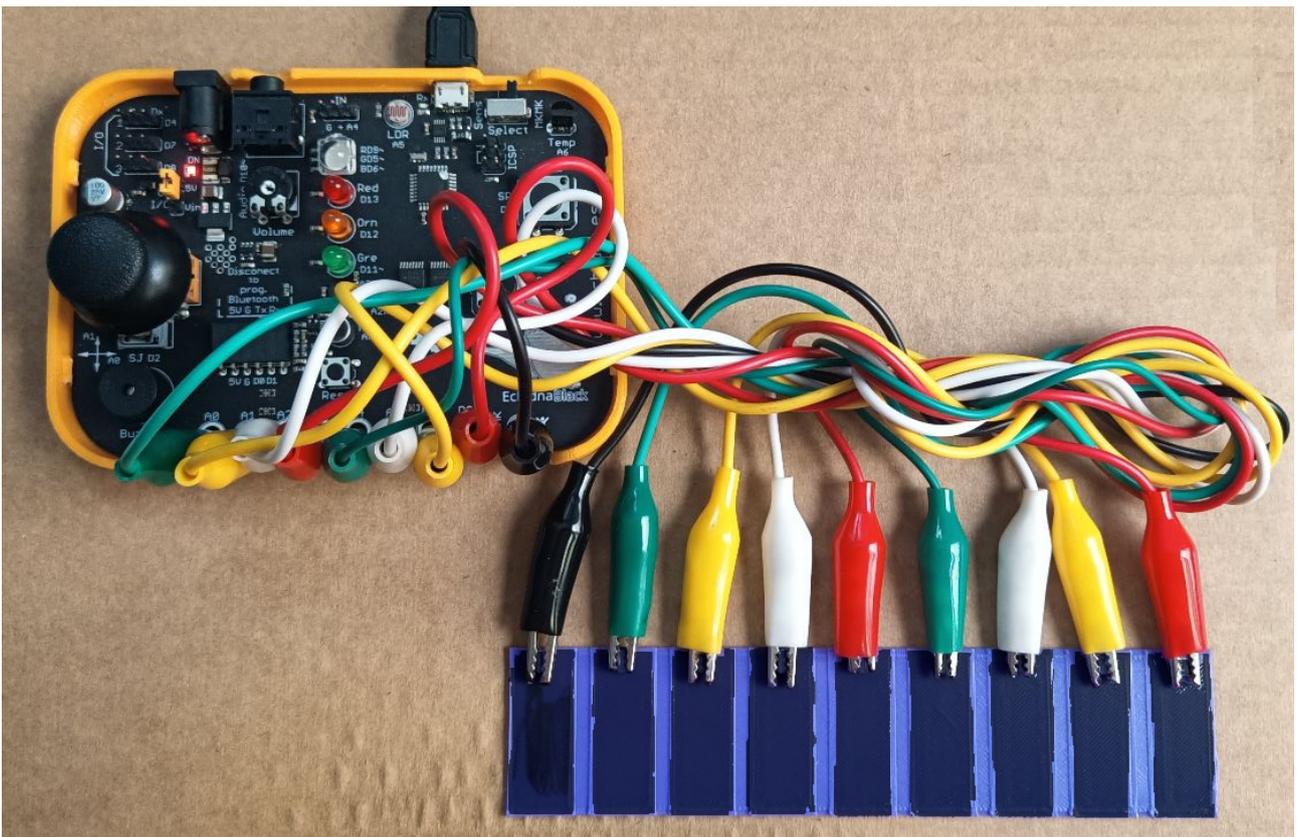
Ahora vamos a jugar un poco en el modo MKMK. Podemos hacer casi todo lo anterior, cambiando los pulsadores por elementos que conduzcan algo de corriente eléctrica, como se comentó en el [“Modo MkMk”](#).

Necesitamos tantos elementos conductivos como teclas queramos, (máximo de 8).

Podemos usar teclas dibujadas con lápiz sobre un papel; cuanto más anchas y gruesas las dibujemos, mejor conducirán la corriente y serán más fáciles de detectar.

También se pueden usar golosinas, ya que la mayoría conducen un poco la corriente.

Conectaremos las “teclas” mediante cables; las pinzas de cocodrilo facilitan mucho la conexión.



Impreso en PLA, teclas pintadas con grafito en spray



```
/* Piano
Seleccionar el modo Mkmk.
Se usan las entradas A0..A5, D2 y D3 como entradas.
D10 será la salida de sonido.
Siempre que cerremos el circuito entre el común y la tecla conectada a la
entrada
Se emite un tono con un tiempo establecido
*/
#define Tiempo 500 //Duración del tono en milisegundos
#define Buz 10 //Zumbador, salida sonido

void setup() {
  pinMode(A0, INPUT); //Modo entradas
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(10, OUTPUT); //Modo salida
}

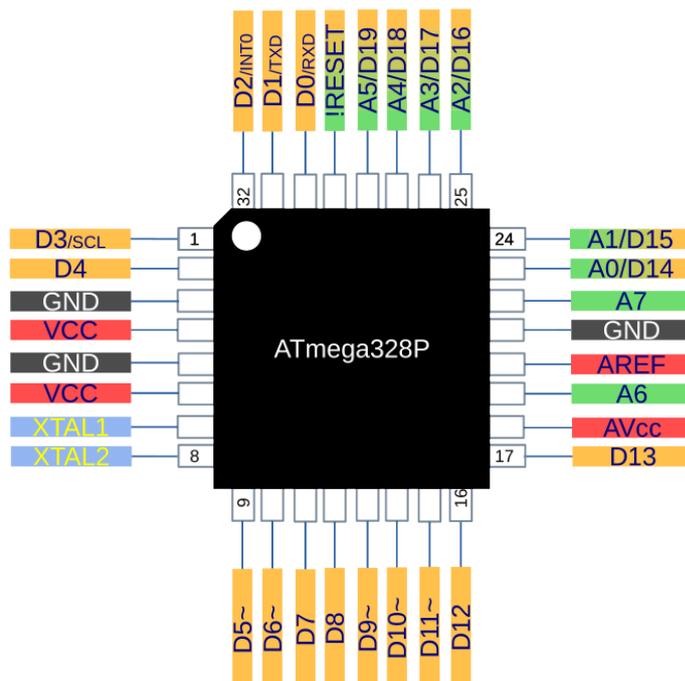
void loop() {
  if (digitalRead(A0) == 1) { //Leemos la entrada A0
    tone(10, 262, Tiempo); //Si es 1 emite un tono
    delay(Tiempo); //Pausa
  }
  if (digitalRead(A1) == 1) { // " A1
    tone(Buz, 294, Tiempo);
    delay(Tiempo);
  }
  if (digitalRead(A2) == 1) { // " A2
    tone(Buz, 330, Tiempo);
    delay(Tiempo);
  }
  if (digitalRead(A3) == 1) {
    tone(Buz, 349, Tiempo);
    delay(Tiempo);
  }
}
```



```
if (digitalRead(A4) == 1) {  
    tone(Buz, 392, Tiempo);  
    delay(Tiempo);  
}  
if (digitalRead(A5) == 1) {  
    tone(Buz, 440, Tiempo);  
    delay(Tiempo);  
}  
if (digitalRead(2) == 1) {  
    tone(Buz, 494, Tiempo);  
    delay(Tiempo);  
}  
if (digitalRead(3) == 1) {  
    tone(Buz, 523, Tiempo);  
    delay(Tiempo);  
}  
}
```

🔑 Análisis:

Estamos realizando una lectura digital de las entradas analógicas, casi todos los pines del Atmega328P pueden leerse digitalmente, salvo A6 y A7, que son exclusivamente entradas analógicas.





Podemos hacer el mismo programa sin necesidad de escribir tantas líneas, utilizando bucles for y matrices.

Piano_2.

```

/* Piano_2
Seleccionar el modo Mkmk.
Se usan las entradas A0..A5, D2 y D3 como entradas.
D10 será la salida de sonido.
Siempre que cerremos el circuito entre el común y la tecla conectada a la
entrada
Se emite un tono con un tiempo establecido
*/
int Mkmk[] = {A0, A1, A2, A3, A4, A5, 2, 3};           //Entradas Mkmk
int Tono[] = {262, 294, 330, 392, 440, 494, 523};     //Frecuencias de los tonos

#define Tiempo 500           //Duración del tono en milisegundos
#define Buz 10               //Zumbador, salida sonido

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(Mkmk[i], INPUT); //Define como entradas las Mkmk
  }
  pinMode(Buz, OUTPUT); //Modo salida
}

void loop() {
  for (int i = 0; i < 8; i++) {           //Recorremos la matriz
    if (digitalRead(Mkmk[i]) == 1) {     //Leemos la entrada Mkmk[]
      tone(Buz, Tono[i], Tiempo);        //Si es 1, emite un tono
      delay(Tiempo);                     //Pausa
    }
  }
}

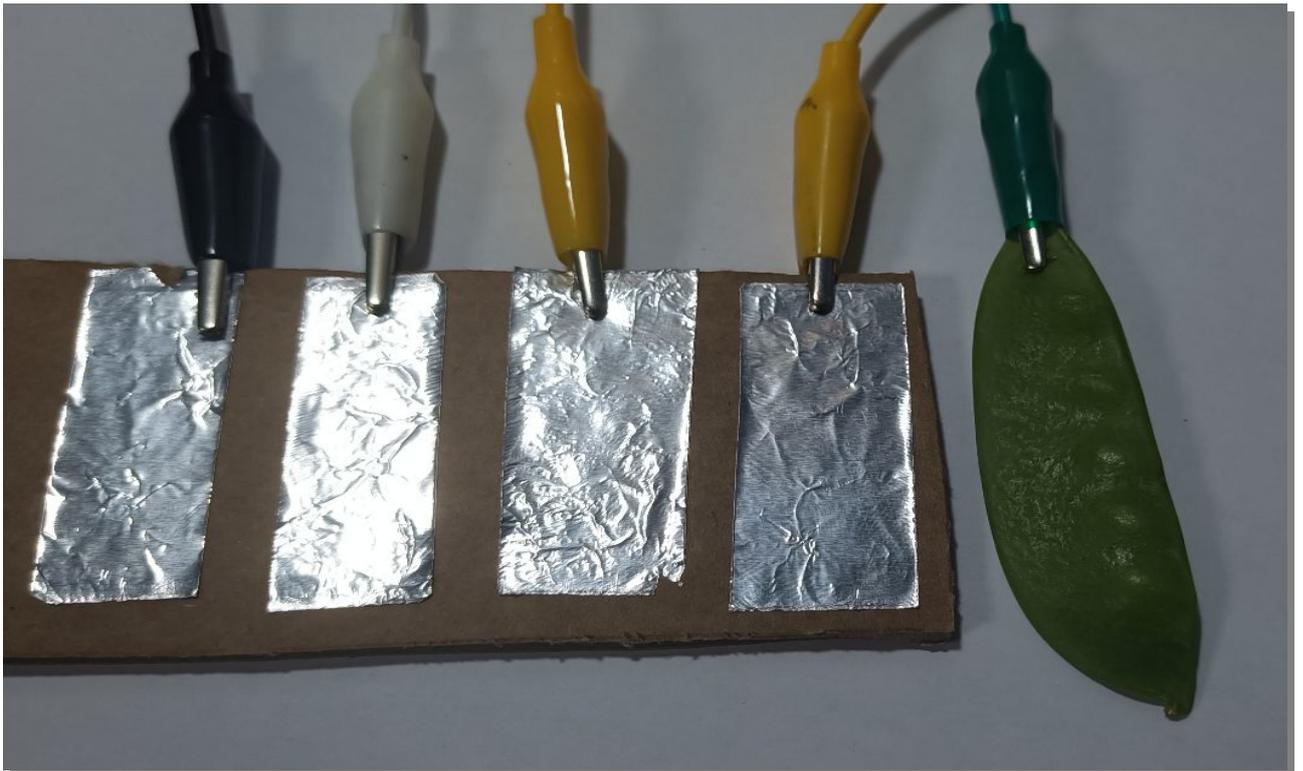
```



✂ Si necesitamos detectar algo que conduzca muy poco, podemos hacer una lectura analógica y comprobar el nivel mínimo de disparo, teniendo diferentes umbrales.

Ejemplo:

```
if (analogRead (MkMk[3]) > umbral { //Leemos la entrada A3
  tone(Buz, Tono[i], Tiempo);      //si es = 1 emite un tono
  delay(Tiempo);                    //pausa
}
```





Encender LEDes desde el ordenador.

Hasta ahora hemos visto cómo enviar datos desde la placa al ordenador, pero el sentido contrario nos permite controlar cosas en la EchidnaBlack mediante la conexión serie desde el ordenador.

```

/* Led Via Serie
   Enciende /apaga un LED mediante la consola serie
   1 = enciende, 2 = apaga
*/
#define RGB_B 6 // Pin del LED azul
char Rec_Ser; // Variable de caracteres de la recepción serie

void setup() {
  pinMode(RGB_B , OUTPUT); // Pin LED como salida
  Serial.begin(115200);
  Serial.println("Escribe un '1' para encender o '2' para apagar");
  digitalWrite(RGB_B , LOW); // Asegúrate de que el LED está apagado al comenzar
}

void loop() {
  if (Serial.available()) { // Comprueba que tenemos conexión establecida
    Rec_Ser = Serial.read(); // Pasa el dato recibido a la variable Rec_Ser
    Serial.println(Rec_Ser); // Reenvía el dato recibido

    if (Rec_Ser == '1') { // Si el carácter recibido es '1' código ASCII 49
      digitalWrite(RGB_B , HIGH); // Enciende el LED
    }
    else if (Rec_Ser == '2') { // Si el carácter recibido es '2' ASCII 50
      digitalWrite(RGB_B , LOW); // Apaga el LED
    }
  }
}

```

Análisis:

Estamos comprobando si el carácter recibido es igual a "1", que corresponde al código ASCII 49. Si queremos ver los códigos ASCII que tecleamos, solo tenemos que cambiar "char Rec_Ser;" por "int Rec_Ser;".



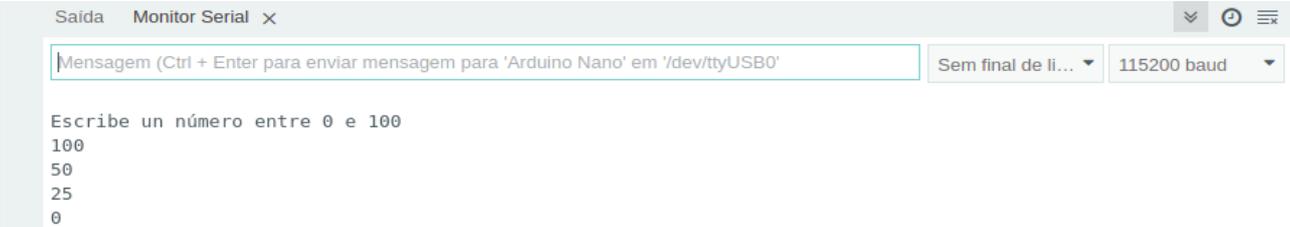
Ajusta la iluminación del LED desde el ordenador.

Ahora queremos controlar la intensidad luminosa de un LED, para ello tendremos que convertir los caracteres enviados en un valor numérico utilizando "**Serial.parseInt()**", que convierte los dígitos recibidos en un dato entero, omitiendo los caracteres iniciales que no sean "-"

```
/* Dim Led Via Serie
   Ajusta % luminosidad LED mediante la consola serie 0 = apagado, 100 = máximo */
#define RGB_G 5 // Pin del LED verde
int Rec_Ser; // Variable de caracteres de la recepción serie
void setup() {
  pinMode(RGB_G, OUTPUT); // Pin LED como salida
  Serial.begin(115200);
  Serial.println("Escribe un número entre 0 e 100");
  digitalWrite(RGB_G, LOW);
}
void loop() {
  if (Serial.available()) { // Comprueba que tenemos conexión establecida
    Rec_Ser = Serial.parseInt(); // Pasa el dato recibido a la variable Rec_Ser
    // Comprueba que o carácter recibido é un salto de línea ('\n')
    // e que o valor recibido de Rec_Ser está entre 0 e 100.
    if (Serial.read() == '\n' && Rec_Ser >= 0 && Rec_Ser <= 100) {
      Serial.println(Rec_Ser); // Reenvia el dato recibido por comprobación
      Rec_Ser = map(Rec_Ser, 0, 100, 0, 255); //Escala los valores de entrada
      analogWrite(RGB_G, Rec_Ser);
    }
  }
}
```

Análisis:

La línea `if (Serial.read() == '\n' && Rec_Ser >= 0 && Rec_Ser <= 100)` asegura que la entrada recibida es válida, es decir, que el valor está en el rango de 0 a 100 y que la entrada termina con un salto de línea (`\n`), lo que indica que se ha completado la entrada.



```
Saída Monitor Serial x
Mensaje (Ctrl + Enter para enviar mensaje para 'Arduino Nano' em '/dev/ttyUSB0') Sem final de li... 115200 baud
Escribe un número entre 0 e 100
100
50
25
0
```



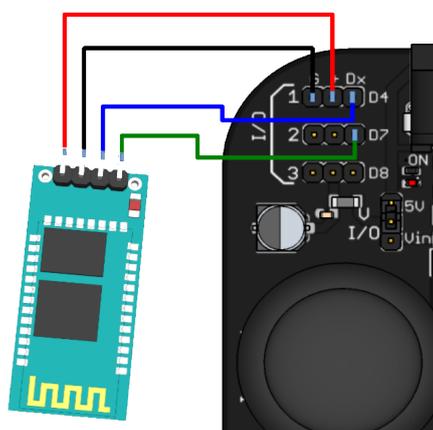
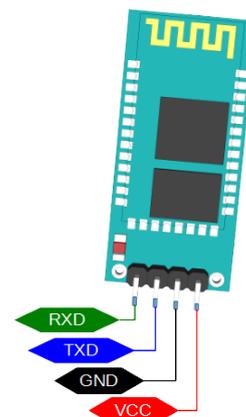
Enciende LEDs desde el Smartphone.

Usando un adaptador Bluetooth HC05 o HC06 y una aplicación, podemos utilizar los dos programas anteriores sin realizar cambios para controlar el LED desde el Smartphone.

El adaptador HC-06 tiene cuatro conexiones: "RXD" para la recepción de datos, "TXD" para la transmisión de datos, "GND" para la masa o negativo de la alimentación, y "VCC" para el positivo de la alimentación.

Es necesario programar el adaptador Bluetooth para ajustarlo a nuestras necesidades.

Configuraremos un nuevo canal serie en dos de los pines libres, utilizando la librería "SoftwareSerial".



| conexi3ns | |
|------------|-----------------|
| Bluetooth | EchidnaBlack |
| VCC | + |
| GND | G |
| TXD | D4 (RXD) |
| RXD | D7 (TXD) |

```
/* SoftwareSerial
```

```
* Abre un canal serie en los pines especificados
* Usamos dos de los de libre disposici3n
* D4 como RXD,
* D7 como TXD
*/
```

```
#include <SoftwareSerial.h> //Librería incluida en el IDE
```

```
#include BaudRate 9600 // Velocidad de comunicaci3n
```

```
SoftwareSerial SoftSerial(4, 7); // Asigna los pines 4 RXD, 7 TXD
```

```
void setup() {
```

```
    Serial.begin(BaudRate); // Inicializa el puerto de comunicaciones
```

```
    Serial.println("Canle serie funcionando"); // Mensaje de cortesía
```

```
    SoftSerial.begin(BaudRate); // Inicializa el puerto de serie-software
```

```
}
```

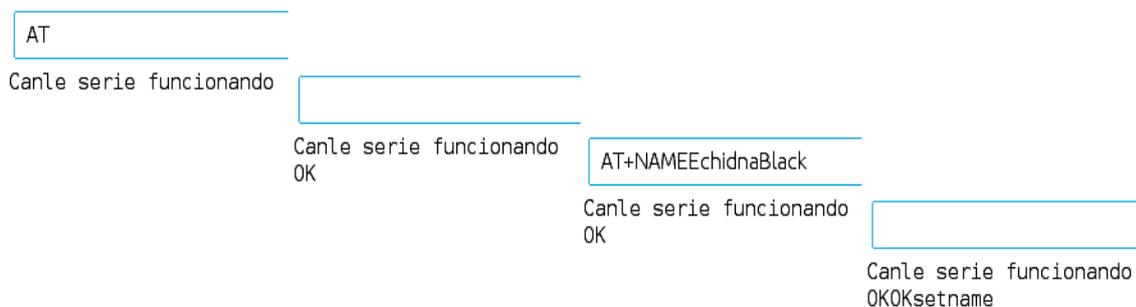


```
void loop() {  
  if (SoftSerial.available()) { // Si tiene datos el SoftSerial los pasa al  
                                // canal serie normal  
    Serial.write(SoftSerial.read());  
  }  
  if (Serial.available()) {  
    SoftSerial.write(Serial.read()); // Si tiene datos los pasa a SoftSer  
  }  
}
```

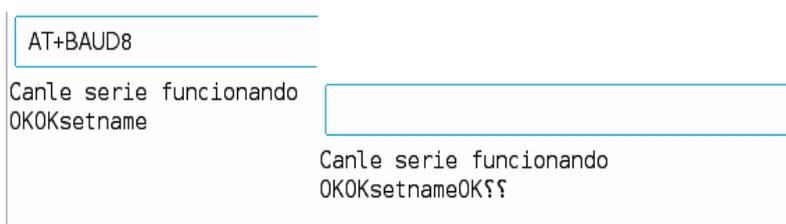
Una vez realizadas las conexiones, el LED del módulo Bluetooth parpadea, indicando que está listo pero no conectado por radio. Este es el modo para poder programarlo.

Enviamos el programa a la placa EchidnaBlack y abrimos el terminal serie (9600 bps), donde se mostrará un mensaje de cortesía "Canal serie funcionando". Para comprobar que tenemos comunicación, enviamos el comando de atención "AT", y debería responder "OK" (si no contesta la primera vez, es posible que el SoftSerial no sea lo suficientemente rápido, así que intenta nuevamente).

Para cambiar el nombre del módulo Bluetooth, enviamos "AT+NAMEnome" y responderá con "OKsetname".



Ahora solo queda programar la velocidad "AT+BAUD8" a 115200 baudios o bps.



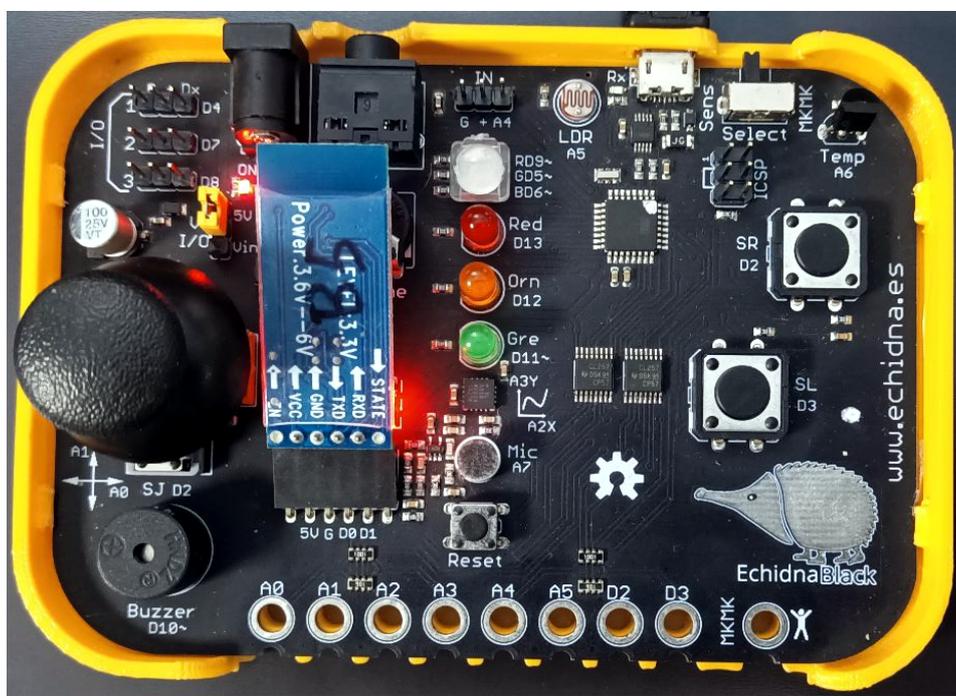
Contesta OK y unos caracteres no reconocibles, esos caracteres son transmitidos a 115200bps, el programa y la consola serie están funcionando a 9600, y no los interpretan bien. Ya tenemos listo el módulo Bluetooth programado, ahora podemos usarlo en la EchidnaBlack.



Comandos AT para HC-06.

| | <i>Envío</i> | - | <i>Respuesta</i> | <i>Ejemplo</i> |
|------------|---------------|---|------------------|-----------------------|
| Test: | AT | | - OK | |
| Nombre: | AT+NAME<NOME> | | - Oksetname | (AT+NAMEEchidnaBlack) |
| Pin: | AT+PIN<Pin> | | - OKsetPIN | (AT+PIN1234) |
| Velocidad: | AT+BAUD<X> | | - OK<baudrate> | (AT+BAUD8) |
| Versión: | AT+VERSION | | - OK<Versión> | OKLinvor1.8 |

X: 1 = 1200, 2 = 2400, 3 = 4800, 4 = 9600, 5 = 19200, 6 = 38400, 7 = 57600, 8 = 115200



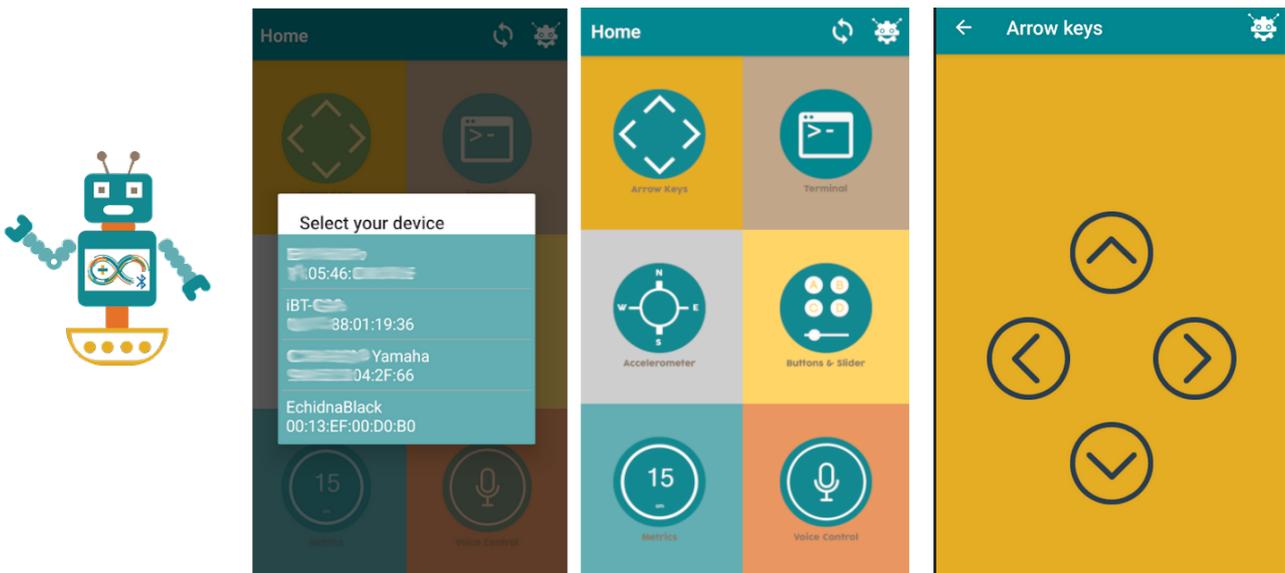
Podemos usar los programas anteriores de control del LED desde el ordenador, ahora usando una app. Recuerda que para subir los programas a la EchidnaBlack es **recomendable** no conectar el módulo Bluetooth hasta que el programa esté subido.

Subimos el programa "Led_Via_Serie". Conectamos el Bluetooth en EchidnaBlack, en el Smartphone vamos al menú [Dispositivos..Bluetooth...] [Vincular nuevo dispositivo] esperamos a que aparezca "EchidnaBlack", lo seleccionamos, ponemos el pin "1234" y aceptamos.

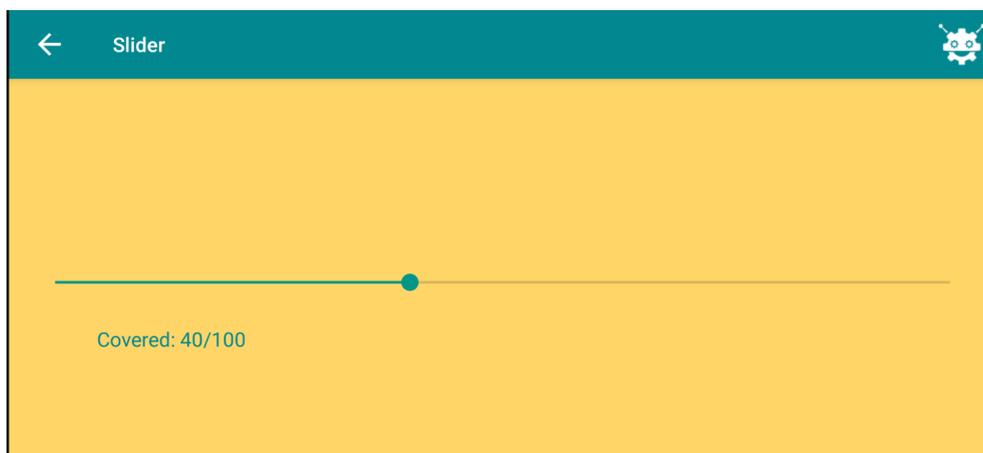


Queda instalar la APP [“Arduino Bluetooth Control”](#) de “broxcodes”. Seleccionando el dispositivo “EchidnaBlack” ya podemos controlar el LED (cuando la APP se conecte al módulo Bluetooth, el LED de este quedará encendido).

Elegimos Arrow keys, la flecha hacia arriba enciende el LED y la flecha hacia abajo lo apaga.



Ahora probamos con el programa “Dim_Led_Via_Serie” y en la APP elegimos [Buttons & Slider], giramos la pantalla y aparece una barra deslizante, que podemos cambiar para ajustar la luminosidad del LED.



Podemos encontrar multitud de aplicaciones que son compatibles, Siempre tendremos que comprobar la velocidad de comunicación.



Si el adaptador que tenemos es un HC-05, este módulo puede tener dos roles: esclavo o maestro, mientras que el HC-06 siempre tiene el rol de esclavo.

El HC-05 tiene seis conexiones:

- "State": estado de comunicación
- "RXD": recepción de datos
- "TXD": transmisión de datos
- "GND": masa o negativo de alimentación
- "VCC": positivo de alimentación
- "EN": habilitación o KEY

Nosotros usaremos las cuatro conexiones centrales.

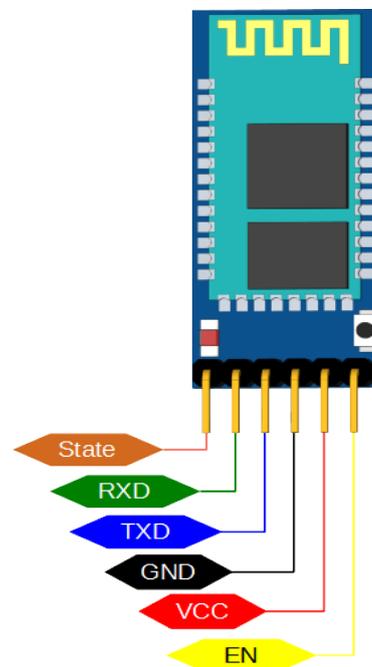
La programación cambia con respecto al HC-06.

Lo primero es ajustar la velocidad "BaudRate" en el programa "SoftwareSerial" a 38400 bps. Antes de encender presionaremos el botón del HC-05 y no lo soltaremos hasta verificar que el LED del HC-05 parpadea lentamente, lo que indica que hemos entrado en modo de programación.

Una vez que subimos el Sketch con la nueva velocidad a la EchidnaBlack, abrimos una terminal, ajustamos la velocidad y elegimos Ambos NL y CR.

- Comenzamos con el comando de atención "AT", lo que responderá "OK".
- Cambiarle el nombre "AT+NAME=EchidnaBlack", responderá "OK".
- Clave de conexión AT+PSWD="1234", responderá "OK".
- Rol como esclavo AT+ROLE=0, responderá "OK".
- Velocidad AT+UART=115200,0,0.
- Reiniciar para establecer los cambios AT+RESET.

- [Manual de comandos para HC-03/05](#)





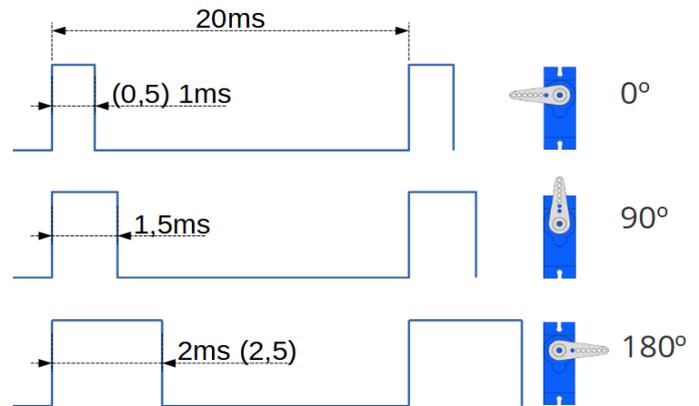
Usando elementos externos.

Servomotor

El “Servo” tiene la capacidad de mantener una posición que enviamos mediante la señal de control.

El rango de giro típico es de 180°, pero también encontraremos servos de rotación continua.

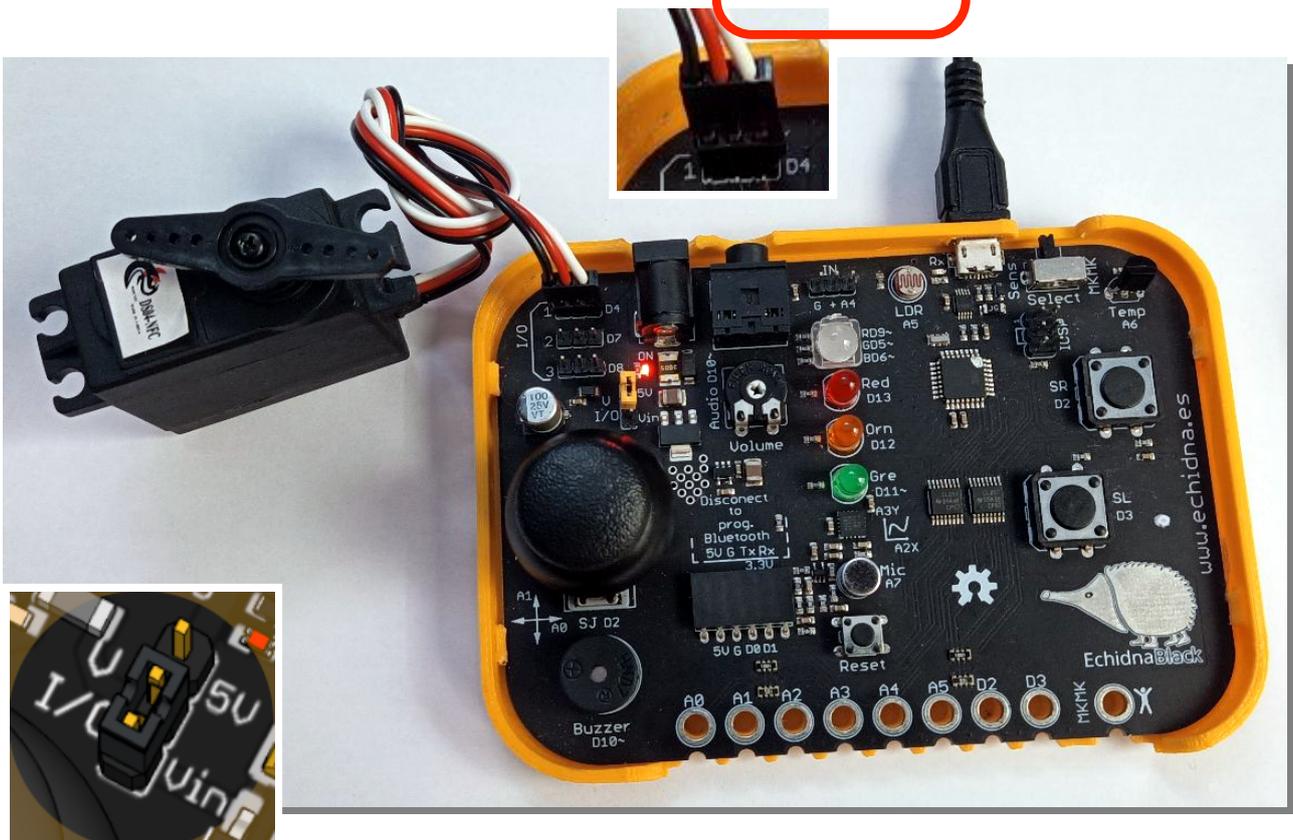
El cable de conexión está formado por tres (3) hilos:



Alimentación (+V)

Negativo (GND)

Señal de control



🦘 Debemos ser conscientes del consumo del (de los) servo(s) que vamos a manejar. Si consume más de 500mA, es recomendable usar una alimentación externa y conectar la fuente de alimentación a Vin (Ver página Selector de Alimentación).



El IDE de Arduino tiene ejemplos de manejo de servos, y con pequeños cambios podemos adaptarlos a nuestras necesidades.

```
/*Servo
```

```
Controlling a servo position
using a potentiometer (variable
resistor)
```

```
by Michal Rinott
```

```
<http://people.interaction-ivrea.it/m.rinott>
```

```
modified on 8 Nov 2013 by
Scott Fitzgerald
```

```
http://www.arduino.cc/en/Tutorial/Knob
```

```
Adaptado a EchidnaBlack
```

```
*/
```

```
// Biblioteca de control del
servo
```

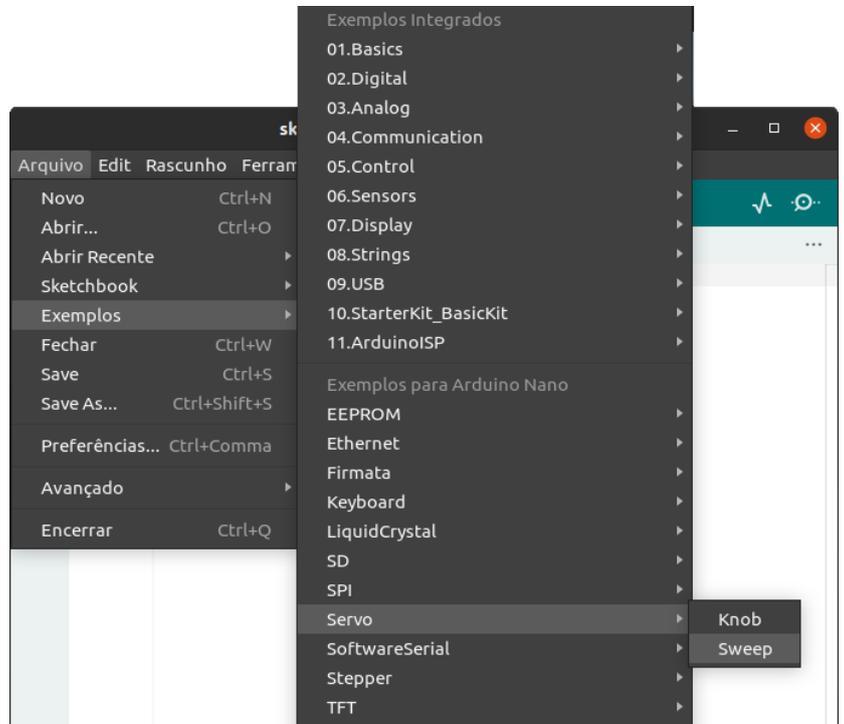
```
#include <Servo.h>
```

```
Servo myservo; // Creamos el objeto myservo para controlar el servo
#define Joy_X A0 // Pin conectado al eje X del Joystick
int val; // Variable para almacenar el valor de lectura del Joystick
```

```
void setup() {
  myservo.attach(4); // Conectamos el pin 4 al objeto myservo
}
void loop() {
  val = analogRead(Joy_X); // Leemos el potenciómetro del Joystick (X)
  val = map(val, 0, 1023, 0, 179); // Escala los valores de entrada
  myservo.write(val); // Envía la posición al Servo
  delay(15); // Una espera para que el servo llegue a la posición
}
```

 Análisis:

Aquí encontramos “`#include <Servo.h>`” que incluye en el programa la librería servo, que facilita el control del servo. Las librerías son programas escritos por otras personas que nos ayudan



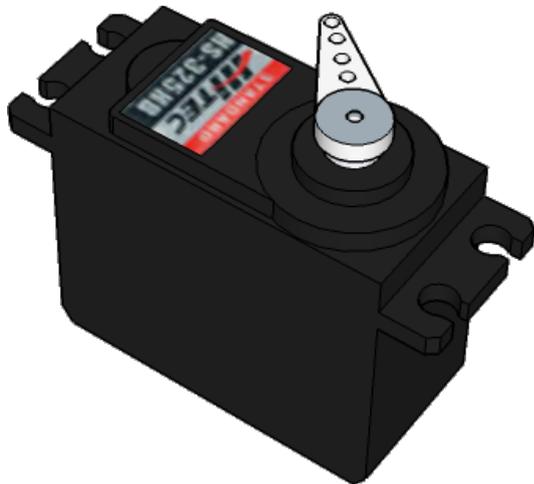


a añadir nuevas funcionalidades a nuestros programas y facilitan la conexión con sensores y actuadores. Más adelante volveremos a hablar de librerías.

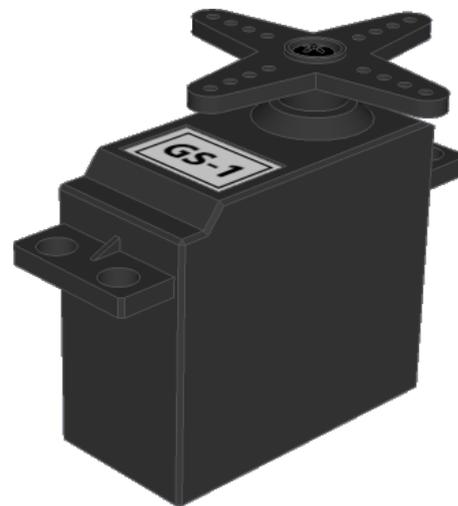
En la instrucción “Servo myservo;” declaramos myservo como un objeto o variable, con el método “myservo.attach(4);” le indicamos en qué pin tenemos conectado el servo, en este caso el pin 4. Realizamos la lectura del eje “X” del Joystick, que nos devuelve un valor de 0 a 1023, Mediante la instrucción “val = map(val, 0, 1023, 0, 179);” escalamos los valores de entrada a los valores en grados que puede manejar el Servo.

El siguiente método que usamos es “myservo.write(val);” donde le pasamos el valor “val” en grados que debe girar.

Para controlar un servo preparado para rotación continua, tenemos que tener en cuenta que quedará parado cuando le enviemos un valor de 90°, girará en sentido horario cuando supere el valor de 90° alcanzando la máxima velocidad cerca de los 179°, y si el valor es inferior a 90° girará en sentido antihorario y cuando se acerque a los 0° alcanzará la máxima velocidad.



0 - 179°



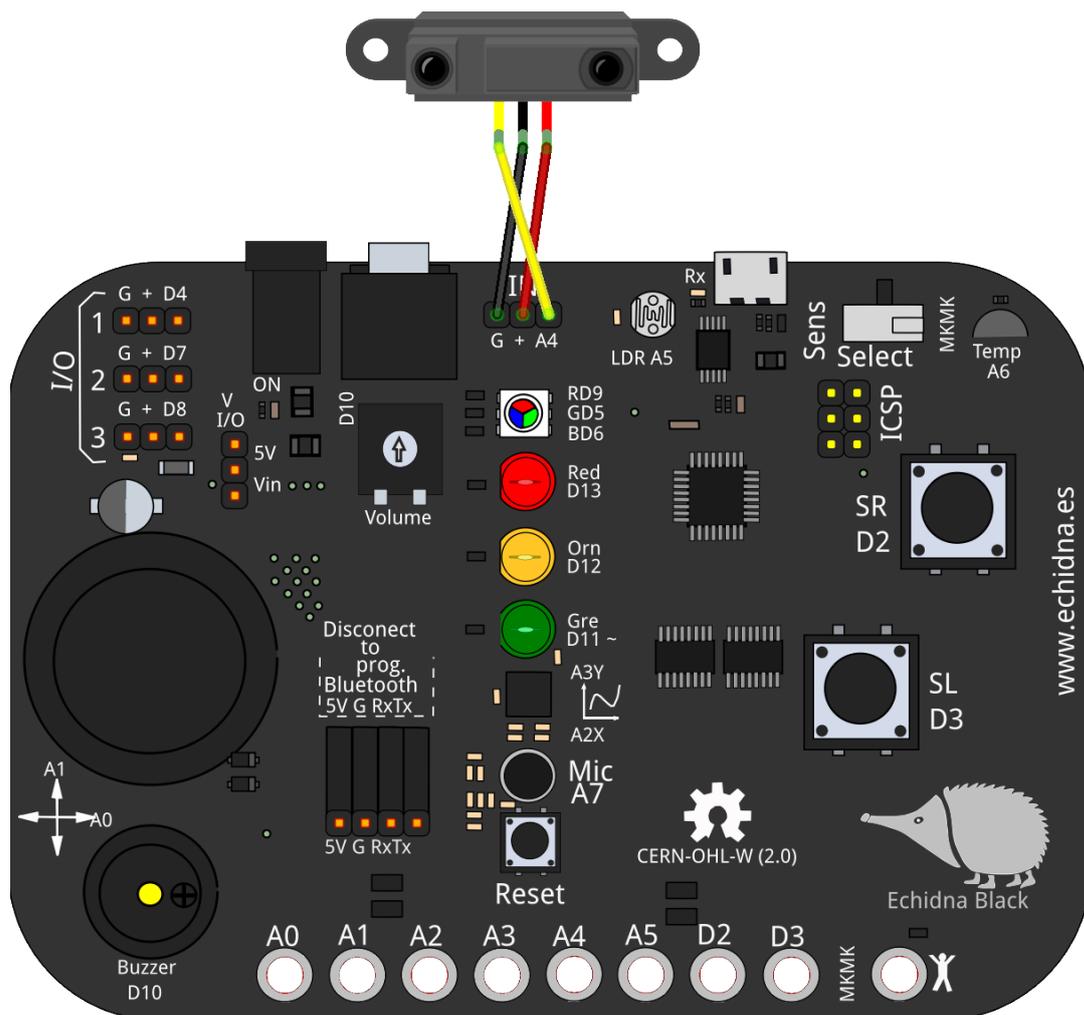
Giro continuo



Medida de distancia.

Sensor de proximidad infrarrojos Sharp 0A41SK

Es un sensor de distancia que proporciona una tensión según el ángulo de rebote de los infrarrojos en una superficie. Tiene un rango de detección de 10 a 80 cm y consume alrededor de 30mA a 5Vcc.





Podemos medir simplemente el valor de tensión de la salida del sensor.

```
/* 0A41SK\_simple
 * Medida directa de los valores de la salida del sensor
 */
#define Sensor A4 //Pin conectado al sensor
int ValSensor = 0; //Variable de lectura

void setup() {
  Serial.begin(9600); //Abrir comunicación serie 9600bps
}

void loop() {
  ValSensor = analogRead(Sensor); //lectura de sensor
  Serial.println(ValSensor); //envío datos vía serie
  delay(10); //pausa para estabilizar
}
```

Usando el Plotter serie comprobamos que tenemos bastante ruido.

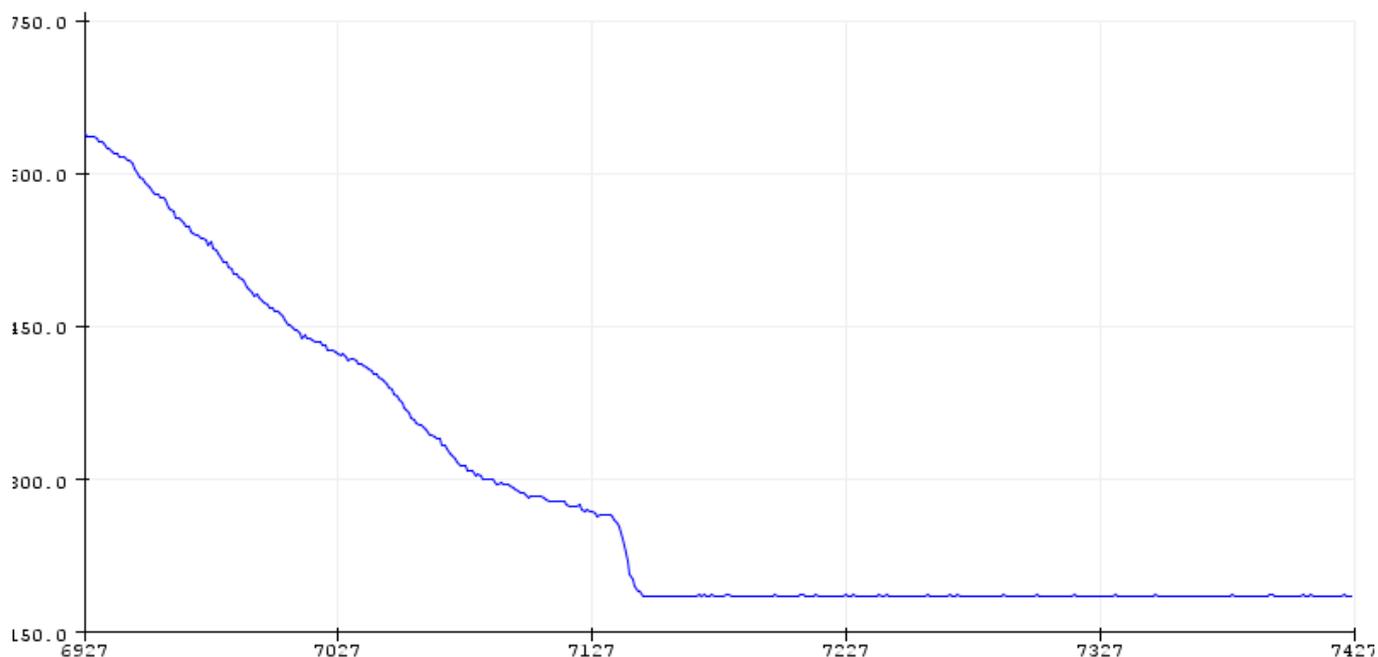




Para intentar evitar esto, podemos calcular un promedio de varias mediciones.

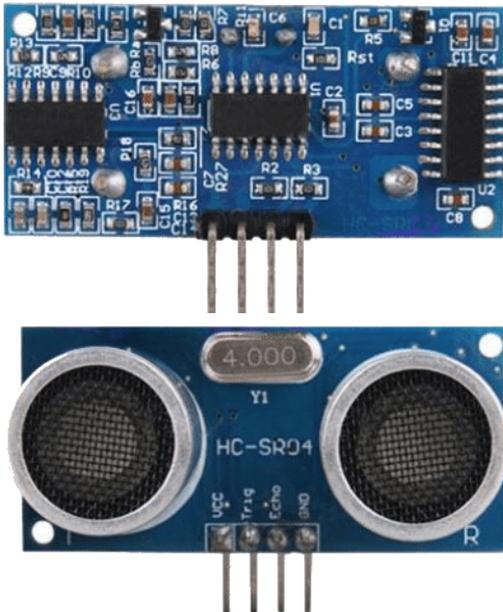
```
/* 0A41SK_media
   Media de los valores de la salida del sensor
*/
#define Sensor A4 //pin conectado al sensor
int ValSensor = 0; //Variable de lectura
int n = 20;        //Numero de lecturas
long Media = 0;    //Variable a acumular medidas
void setup() {
  Serial.begin(9600); //comunicación serie 9600bps
}
void loop() {
  for (int i = 0; i < n; i++) { //Bucle de n medidas
    ValSensor = analogRead(Sensor); //Lectura del sensor
    Media = Media + ValSensor; //acumula medidas
    delay(1); //pausa para estabilizar
  }
  Media = Media/n; //Realiza la media
  Serial.println(Media); //envía la media vía serie
}
```

Podemos observar que la medida ya es más limpia.





Sensor distancia con ultrasonidos HC-SR04



El sensor HC-SR04 utiliza el sonar para determinar la distancia a un objeto, de manera similar a cómo lo hacen los murciélagos o los delfines. Ofrece una excelente detección en un rango de medida de 2 cm a 400 cm. Su funcionamiento no se ve afectado por la luz solar o materiales negros, como ocurre con el sensor Sharp, aunque los materiales acústicamente absorbentes, como una tela, pueden ser difíciles de detectar.

Características:

- Alimentación: +5V DC
 - Corriente en reposo: <2mA
 - Corriente de trabajo: 15 mA
 - Ángulo efectivo: <15°
 - Distancia de alcance: 2 cm – 400 cm
 - Resolución: 0,3 cm
- Ángulo de medición: 30 grados
 - Ancho de pulso en la entrada de disparo: 10µs

Conexiones:

- VCC = +5VDC
- Trig = "Trigger", disparo del sensor
- Echo = "Echo", pulso de salida con ancho proporcional a la distancia
- GND = Masa (-) o negativo de alimentación

Podemos conectar los pines Trig y Echo entre sí para simplificar la conexión con la EchidnaBlack.





```

/* UltraSon
   Medidor de distancias mediante ultrasonidos HC-SR04
   conectado a la entrada/salida IN "A4", como solo tiene un pin,
   unimos os pines "Trigger y ECHO" del HC-SR04 y conectado a IN
*/

#define IN A4 // Pin de conexión
const int Vson = 34300.0; // Velocidad del sonido en cm/s
void setup() {
  Serial.begin(9600); // Inicia comunicación serie
}

void loop() {
  pinMode (IN, OUTPUT); // Define el pin como salida para provocar el disparo
                        // "TRIGGER"
  digitalWrite(IN, LOW); // Trigger en estado bajo y espera 2 ms
  delayMicroseconds(2);
  digitalWrite(IN, HIGH); // Trigger a estado alto y espera 10 ms
  delayMicroseconds(10);
  digitalWrite(IN, LOW); // Trigger en estado bajo.
  pinMode (IN, INPUT); // Define o pin como entrada "ECHO"
  //L a función pulseIn obtiene el tiempo que tarda en cambiar entre estados,
  // en este caso a HIGH
  unsigned long tempo = pulseIn(IN, HIGH);

  // Para obtener la distancia en cm, pasmos el tempo a segundos ya que está en
  // microsegundos por eso multiplicamos por 0.000001 (podemos ajustar este por
  // tolerancias)
  int distancia = tempo * 0.000001 * Vson / 2.0;

  Serial.print(distancia);
  Serial.print("cm");
  Serial.println();
  delay(500);
}

```

Análisis:

El procedimiento de lectura es muy sencillo: Configuramos el pin (A4) como salida y aplicamos un "1" durante 10 ms.



A continuación, configuramos el pin (A4) como entrada y medimos el ancho del pulso que devuelve el HC-SR04. Multiplicando este valor por la velocidad del sonido y dividiéndolo entre 2, obtenemos la distancia a la que rebota la señal.

Si convertimos el tiempo a segundos y usamos la velocidad del sonido en cm/s, obtendremos directamente la distancia en centímetros.



Para los siguientes ejemplos utilizaremos alguna biblioteca para simplificar los programas.

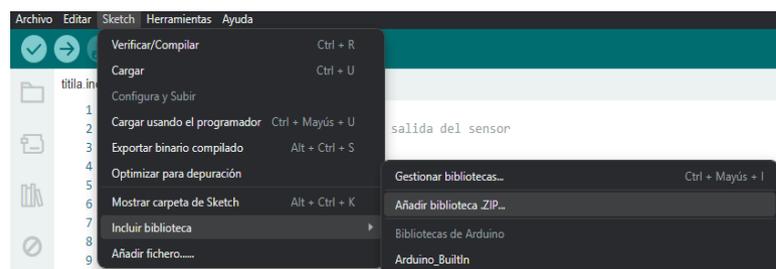
Instalar librerías

El IDE de Arduino viene con muchas bibliotecas preinstaladas. En la página arduino.cc/reference/en/libraries puedes encontrar información sobre cada una de ellas.

Tenemos varias formas de incluir bibliotecas en el IDE de Arduino:

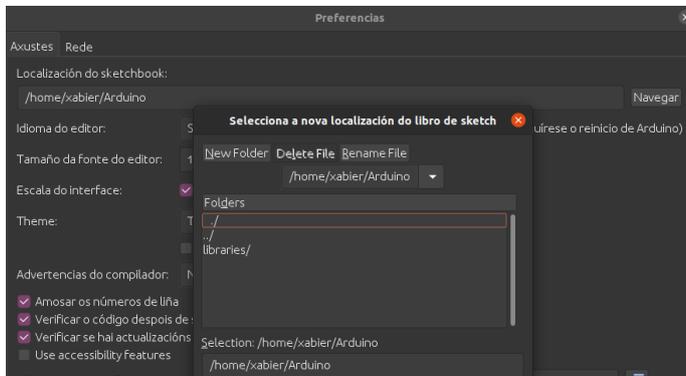
- Usar el icono  o el menú [Herramientas] → [Gestionar bibliotecas], que nos permite instalar nuevas bibliotecas y mantenerlas actualizadas desde los repositorios.
- Para incluir una biblioteca en nuestro programa, utilizamos el menú [Sketch] → [Incluir biblioteca].
- Las bibliotecas no estándar de Arduino deben instalarse manualmente. Para hacerlo, existen varias formas.

Podemos incluirla desde un archivo “.zip”.

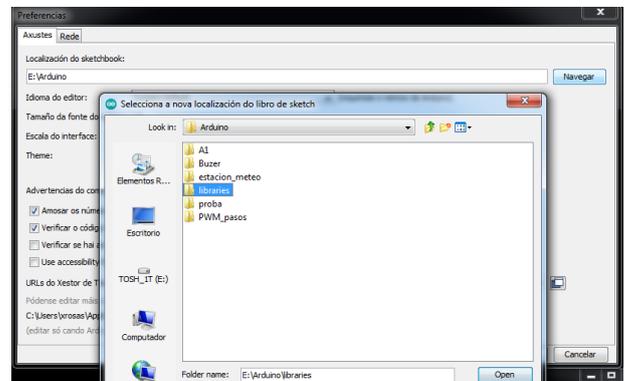




Otra forma para versiones anteriores al IDE 2.x.x es totalmente manual: descargamos la librería y la descomprimos dentro de la carpeta de librerías del IDE Arduino. La localización de esa carpeta se puede encontrar en las preferencias de Arduino.



GNU Linux

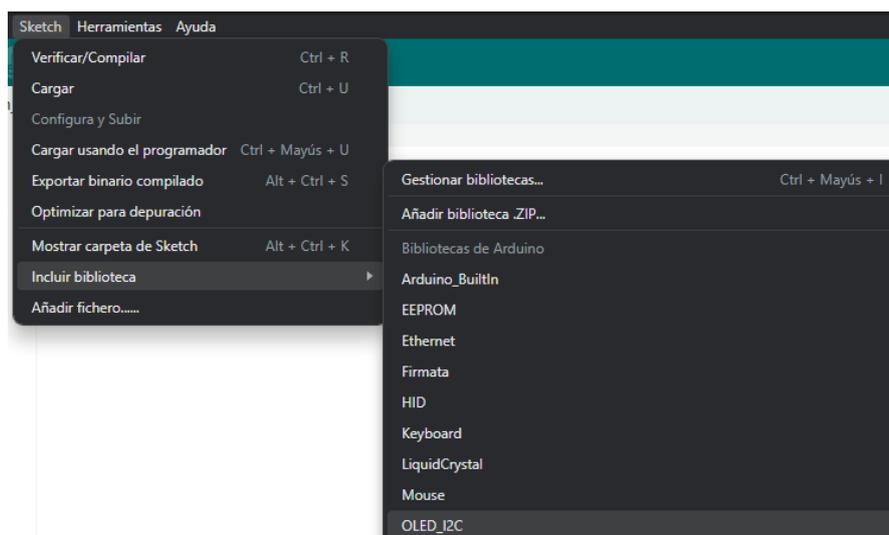


Windows

Una vez descomprimida la librería, debemos reiniciar el IDE de Arduino para que la librería aparezca en el menú.

Recuerda que no todas las librerías son eficaces; no siempre se depuran, se mantienen con el tiempo ni son fáciles de usar. Lo normal es que tengan varios ejemplos de uso.

No debemos tener librerías duplicadas instaladas, ya que al compilar el programa, el IDE seleccionará la primera que aparezca en el listado de preferences.txt, y puede ser que no sea la que queremos.



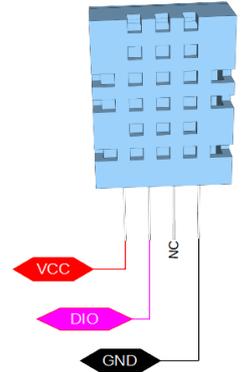
Si tenemos dudas, podemos visitar los foros de Arduino para buscar información.



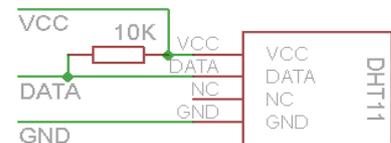
Medida del % de humedad y temperatura con el DHT-11.

El DHT11 es un sensor de bajo costo utilizado para medir la temperatura (en un rango de 0 a 50 grados centígrados con una precisión de $\pm 2^{\circ}\text{C}$) y la humedad (en un rango de 20% a 80% con una precisión de $\pm 5\%$). Se alimenta con un voltaje de 3 a 5V.

Para la medición de la temperatura, tiene un termistor NTC resistivo.



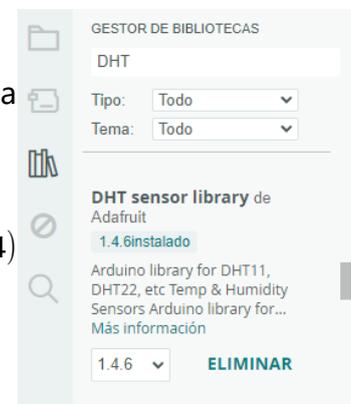
Para conectar el DHT11 a Echidna Black, necesitamos asegurarnos de que la salida de datos esté conectada a una resistencia "Pull-Up". Esto se puede hacer de manera sencilla utilizando la instrucción "pinMode(Pin_Sen, INPUT_PULLUP);", que activa automáticamente una resistencia Pull-up interna en el pin que estemos utilizando para leer los datos del sensor.



Usaremos a librería DHT de Adafruit, a instalaremos como ya vimos,

Recuerda instalar todo lo necesario, ya que esta librería depende de Adafruit_Sensor.h.

En este ejemplo, vamos a utilizar el pin de entrada/salida (D4) para conectar el sensor



```
/* DHT\_HT
```

```
Lectura del % de humedad y temperatura con el sensor DHT11
```

```
Basado en el programa escrito por Ladyada (Limor Fried) de Adafruit
```

```
Necesitamos las librerías DHT-sensor-library y Adafruit_Sensor
```

```
*/
```

```
#include "DHT.h"
```

```
#define Sensor DHT11 // Tipo de sensor, también podemos usar DHT22 que es mejor
```

```
#define Pin_Sen 4 // Pin conectado al sensor
```

```
DHT dht(Pin_Sen, Sensor); // Declaramos el objeto dht
```

```
int h; // Variable para almacenar el % de la humedad
```

```
int t; // Variable para la temperatura
```



```
void setup() {
  pinMode(Pin_Sen, INPUT_PULLUP); // Activamos la resistencia interna Pull up
  Serial.begin(9600); // Inicia las comunicaciones
  Serial.println("¡Test DHT11!");
  dht.begin(); // Iniciamos el sensor
}

void loop() {
  delay(2000); // Esperamos dos segundos entre medidas, es un sensor lento
  h = dht.readHumidity(); // Leemos el % de humedad
  t = dht.readTemperature(); // Leemos la temperatura
  // Comprobamos si alguna medida falla y lo intentamos otra vez
  if (isnan(h) || isnan(t)) {
    Serial.println("¡Fallo de lectura del DHT!");
    return;
  }
  int hic = dht.computeHeatIndex(t, h, false);
  // Devuelve el valor en °C. (Fahrenheit = false)
  Serial.print(F("Humedad: ")); // Envía los valores vía serie
  Serial.print(h);
  Serial.print("% Temperatura: ");
  Serial.print(t);
  Serial.println(F("°C "));
}
```

Análisis:

El sensor utilizado no es el mejor, pero es uno de los más accesibles y lo encontraremos en muchas montajes con Arduino. El DHT22 tiene  mayor rango y mayor resolución.

Este es un claro ejemplo de lo que ahorramos usando una buena librería.

Solo iniciamos el sensor y le pedimos las medidas.



```
Salida Monitor Serie x
Mensaje (Intro para mandar el mensaje de
Humedad: 43% Temperatura: 23°C
```



Sensor capacitivo de humedad del terreno.

Para medir la humedad del suelo existen múltiples técnicas (Resistiva, reflectometría en el dominio de la frecuencia y sensores de neutrones...), en este caso vamos a usar un sensor capacitivo. Cuanto mayor sea el contenido de agua en el suelo, mayor será la capacitancia y menor será la tensión de salida del sensor. Este sensor está basado en el famoso temporizador NE555.

```
/* Sensor_Humedad_Capacitivo
 * Lectura de la humedad del suelo
 * Memoriza dos puntos, uno seco y otro húmedo,
 */

#define Sensor A4 // Entrada analógica para conectar el sensor
#define SR 2 // Entrada Pulsador SR para memorizar valores (calibrar)
#define L_Red 13 // LED rojo (Seco)
#define L_Org 12 // LED naranja (Normal)
#define L_Gre 11 // LED verde (Húmedo)
#define RGB_B 6 // LED azul (bomba funcionando)
#define Rele 4 // Salida I/O 4 relé para la bomba de riego
#define Buz 10 // Buzzer (Zumbador)

int ValSensor; // Variable para el valor de lectura del sensor
int VSeco = 510; // Valor inicial para seco
int VHumi = 300; // Valor inicial para húmedo
double Media; // Valor medio de las muestras del sensor
int n = 512; // Número de lecturas para el valor medio

void setup() {
    Serial.begin(9600); // Inicializa las comunicaciones
    pinMode(Sensor, INPUT); // Configura las entradas/salidas
    pinMode(SR, INPUT);
    pinMode(L_Red, OUTPUT);
    pinMode(L_Org, OUTPUT);
    pinMode(L_Gre, OUTPUT);
    pinMode(RGB_B, OUTPUT);
    pinMode(Rele, OUTPUT);
    pinMode(Buz, OUTPUT);
}
```





```
void loop() {
  // Realiza la media aritmética de las medidas de humedad
  calcularMedia();
  Serial.println(Media);
  if (Media >= VSeco) { // El valor indica humedad muy baja
    encenderRojo(); // Enciende el LED rojo
    encenderRele(); // Activa el relé y el LED azul
  }
  if (Media > VHumi && Media < VSeco) { // Valor de humedad correcto
    encenderNaranja(); // Enciende el LED naranja
  }
  if (Media <= VHumi) { // Valor de humedad alta
    encenderVerde(); // Enciende el LED Verde
    apagarRele(); // Apaga el relé y el LED azul
  }
  // Lectura del pulsador SR para entrar en el menú de memorizar valores
  if (digitalRead(SR) == 1) {
    tone(Buz, 4000, 200); // Beep
    memorizar(); // Llama a la función memorizar
  }
}

// Función para actualizar los valores de seco y húmedo
void memorizar() {
  apagarRele(); // Apaga el relé y el LED azul mientras se calibra
  // Memoriza el valor de la tierra seca
  Serial.println("Coloca el sensor en tierra seca");
  Serial.println("Cuando esté listo pulsa SR");
  delay(500);
  while (digitalRead(SR) == 0) { // Mientras no se pulse SR
    digitalWrite(L_Red, !digitalRead(L_Red)); // Parpadeo del LED rojo
    delay(50);
  }
  tone(Buz, 2300, 200); // Beep, se pulsó SR
  calcularMedia(); // Calcula la media de n medidas
  VSeco = Media; // Asigna el valor actual como tierra seca
  digitalWrite(L_Red, 0);
  mostrar(); // Muestra los valores memorizados hasta el momento
}
```



```
// Memoriza el valor de la tierra húmeda
Serial.println();
Serial.println("Coloca el sensor en tierra húmeda");
Serial.println("Cuando esté listo pulsa SR");
delay(500);
while (digitalRead(SR) == 0) { // Mientras no se pulse SR
    digitalWrite(L_Gre, !digitalRead(L_Gre)); // Parpadeo del LED verde
    delay(50);
}
tone(Buz, 2300, 200); // Beep
calcularMedia();
VHumi = Media; // Asigna el valor actual como tierra húmeda
while (digitalRead(SR) == 1) {} // Espera que suelte el pulsador SR
digitalWrite(L_Gre, 0); // Apaga el LED Verde
// Si los valores memorizados están mal, volvemos a la lectura de los valores
if (VHumi >= VSeco) {
    tone(Buz, 200, 1000); // Beep error
    Serial.println("Error");
    Serial.println("¡Vuelve a calibrar el sensor!");
    memorizar(); // Volvemos a la función memorizar (no falles mucho)
}
}

// Cálculo de la media de la medida del sensor
void calcularMedia() {
    for (int i = 0; i < n; i++) { // Bucle de n medidas
        ValSensor = analogRead(Sensor); // Lectura del sensor
        Media = Media + ValSensor; // Acumula las medidas
        delay(1);
    }
    Media = (int)(Media / n); // Realiza la media de las n medidas y elimina los
    decimales
}

// Enciende el Relé y el LED azul
void encenderRele() {
    digitalWrite(Rele, 1);
    digitalWrite(RGB_B, 1);
}
```



```
// Apaga el Relé y el LED azul
void apagarRele() {
    digitalWrite(Rele, 0);
    digitalWrite(RGB_B, 0);
}

// Enciende el LED rojo apaga los otros
void encenderRojo() {
    digitalWrite(L_Red, 1);
    digitalWrite(L_Org, 0);
    digitalWrite(L_Gre, 0);
}

// Enciende el LED naranja apaga los otros
void encenderNaranja() {
    digitalWrite(L_Red, 0);
    digitalWrite(L_Org, 1);
    digitalWrite(L_Gre, 0);
}

// Enciende el LED verde apaga los otros
void encenderVerde() {
    digitalWrite(L_Red, 0);
    digitalWrite(L_Org, 0);
    digitalWrite(L_Gre, 1);
}

// Muestra los valores memorizados hasta el momento.
void mostrar() {
    Serial.print("VSeco = ");
    Serial.print(VSeco);
    Serial.print(" VHumi = ");
    Serial.print(VHumi);
    Serial.println();
}
```



🔍 Análisis:

En el programa "sketch", además de la definición y ajuste de los roles de entrada/salida, nos encontramos con tres comparaciones para determinar si activamos la bomba de riego.

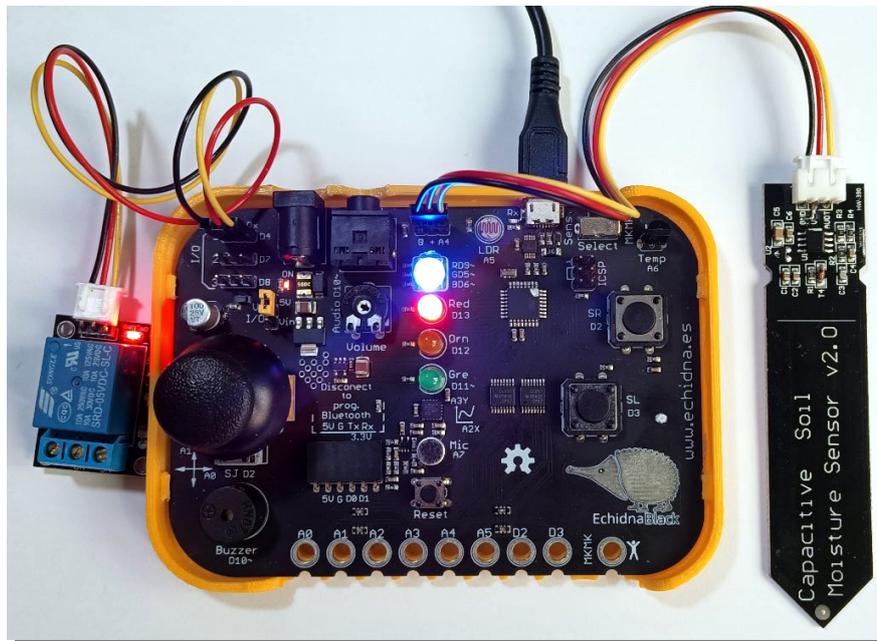
Cuando la media aritmética de las lecturas del sensor de humedad capacitivo es mayor que el Valor Seco, se activará la bomba señalizando mediante el LED azul, y permanecerá activa hasta que se detecte que el valor del sensor es inferior al umbral húmedo.

Si el valor de la medida está entre ambos valores, se encenderá el LED naranja.

Un examen cíclico en el "loop" comprueba la pulsación del SR (beep), lo que nos llevará a la función de memorizar (calibrar) nuevos valores de tierra seca y húmeda. Realiza un cambio rápido de estado del LED rojo para indicar que debemos colocar el sensor en tierra seca; esta es una forma muy simplificada de cambiar el estado de una salida, consiste en poner la salida en el estado negado "!digitalRead(L_Red)" de lo que tenía previamente y esperar un pequeño tiempo.

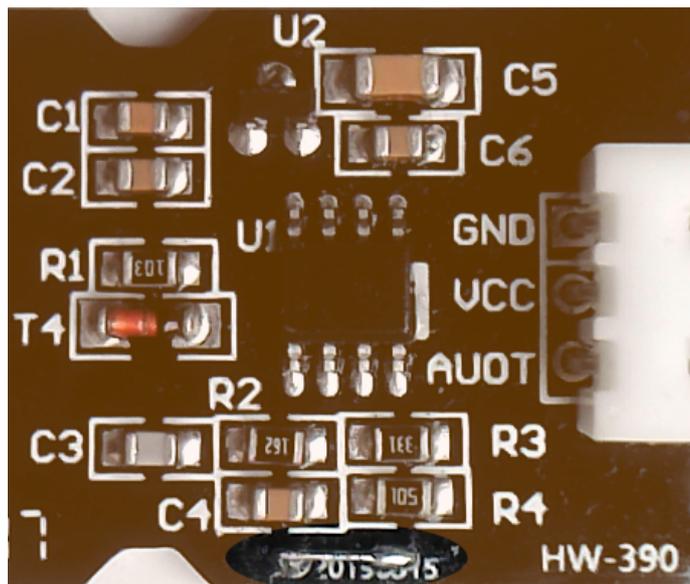
Al pulsar "SR", y antes de salir de esta función, se realiza una comprobación de los valores; si el valor húmedo es mayor o igual que el valor seco, indicaría que no tenemos mediciones correctas, y nos devolvería a la calibración con un beep de error.

Las funciones de activación de los LEDs, relé, etc., están fuera del bucle "loop" para mayor claridad.





Este sketch presenta un inconveniente: si apagamos, perdemos los valores de calibración. Para solucionarlo, podemos guardar los valores en la memoria EEPROM del microcontrolador. Así que vamos a dedicar un rato a aprender a usar esta memoria no volátil y hacer una nueva versión del programa anterior, que sea capaz de guardar los datos límites de calibración del sensor capacitivo.



Nota: algunos sensores tienen un error de fabricación, falta la conexión de la resistencia R4 a GND, lo que provoca que la respuesta sea muy lenta.

Un simple puente soluciona el error



EEPROM

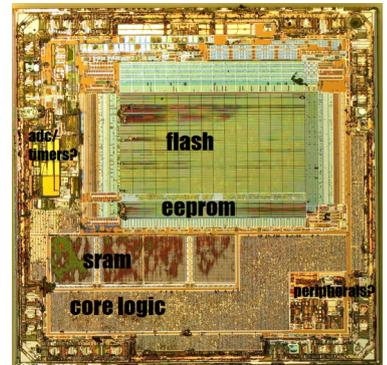
Las siglas vienen de “Electrically Erasable Programmable Read-Only Memory”. Es una memoria que se puede escribir y borrar eléctricamente pierden cuando se apaga, reteniéndolos de forma permanente.

El Atmega328P dispone en esta memoria de 1024 bytes (1 byte una memoria rápida, tarda unos 3,3 ms en escribir cada celda). Puede ser escrita/borrada unas 100.000 veces, no podemos usarla como RAM.

Por defecto, la memoria aparece con el valor 255 (FF), indicando que no se ha usado.

Usamos la librería EEPROM, que ya viene instalada en el IDE de

```
#include <EEPROM.h>
```



ATmega328p Imax. u/Ryancor

Las funciones más básicas son “read” y “write”, que pueden leer y escribir un byte (0 a 255) en la dirección indicada (0 a 1023):

Sintaxis:

```
EEPROM.read(EMemor); // Lee un byte en la dirección especificada
```

```
EEPROM.write(dirección, dato); // Escribe en la dirección el dato (byte)
```

Si queremos escribir algo más grande que un byte, tenemos:

```
EEPROM.put(Dirección, Dato); //Escribe en la dirección el dato (byte, int, float...)
```

Cuidado al usar `EEPROM.put()`, tendremos en cuenta el tamaño de los datos en bytes que almacenamos para calcular la dirección del siguiente valor.

```
EEPROM.get(Dirección, Dato); // Lee en la dirección un dato de cualquier tipo.
```

Cuidado, tenemos que almacenarlo con el tipo de dato correcto.

Actualizar un dato solo escribe el dato si es diferente al almacenado, esto es muy útil para no gastar ciclos de escritura de EEPROM:

```
EEPROM.update(dirección, dato); // Solo almacena datos tipo byte.
```

Si necesitamos comprobar la longitud de una variable, podemos usar:

```
longitud = sizeof(variable); // Para el caso de una variable “int” devolverá 2.
```



Sensor capacitivo de humedad con Limites en EEPROM.

```

/* Humid_Capacitivo_EEPROM
   Lectura de la humedad de la tierra
   Memoriza dos puntos: uno seco y otro húmedo
*/
#include <EEPROM.h> // Librería para manejar la memoria EEPROM
#define Sensor A4 // Entrada analógica para conectar el sensor
#define SR 2 // Entrada Pulsador SR para memorizar valores
(calibrar)
#define L_Red 13 // LED rojo (Seco)
#define L_Org 12 // LED naranja (Normal)
#define L_Gre 11 // LED verde (Húmedo)
#define RGB_B 6 // LED azul (bomba funcionando)
#define Rele 4 // Salida I/O 4 relé para la bomba
#define Buz 10 // Buzzer (Zumbador)

int ValSensor; // Variable para Valor de lectura del sensor
int VSeco = 510; // Valor inicial para seco
int VHumi = 300; // Valor inicial para húmedo
double Media; // Valor medio de las muestras del sensor
int n = 512; // Número de lecturas para el valor medio
byte Memor; // Si es 255 indicará que no se grabó en la EEPROM
int EMemor = 0; // Dirección con el dato de comprobación
int EVSeco = 2; // Dirección para guardar/leer el valor para tierra seca
int EVHumi = 4; // Dirección para guardar/leer el valor para tierra húmeda
void setup() {
  Serial.begin(9600); // Inicializa las comunicaciones
  pinMode(Sensor, INPUT); // Configura las entradas/salidas
  pinMode(SR, INPUT);
  pinMode(L_Red, OUTPUT);
  pinMode(L_Org, OUTPUT);
  pinMode(L_Gre, OUTPUT);
  pinMode(RGB_B, OUTPUT);
  pinMode(Rele, OUTPUT);
  pinMode(Buz, OUTPUT);

  if (EEPROM.read(EMemor) != 255) { // Comprueba si valores actualizados en la EEPROM

```



```
EEPROM.get(EVSeco, VSeco);    // Pasa los valores a las variables
EEPROM.get(EVHumi, VHumi);
Serial.print("Valores almacenados"); // Envía los valores almacenados vía serie
visu();
}
}
void loop() {
// Realiza la media aritmética de las medidas de humedad
media();
Serial.println(Media);
if (Media >= VSeco) {    // El valor indica humedad muy baja
  Vermello();          // Enciende el LED rojo
  Rele_On();           // Activa el relé y el LED azul
}
if (Media > VHumi && Media < VSeco) { // Valor de humedad correcto
  Laranja();           // Enciende el LED naranja
}
if (Media <= VHumi) {   // Valor de humedad alta
  Verde();              // Enciende el LED verde
  Rele_Off();           // Apaga el relé y el LED azul
}
// Lectura del pulsador SR para entrar en el menú de memorizar valores
if (digitalRead(SR) == 1) {
  tone(Buz, 4000, 200); // Beep
  memoriza(); // Llama a la función memoriza
}
}
void memoriza() { // Función para actualizar los valores de seco y húmedo
  Rele_Off(); // Apaga el relé mientras se calibra
// Memoriza el valor de la tierra seca
Serial.println("Pincha el sensor en tierra seca");
Serial.println("Cuando esté listo pulsa SR");
delay(500);
while (digitalRead(SR) == 0) { // Mientras no se pulse SR
  digitalWrite(L_Red, !digitalRead(L_Red)); // Parpadeo del LED rojo
  delay(50);
}
}
```



```

tone(Buz, 2300, 200); // Beep, se pulsó SR
media();              // Calcula la media de n medidas
VSeco = Media;        // Asigna el valor actual como tierra seca
digitalWrite(L_Red, 0);
visu(); // Muestra los valores memorizados hasta el momento
// Memoriza el valor de la tierra húmeda
Serial.println();
Serial.println("Pincha el sensor en tierra húmeda");
Serial.println("Cuando esté listo pulsa SR");
delay(500);
while (digitalRead(SR) == 0) { // Mientras no se pulse SR
    digitalWrite(L_Gre, !digitalRead(L_Gre)); // Parpadeo del LED verde
    delay(50);
}
tone(Buz, 2300, 200); // Beep
media();
VHumi = Media;        // Asigna el valor actual como tierra húmeda
while (digitalRead(SR) == 1) {} // Espera a que se suelte el pulsador SR
digitalWrite(L_Gre, 0); // Apaga el LED verde
// Si los valores memorizados están mal, volvemos a la lectura de los valores
if (VHumi >= VSeco) {
    tone(Buz, 200, 1000); // Beep error
    Serial.println("Error");
    Serial.println("Vuelve a calibrar el sensor!");
    memoriza(); // Volvemos a la función memoriza (no lo uses demasiado)
}
EEPROM.get(EMemor, Memor); // Recuperamos el valor de las veces que se graba
Memor = Memor - 1;         // Decrementamos el valor
EEPROM.update(EMemor, Memor); // Actualizamos el valor en la memoria EEPROM
EEPROM.put(EVSeco, VSeco); // Actualizamos el valor Seco en la memoria EEPROM
EEPROM.put(EVHumi, VHumi); // Actualizamos el valor Húmedo en la memoria
                           // EEPROM
Serial.print(" Datos almacenados "); // Pasamos los valores... vía serie.
Serial.print(255 - Memor);
Serial.println(" veces");
visu();
}

```



```
// Cálculo de la media de la medida del sensor
void media() {
    for (int i = 0; i < n; i++) { // Bucle de n medidas
        ValSensor = analogRead(Sensor); // Lectura del sensor
        Media = Media + ValSensor; // Acumula las medidas
        delay(1);
    }
    Media = (int)(Media / n); // Hace la media de las n medidas y elimina los decimales
}

// Enciende el Relé y el LED azul
void Rele_On() {
    digitalWrite(Rele, 1);
    digitalWrite(RGB_B, 1);
}
// Apaga el Relé y el LED azul
void Rele_Off() {
    digitalWrite(Rele, 0);
    digitalWrite(RGB_B, 0);
}
// Enciende el LED rojo y apaga los otros
void Vermello() {
    digitalWrite(L_Red, 1);
    digitalWrite(L_Org, 0);
    digitalWrite(L_Gre, 0);
}
// Enciende el LED naranja y apaga los otros
void Laranxa() {
    digitalWrite(L_Red, 0);
    digitalWrite(L_Org, 1);
    digitalWrite(L_Gre, 0);
}
// Enciende el LED verde y apaga los otros
void Verde() {
    digitalWrite(L_Red, 0);
    digitalWrite(L_Org, 0);
    digitalWrite(L_Gre, 1);
}
```



```
// Muestra los valores memorizados hasta el momento.
void visu() {
  Serial.println();
  Serial.print("VSeco = ");
  Serial.print(VSeco);
  Serial.print("  VHumi = ");
  Serial.print(VHumi);
  Serial.println();
}
```

Análisis:

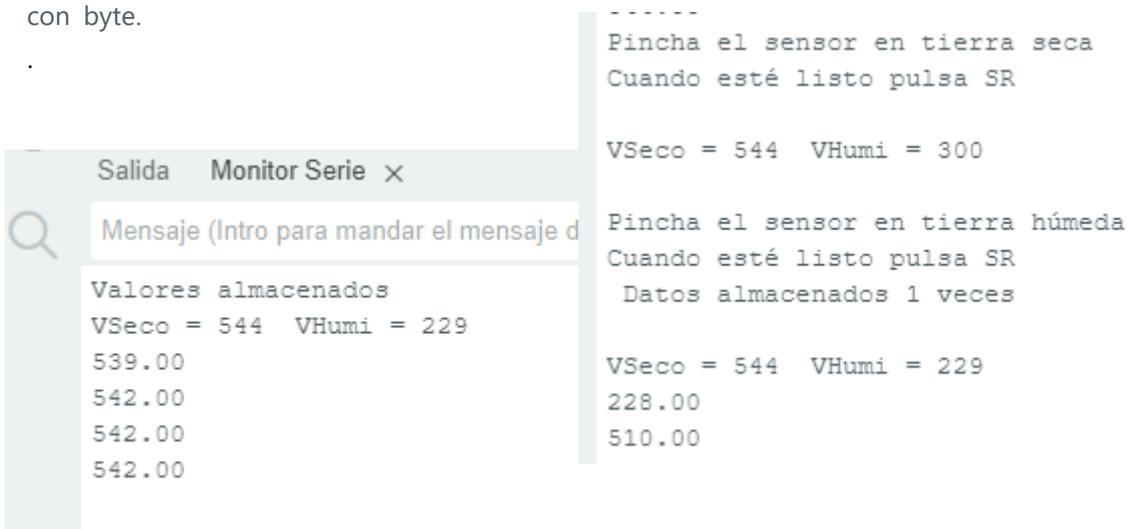
El primero es comprobar si se hicieron cambios. Para eso tenemos una variable “Memor” que nos dice cuántas veces se grabaron los datos. Comienza en 255, indicando que no se grabaron datos.

Si se hicieron grabaciones y necesitamos recuperar los datos, comprobamos que la variable es diferente de 255 con `if (EEPROM.read(EMemor) != 255)`. Si es diferente, recuperamos los datos y los pasamos a las variables con `EEPROM.get(EMemor, Memor)`;

Para grabar los datos de calibración, una vez que se comprueba que no hay errores, tomamos el valor “Memor” y lo decrementamos en una unidad, actualizando el valor en la memoria con `EEPROM.update(EMemor, Memor)`;

Ahora se pasan los datos de los límites VSeco y VHumi a la memoria con:
`EEPROM.put(EVSeco, VSeco)`;

No podemos usar `EEPROM.update()`; ya que estamos grabando variables int y update solo funciona con byte.



```
-----
Pincha el sensor en tierra seca
Cuando esté listo pulsa SR

VSeco = 544  VHumi = 300

Pincha el sensor en tierra húmeda
Cuando esté listo pulsa SR
  Datos almacenados 1 veces

VSeco = 544  VHumi = 229
228.00
510.00

Salida  Monitor Serie x
Mensaje (Intro para mandar el mensaje d
Valores almacenados
VSeco = 544  VHumi = 229
539.00
542.00
542.00
542.00
```



Receptor de infrarojos.

Los receptores IR de Vishay (entre otros) se utilizan para aplicaciones de control remoto y comunicación por infrarrojos, en repetidores/almacenamiento de código infrarrojo, barreras inmatrimales infrarrojas y sensores de proximidad infrarrojos. Los receptores IR funcionan en el rango de longitud de onda de 840 nm a 960 nm. Cuentan con un filtro para minimizar los efectos de la luz ambiental."

<https://www.vishay.com/docs/82667/tsdp341.pdf>

Los mandos de control remoto utilizan diversos protocolos de emisión, los más habituales son los de Philips RC5, RC6, NEC, Sony.

Podemos usar la librería "IRremote" que es una de las más usadas y es compatible con la mayoría de los protocolos de los mandos.

Buscamos la librería "IRremote" y la instalamos.

En el menú Sketch ya podemos encontrar la librería para usarla.

Actualmente es compatible con los siguientes protocolos:

Denon/Sharp, JVC, LG/LG2, NEC/Onkyo/Apple, Panasonic/Kaseikyo, RC5, RC6, Samsung, Sony, BoseWave, Lego, Whynter, MagiQuest.

Autor: shirriff, z3t0, ArminJo.

Mantenedor: Armin Joachimsmeier

Repositorio:

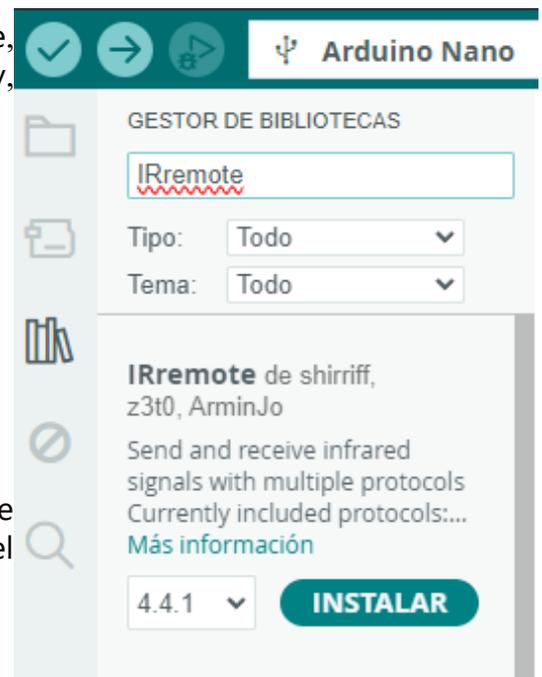
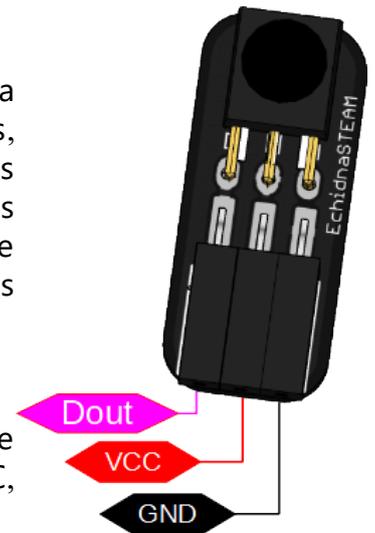
<https://github.com/Arduino-IRremote/Arduino-IRremote>

Vamos a hacer un pequeño programa que encienda/apague un LED con cualquier tecla del control remoto (compatible con muchos mandos).

```
/* IR\_LED
 * Encender/apagar un LED al recibir
 * datos infrarrojos de un mando a distancia
 */

#include <IRremote.h> // librería instalada

#define PIN_IR A4 // pin para recibir los datos
#define LED_ROJO 13 // LED a controlar
```





```

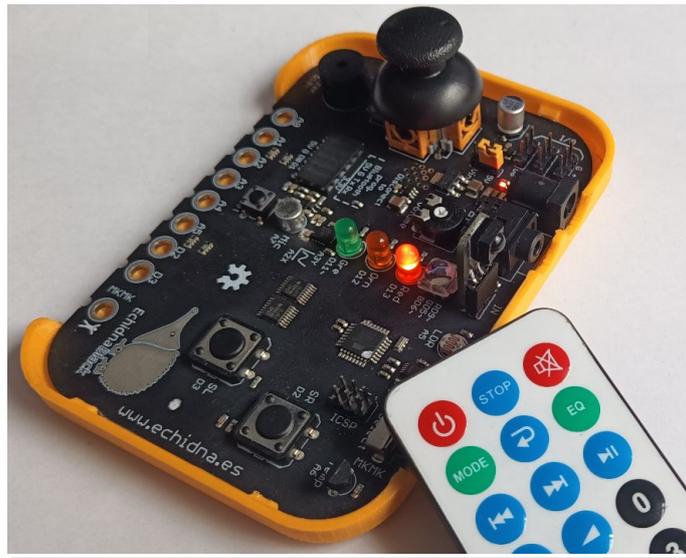
IRrecv Receptor_ir(PIN_IR); // variable de recepción es asignada al pin
decode_results resultado; // variable con el resultado de la recepción

boolean estado = 0;

void setup(){
  Receptor_ir.enableIRIn(); // Habilitamos la recepción IR
  pinMode(LED_ROJO, OUTPUT); // Modo salida
}

void loop() {
  if (Receptor_ir.decode(&resultado)) {
    estado = !estado; // Alternamos el estado
    digitalWrite(LED_ROJO, estado); // Ponemos el valor del estado en el led
    Receptor_ir.resume(); // Preparamos para una nueva recepción.
  }
  delay(300);
}

```



Análisis:

Este ejemplo es otra muestra de lo sencillo que es usar una librería. Se asigna el pin del receptor y la variable de recepción, usamos una variable binaria “estado” que conmutamos cada vez que recibimos un dato por infrarrojos, en la línea “estado= !estado;” haciendo que si estado tenía un “0” ahora tendrá un “1”, ese valor lo pasamos al estado del LED.

Ahora que ya tenemos un receptor podemos ampliar el programa para recoger los distintos códigos del mando, para poder utilizar EchidnaBlack y un emisor de infrarrojos como un Control Remoto.



IR_Receptor

```
/* IR\_Receptor
Receptor de Infrarrojos. Muestra el Protocolo y los Datos Recibidos
Basado en el Programa ReceiveDump de Armin Joachimsmeier*/

#include <IRremote.h> //Librería

#define PIN_RECEPTOR A4 //Pin Asignado al Receptor

IRrecv receptor_ir(PIN_RECEPTOR);

void setup(){
  Serial.begin(9600); // Iniciamos las Comunicaciones Serie
  receptor_ir.enableIRIn(); // Inicia la Recepción IR
  Serial.println("Receptor Listo ");
}

void loop() {
  if (IrReceiver.decode()) { //Si Tenemos Recepción Entramos en la Estructura Switch
    digitalWrite (LED_ROJO, HIGH); //Enciende el Led Cuando Recibe Datos
    Serial.print("Protocolo ");
    switch (IrReceiver.decodedIRData.protocol) { //Descodifica el Protocolo
      case NEC: // En el Caso de Recibir Datos con el Protocolo NEC
        Serial.print("NEC: "); // Envía Serie "NEC"
        break;
      case SONY:
        Serial.print("SONY: ");
        break;
      case SAMSUNG:
        Serial.print("SAMSUNG: ");
        break;
      case RC5:
        Serial.print("RC5: ");
        break;
      case RC6:
        Serial.print("RC6: ");
        break;
      case UNKNOWN:
        Serial.print("Desconocido: ");
        break;
    }
    //Envía los Datos Vía Serie
    Serial.print ("Dirección=0x");
    Serial.print ( IrReceiver.decodedIRData.address, HEX);
    Serial.print (" Comando=0x");
  }
}
```



```
Serial.println ( IrReceiver.decodedIRData.command, HEX);
delay(200); //Espera 200ms
irrecv.resume(); //Limpia el Buffer para Atender Otra Recepción
digitalWrite (LED_ROJO, LOW); //Apaga el Led al Finalizar la Recepción
}
}
```

🔑 Análisis:

Esperamos en “`IrReceiver.decode()`” a que lleguen datos por infrarrojos, y entramos en una estructura `switch..case`. Del mismo modo que `if`, `switch..case` controla el flujo del programa. En este caso en la instrucción `switch` compara el valor “`IrReceiver.decodedIRData.protocol`” con lo especificado en cada “`case`”, y ejecuta el contenido hasta la instrucción “`break`”.

Admite también un “`default`” que se usa para ejecutar un bloque en el caso de que ninguna de las condiciones anteriores se cumpla.

Una vez que se envía el tipo de protocolo, adjunta la dirección y el comando en hexadecimal.

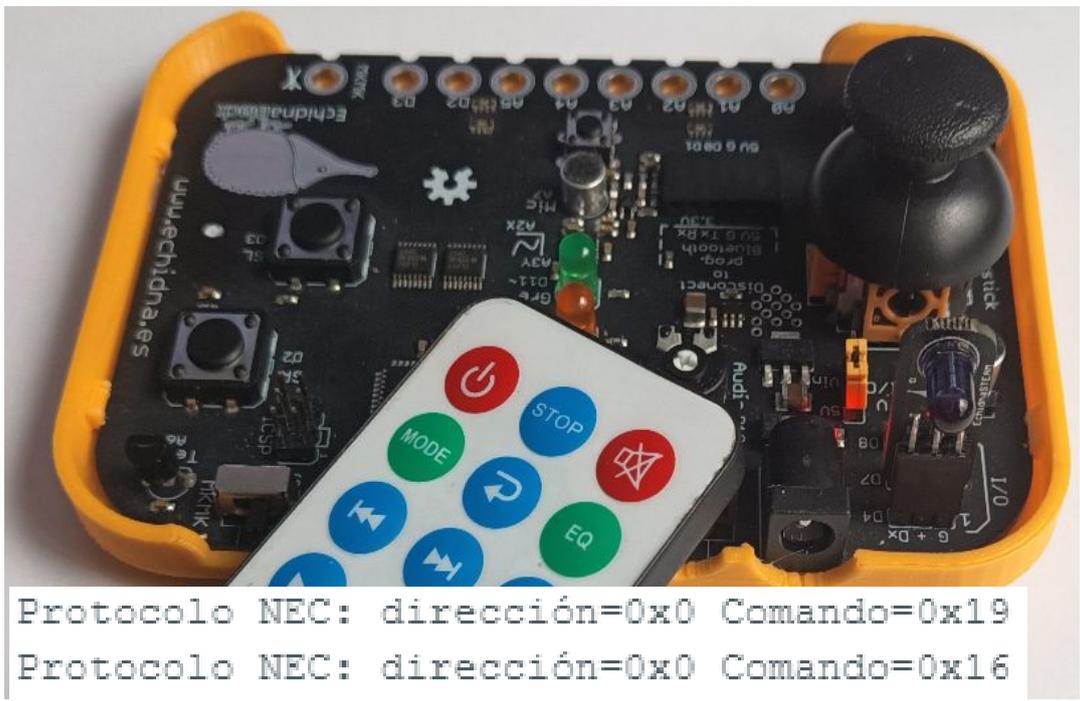
```
Receptor listo
Protocolo NEC: dirección=0x78 Comando=0x12
Protocolo SAMSUNG: dirección=0x7 Comando=0xD
Protocolo NEC: dirección=0x78 Comando=0x2
Protocolo SAMSUNG: dirección=0x7 Comando=0x11
Protocolo SAMSUNG: dirección=0x7 Comando=0x2
```

Emisor de Infrarrojos.

Usando un led infrarrojo TSAL4400 de Vishay u otro compatible, podemos realizar un control remoto (mando a distancia), para poder controlar el aparato que necesitemos, vamos a hacer un pequeño ejemplo para tener el control del volumen.

www.vishay.com/docs/81006/tsal4400.pdf

Usando el programa anterior IR_Receptor, podemos comprobar la dirección y el comando de las teclas del control remoto que queremos emular.



```
/* IR\_Emisor
   Un mando a distancia para el equipo de audio.
   El pulsador SR envía comandos de subir volumen, SL bajar.
*/
#define IR_SEND_PIN 4 //Pin del emisor de Ir
#include <IRremote.h>
#define L_Red 13 //LED para indicar que emite
#define SR 2 //Pulsador para enviar subir volumen
#define SL 3 //Pulsador para bajar volumen
int direccion = 0x0; //endereço do mando-receptor
byte subeVol = 0x19; //comando subir volumen
byte vajaVol = 0x16; //comando baixar volumen
byte comando ;
```



```

void setup() {
  pinMode(L_Red, OUTPUT); // LED
  Serial.begin(9600);
  IrSender.begin(); //Inicializa el envío de datos Ir
  Serial.print("Listo para enviar comandos vía Ir, en el pin ");
  Serial.println(IR_SEND_PIN);
}
void loop() {
  if ((digitalRead (SR)) == 1) { //Si SR está pulsado sube volumen
    comando = subeVol;
    envia();    //Llama a la función enviar datos
  }
  if ((digitalRead (SL)) == 1) { //Si SR está pulsado baja volumen
    comando = vajaVol;
    envia();
  }
}
// función para presentar datos vía serie y vía Ir
void envia() {
  Serial.println();
  Serial.print("Enviando: dirección=0x");
  Serial.print(direccion, HEX);
  Serial.print(" comando=0x");
  Serial.print(comando, HEX);
  Serial.println();
  IrSender.sendNEC(direccion, comando,0); //envía datos Ir
  delay(200);
}

```

Análisis:

Una vez que tenemos los valores de la dirección 0x0, e identificados los comandos a emitir subeVol = 0x19 y vajaVol = 0x16, ya podemos decidir cuándo enviamos cada comando. Usamos dos estructuras `if`, una para cada pulsador, haciendo que el comando tenga el valor que nos interesa. Usando la función “`IrSender.sendNEC(dirección, comando, 0);`” enviamos los datos. No olvidar inicializar la emisión IR con “`IrSender.begin();`” en la función “`void setup()`”.



Detectando campos magnéticos.

En este montaje vamos a utilizar el sensor de Efecto Hall SS39ET de la empresa Honeywell. Estos sensores entregan una tensión en la salida que varía en función del campo magnético aplicado.

Valores característicos: 1,4mV/Gauss y 2,50V a 0 Gauss, es decir, la salida tendrá un valor mayor de 2,5V (512) cuando detecte un campo magnético norte, y será menor cuando el campo sea sur.

<https://www.farnell.com/datasheets/2007294.pdf>

```
/*HALL_Proximidad
```

```
Detectando la proximidad de un campo magnético
```

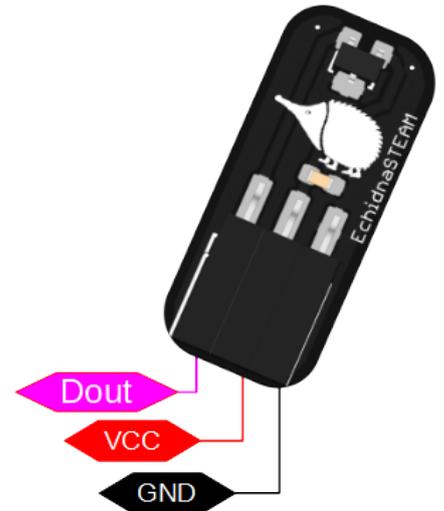
```
Encendiendo un par de LEDs como testigos
```

```
Enciende el LED verde con la intensidad del campo detectado
```

```
#define HALL A4 // Sensor Efecto HALL conectado a A4  
#define L_Red 13 // LED rojo en D13  
#define L_Orn 12 // LED naranja en D12  
#define L_Gre 11 // LED verde en D11  
int umbralN = 550; // Valor activación hacia el norte  
int umbralS = 500; // Valor activación hacia el sur  
int ValorHALL;  
int ValorMap;
```

```
void setup() {  
  Serial.begin(9600);  
  pinMode(HALL, INPUT); // Definimos como entrada  
  pinMode(L_Red, OUTPUT); // Definimos modo salida  
  pinMode(L_Orn, OUTPUT); // Definimos modo salida  
  pinMode(L_Gre, OUTPUT); // Definimos modo salida  
}
```

```
void loop() {  
  ValorHALL = analogRead(HALL); // Leemos la entrada  
  
  // Comprobar si el valor supera el umbralN  
  if (ValorHALL > umbralN) {  
    digitalWrite(L_Red, HIGH); // Enciende el LED rojo  
  }  
  // Si no supera el umbral norte, apaga el LED  
  else {  
    digitalWrite(L_Red, LOW); // Apaga el LED rojo  
  }  
}
```





```
// Comprobar si el valor es inferior a umbralS
if (ValorHALL < umbralS) {
  digitalWrite(L_Orn, HIGH); // Enciende el LED naranja
}
// En el caso contrario, apaga el LED naranja
else {
  digitalWrite(L_Orn, LOW); // Apaga el LED
}

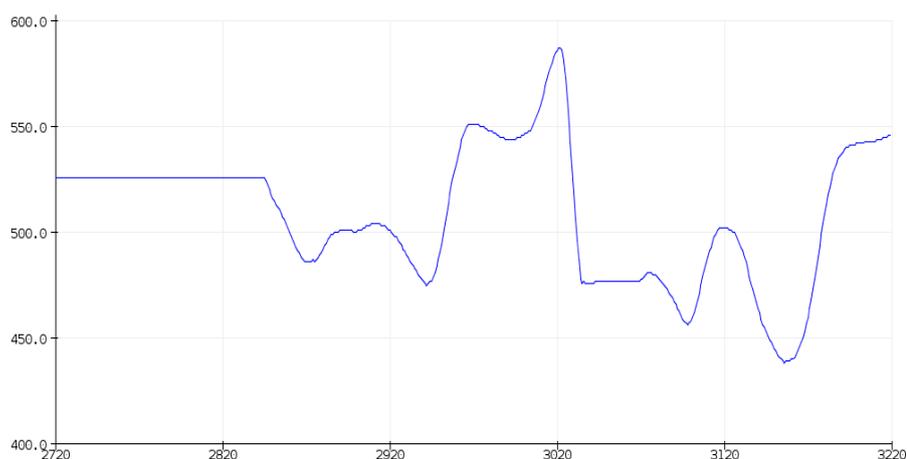
Serial.println(ValorHALL); // Envía el valor por el puerto serie

// Escala el valor de entrada al valor de iluminación del LED verde
ValorMap = map(ValorHALL, 150, 900, 0, 64);
analogWrite(L_Gre, ValorMap); // Enciende el LED verde con el valor escalado

// Un pequeño retardo para estabilizar las medidas.
delay(10);
}
```

Análisis:

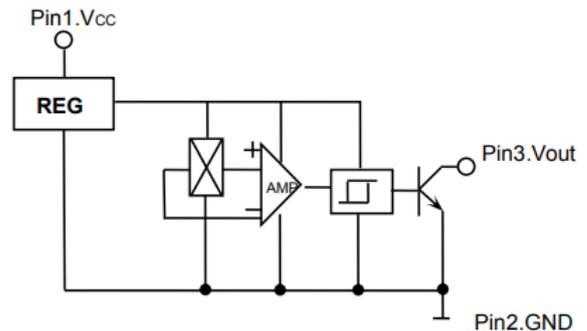
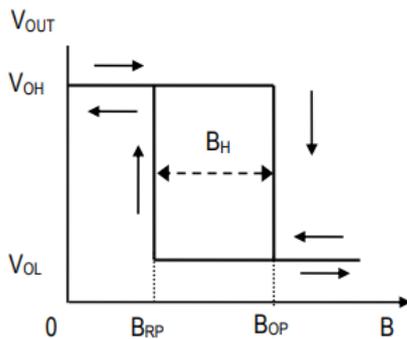
Enciende el LED Rojo cuando detecta un campo magnético norte y enciende el LED naranja cuando detecta un campo magnético sur. Los valores de activación los podemos ajustar en umbralN y umbralS. El LED verde se encenderá con una intensidad proporcional al valor de la tensión de salida del sensor. En reposo, un valor normal es 523.



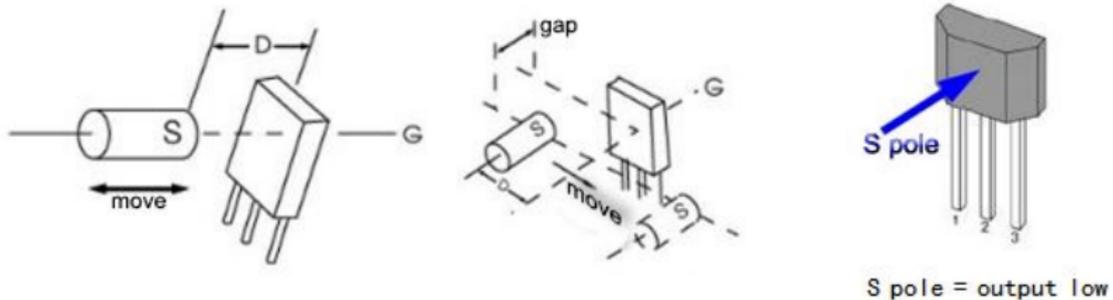


Si lo que necesitamos es detectar el paso de un imán frente al sensor (como en un velocímetro de bicicleta, sensores de nivel, etc.), podemos cambiar el sensor por un [OH44E](#) o [DRV5013-Q1](#), que tienen histéresis, la salida tiene dos estados bien definidos. Esto facilita la detección de un cambio claro entre dos estados (por ejemplo, encendido y apagado) al pasar el imán, sin los valores intermedios que se obtienen en sensores sin histéresis.

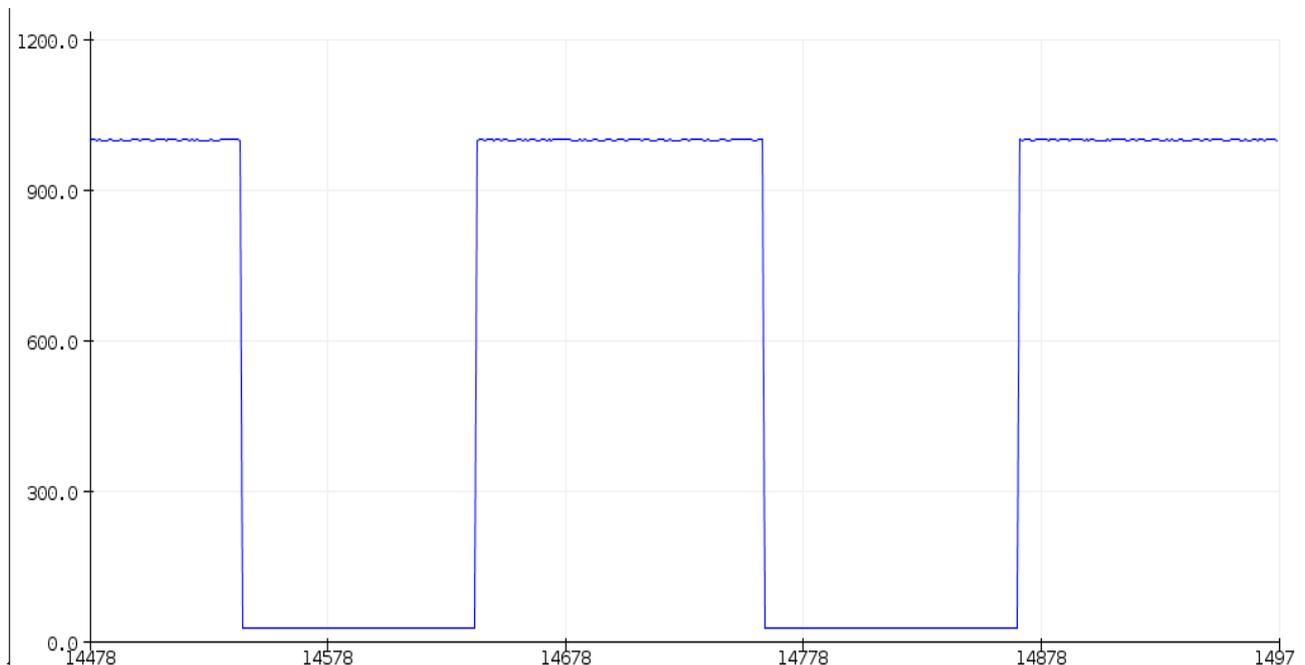
https://win.adrirobot.it/datasheet/speciali/pdf/OH44E-e_sensore-hall.pdf



Typical Working Mode



Nanjing Ouzhuo Technology co., Ltd



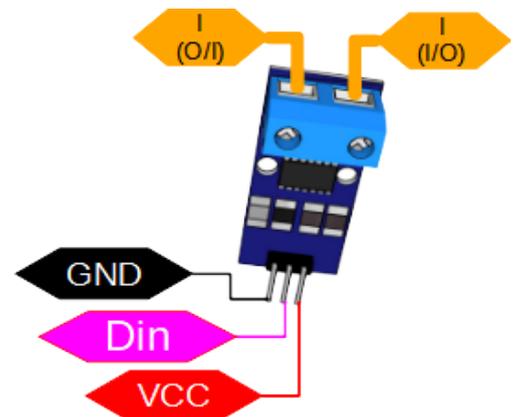


Medida de corriente con ACS712

Otra de las aplicaciones de los sensores de efecto Hall es la posibilidad de medir la corriente que pasa por un cable, ya que cuando por pasa una corriente eléctrica un cable se genera un campo magnético, que puede ser detectado por el sensor ACS712.

Este sensor se presenta en tres versiones:

| Modelo | Rango | Sensibilidad |
|-------------------|-------|--------------|
| ACS712ELCTR-05B-T | ±5A | 185mV/A |
| ACS712ELCTR-20A-T | ±20A | 100mV/A |
| ACS712ELCTR-30A-T | ±30A | 66mV/A |



```
/*Med_Intensidad_ACS712_30A
```

```
Medida de la intensidad usando ACS712-30A
```

```
*/
```

```
float Sensibilidad = 0.066; // Sensibilidad en Volts/Amperios para el sensor de 30A
```

```
float VSensor; // Variable para la tensión del sensor
```

```
float I; // Variable para el cálculo de la intensidad
```

```
float MediaI; // Cálculo de la media de las medidas
```

```
int n = 500; // Nº de muestras para la media
```

```
void setup() {
```

```
  Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
  media(); // Llama a la medida y cálculo de la media
```

```
  Serial.print("Tensión: ");
```

```
  Serial.print(VSensor);
```

```
  Serial.print("V Corriente: ");
```

```
  Serial.print(MediaI, 2);
```

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

```
}
```

```
void media() {
```

```
  for (int i = 0; i < n; i++) { // Bucle de n medidas
```

```
    VSensor = analogRead(A4) * (4.996 / 1023.0); // Lectura de la tensión del sensor
```

```
    I = (VSensor - 2.5) / Sensibilidad; // Ecuación para obtener la corriente
```

```
    MediaI = MediaI + I; // Acumula las medidas
```

```
    delay(1);
```

```
  }
```

```
  MediaI = (MediaI / n); // Calcula la media de las n medidas
```

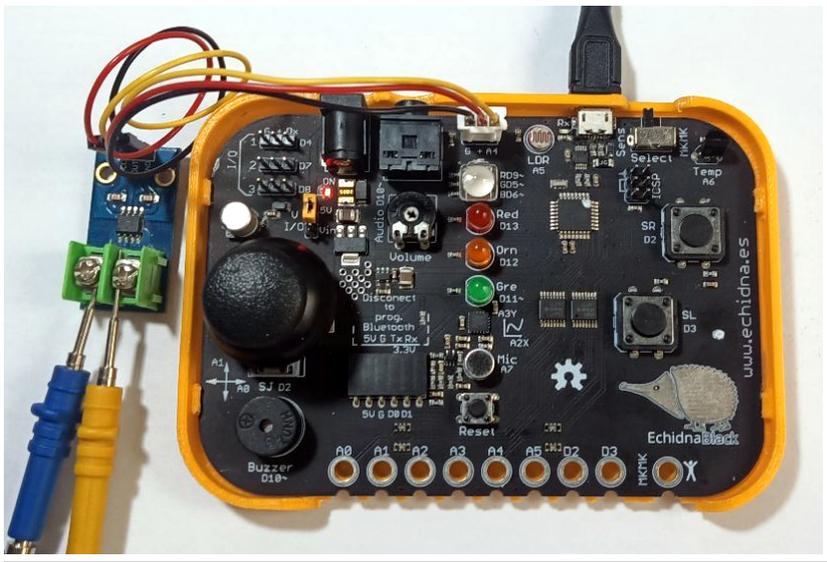
```
}
```



🔑 Análisis:

Tenemos que ajustar la "Sensibilidad" del ACS712 que tengamos. Medimos la tensión de salida teniendo en cuenta que el ACS712 presenta una tensión de "offset" de 2,50V, que nos permite medir corriente en dos sentidos, o corriente alterna. Cuando la salida supere los 2,50V, será corriente en un sentido, y si baja de los 2,50V, será en el sentido contrario.

Es importante tener una referencia de tensión de precisión, ya que la referencia de tensión normal de 5V que tenemos no es muy buena. En el caso de esta placa, se midió la tensión en 4.996V.



Podemos ver tres medidas, la primera de 1A la segunda de 0A y la tercera de -1A

```
Saída Monitor Serial x
Mensaje (Ctrl + Enter para enviar mensaje para 'Arduino Nano' em '/dev/ttyUSB0' Nova linha 9600 baud
Tensión: 2.564V Corrente: 0.99 A
Tensión: 2.569V Corrente: 0.99 A
Tensión: 2.564V Corrente: 1.00 A
Tensión: 2.500V Corrente: 0.57 A
Tensión: 2.500V Corrente: -0.01 A
Tensión: 2.437V Corrente: -0.50 A
Tensión: 2.432V Corrente: -1.01 A
Tensión: 2.437V Corrente: -1.01 A
Tensión: 2.432V Corrente: -1.01 A
Tensión: 2.432V Corrente: -1.00 A
```



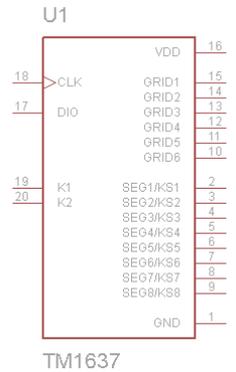
Visualizador de cuatro dígitos. TM1637

El TM1637 es un circuito integrado que permite controlar seis dígitos y un teclado de dieciséis teclas. Este módulo, tiene cuatro dígitos que podemos controlar mediante dos líneas: CLK (reloj) y DIO (entrada y salida de datos). También necesitamos la alimentación de 5V y GND.

Para hacer más fácil el control necesitamos tener instalada la librería.

TM1637Display.h, la instalaremos en el Arduino IDE como siempre:

Poniendo en el filtro "TM1637".



TM1637_Temperatura

/* [TM1637_Temperatura](#)

Usamos el Visualizador de cuatro dígitos controlados por el IC TM1637 de la empresa Titan Micro Electronics. Mediante los pulsadores SR y SL, subimos o bajamos el brillo.

```
#include <TM1637Display.h>
```

```
// Conexiones del módulo de cuatro dígitos
```

```
#define CLK 8 // Señal de reloj
```

```
#define DIO 7 // Señal de datos
```

```
TM1637Display display(CLK, DIO); // Pasamos los valores a la función
```



```
#define SR 2 // Pulsador para subir el brillo
#define SL 3 // Pulsador para bajar el brillo
#define LM35Pin A6 // Pin al que conectamos el sensor de temperatura

uint8_t brillo = 1; // Variable para almacenar el brillo
double temperatura; // Variable para almacenar la temperatura medida (float)
double entera; // Parte entera de la medida de temperatura

const uint8_t celsius[] = { // Matriz para almacenar los caracteres especiales
  SEG_A | SEG_B | SEG_G | SEG_F, // °
  SEG_A | SEG_D | SEG_E | SEG_F // C
};

void setup() {
  Serial.begin(9600);
  analogReference(INTERNAL); // Cambiamos la referencia analógica a 1.1V (AtMega328)
  display.clear(); // Limpiamos los datos del visualizador
  delay(100);
}

void loop() {

  int lectura = analogRead(LM35Pin); // Valor entre 0 y 1023
  temperatura = ((lectura * 1.1 * 100.0) / 1024.0); // Temperatura en ° Celsius

  if (digitalRead(SR) == 1) { // Comprobamos si SR está pulsado
    brillo = brillo + 1; // Subir el brillo
    delay(200);
    if (brillo > 7) { // Evitamos superar el valor 7 del brillo
      brillo = 7;
    }
  }
  if (digitalRead(SL) == 1) { // Comprobamos si SL está pulsado
    brillo = brillo - 1; // Bajar el brillo
    delay(200);
    if (brillo < 1) { // Evitamos bajar de 0 el brillo
      brillo = 0;
    }
  }
  display.setBrightness(brillo); // Enviamos el valor del brillo

  double decimal = modf(temperatura, &entera);

  // Si la parte decimal supera 0,5°, aumentamos en una unidad la parte entera
  if (decimal > 0.5) {
    entera++;
  }
}
```

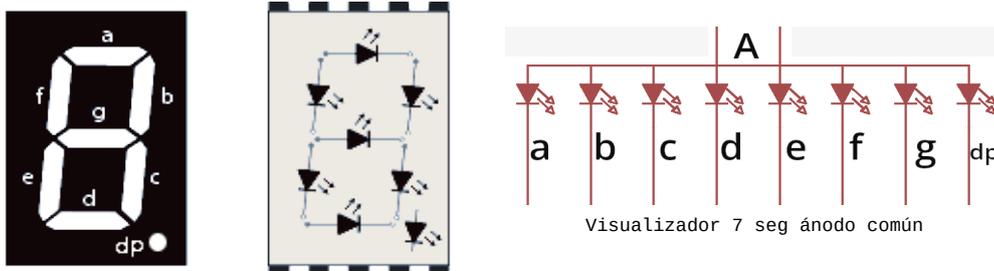


```
display.showNumberDec(entera, false, 2, 0); // Envía la parte entera
display.setSegments(celsius, 2, 2); // Envía los segmentos almacenados

delay(500); // Retardo de 500 milisegundos
}
```

Analise:

Estos visualizadores reciben el nombre de "visualizadores de siete segmentos", están compuestos por siete LEDs y algún punto decimal, generalmente compartiendo en un solo cable el ánodo o cátodo común de todos los LEDs. En el módulo que tenemos entre manos no tenemos que preocuparnos por estas conexiones, ya que el circuito integrado se encarga de multiplexar todas las señales; nosotros solo tenemos que enviar los datos que queremos mostrar.



Para ajustar el brillo usamos dos parámetros:

```
display.setBrightness(brillo, activado);
```

El brillo va de 0 (el más bajo) a 7 (el más alto). Si lo queremos apagar, usamos `display.setBrightness(brillo, false);` o `display.setBrightness(brillo, 0);`.

Si no enviamos el segundo parámetro, se considera "true" o "1".

Para mostrar números usamos `display.showNumberDec(número, ceros a izquierda, longitud, posición);` donde el primer parámetro es el número a mostrar, y el resto de los parámetros son opcionales. El segundo parámetro indica si queremos rellenar con ceros a la izquierda, el tercero es el número de dígitos a mostrar y el último es la posición en el visualizador contando desde la izquierda (0 a 3).

Si queremos mostrar los puntos que tiene el visualizador, podemos usar la versión expandida de la función anterior:

```
display.showNumberDecEx(número, puntos, ceros a izquierda, longitud, posición)
```



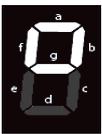
Ejemplo:

```
display.showNumberDecEx(2130, 0b01110000, false, 4, 0)
```

Para mostrar segmentos individuales usamos:

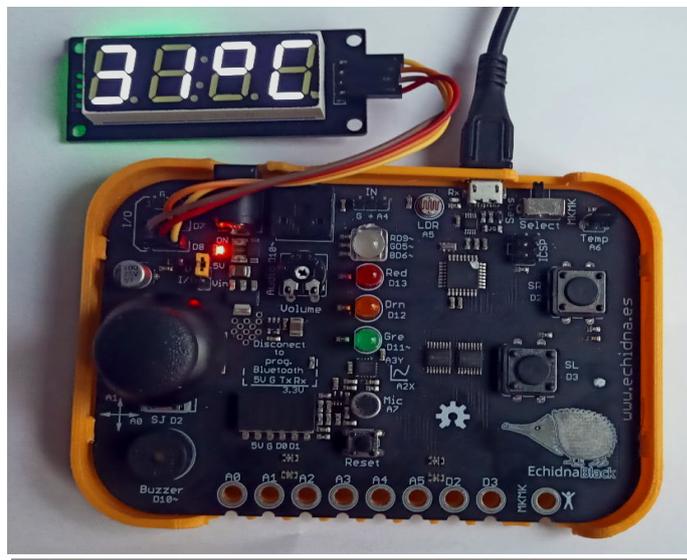
`display.setSegments(segmentos[], longitud, posición)`, donde el primer parámetro es la matriz que contiene la información de los segmentos a iluminar, y el segundo y tercer parámetros son los mismos que en el caso anterior.

La matriz que contiene los dos caracteres “0” y “C” para definir lo que queremos visualizar tiene varias formas de pasar los datos:



“0” en binario es `0b01100011` [`_`, `g`, `f`, `e`, `d`, `c`, `b`, `a`], en decimal es `99`, y en hexadecimal es `0x63`. Sin embargo, es más cómodo pasar solo los segmentos que queremos iluminar, por ejemplo:

```
SEG_A | SEG_B | SEG_G | SEG_F.
```

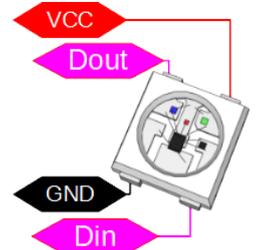




NeoPixel WS2812B

El NeoPixel o LED direccionable consta de tres LEDs RGB y un microprocesador en la misma cápsula “5050”.

Tiene cuatro conexiones: Vcc, GND, Datos de entrada y Datos de salida, lo que nos permite conectarlos entre sí y manejarlos todos mediante un único cable de datos a una velocidad de 800 Kbps y a una distancia máxima de 5 metros, sin necesidad de usar ningún circuito acondicionador de señal.



Podemos ajustar el color de cada LED de forma independiente, alcanzando 16.777.216 colores por píxel. Para controlarlos, podemos usar la librería NeoPixel de Adafruit o la librería FastLED de Daniel García. Tendrás que instalarla.

En el siguiente ejemplo, usamos un solo NeoPixel para mostrar lo sencillo que es controlarlo.

```

/* NeoPixel_I
   Colores al azar en un Neopixel
*/

#include <Adafruit_NeoPixel.h> // Librería
#define PIN A4 // Pin conectado al NeoPixel
#define NUMPIXELS 1 // Número de NeoPíxeles
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

#define Retardo 100 // Tiempo de actualización

void setup() {
  pixels.begin(); // Inicializa el NeoPixel
  pixels.clear(); // Limpia el NeoPixel
}

void loop() {
  // Elegimos colores al azar
  int Rojo = random(64);
  int Verde = random(64);
  int Azul = random(64);
  pixels.setPixelColor(0, pixels.Color(Rojo, Verde, Azul)); // Nº NeoPixel y color
  pixels.show(); // Envía los datos al NeoPixel
  delay(Retardo); // Pausa entre envíos
}

```





Vumetro con tira de NeoPixel WS2812B

Vamos a hacer un [vúmetro](#) usando una pequeña tira de 10 NeoPixels.

```
/* NeoPix\_Vu\_linea
   Vúmetro de una tira de 10 NeoPíxeles
*/

#include <Adafruit_NeoPixel.h> // Librería
#define PIN A4 // Pin donde se conecta la tira de NeoPíxeles
#define NUM_LEDS 10 // Número de píxeles
#define Mic A7 // Entrada conectada al micrófono

int Sonido = 0; // Variable de lectura del sonido
word Amplitud = 35; // Factor de amplificación

word Muestras = 20; // Número de muestras (máx. 255)

double MediaSonido = 0; // Variable de muestras de sonido

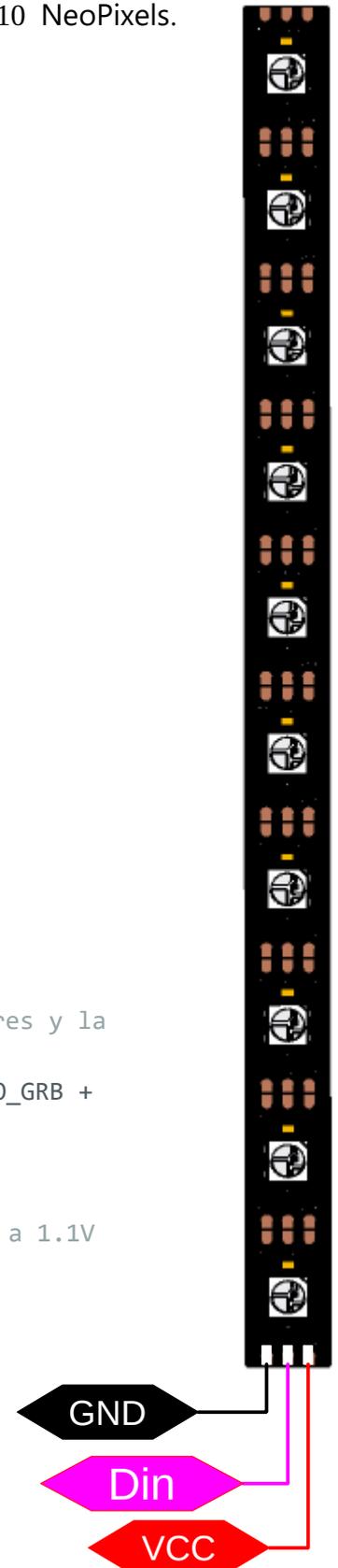
int Nivel = 0; // Variable de amplitud del sonido.
word Retardo = 15; // Número de muestras (máx. 255)

word Rojo; // Variables para los colores
word Verde;
word Azul;
word Y; // Variable eje Y

// Crea el objeto con nº de LEDs, el pin, la secuencia de colores y la
frecuencia de comunicación
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUM_LEDS, PIN, NEO_GRB +
NEO_KHZ800);

void setup() {
  analogReference(INTERNAL); // Ajustamos la referencia interna a 1.1V
  pixels.begin(); // Inicia la comunicación con NeoPíxeles
  pixels.clear(); // Borra todos los píxeles
  pixels.show(); // Envía los datos
}

void loop() {
  // Media aritmética de la entrada de sonido
```





```
for (int i = 0; i < Muestras; i++) {
  Sonido = analogRead(Mic) * Amplitud;
  if (Sonido > 1023) { // Comprueba que no superamos el máximo
    Sonido = 1023;
  }
  MediaSonido = MediaSonido + Sonido;
}
MediaSonido = MediaSonido / Muestras; // Realiza la media aritmética

// Mapea el valor del sonido al nº de LEDs
Nivel = map(MediaSonido, 0, 1023, 0, NUM_LEDS );

// Ajuste del color en función de los NeoPíxeles encendidos entre cero y el valor
máximo
for (Y = 0; Y <= Nivel ; Y++) {
  if (Y >= (NUM_LEDS * 4 / 5 )) {
    // 4/5 del total de NeoPíxeles en rojo
    Rojo = 16;
    Verde = 0;
    Azul = 0;
  }
  if (Y > (NUM_LEDS / 2) && Y < (NUM_LEDS * 4 / 5)) {
    // NeoPíxeles en naranja.
    Rojo = 16;
    Verde = 8;
    Azul = 0;
  }
  if (Y < (NUM_LEDS / 2)) {
    // La mitad de NeoPíxeles en verde
    Rojo = 0;
    Verde = 8;
    Azul = 0;
  }
  pixels.setPixelColor(Y, pixels.Color(Rojo, Verde, Azul));
}

// Borra píxeles superiores entre el valor máximo y el total de NeoPíxeles
for (Y = Nivel + 1; Y < NUM_LEDS; Y++) {
  pixels.setPixelColor(Y, pixels.Color(0, 0, 0));
}

pixels.show(); // Envía los datos a los NeoPíxeles
delay(Retardo); // Pequeño retardo para dejar ver los Neopíxeles
}
```



🔍 Análisis:

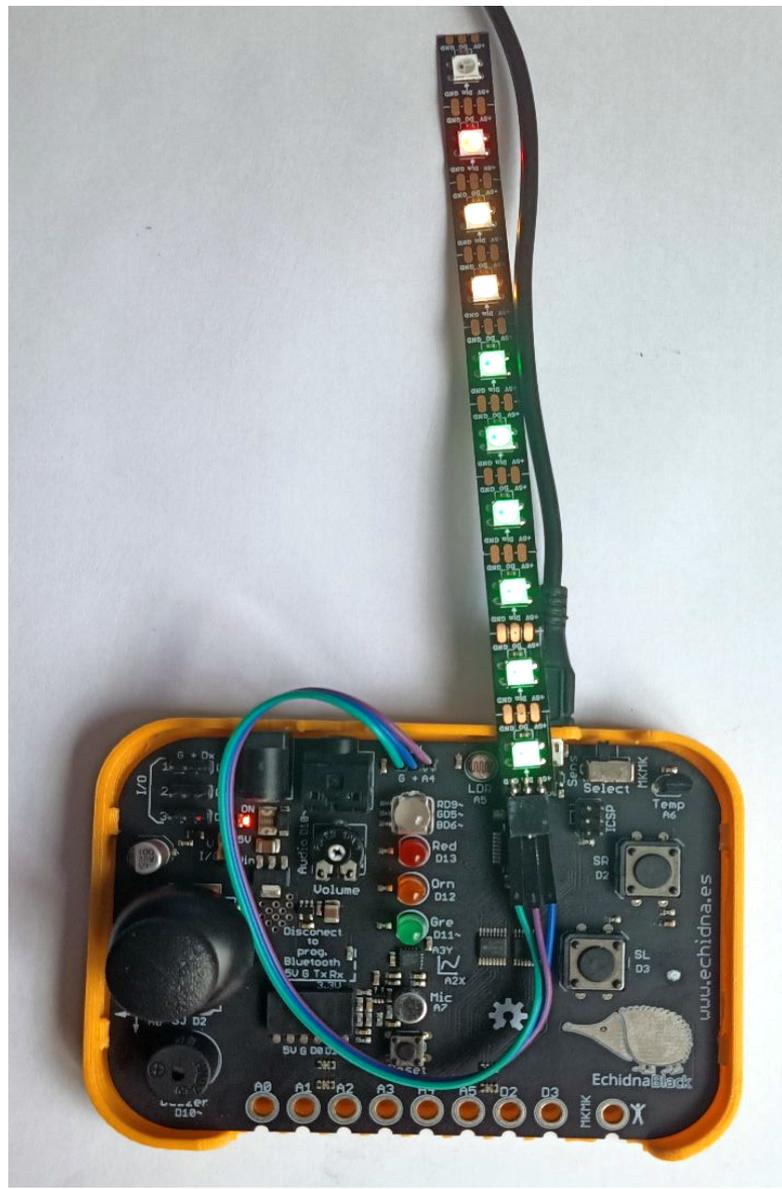
Aparte de las definiciones de pines y variables, pasamos a usar la referencia interna de 1,1V. Realizamos una comprobación para asegurarnos de que la señal de sonido multiplicada por **amp** no supere el máximo (1023).

Tomamos muestras de la señal del micrófono para calcular la media de los valores (cuantas más muestras, más lento será el proceso).

Mapeamos la escala máxima al número total de NeoPixels, comprobamos cuántos NeoPixels encender para elegir el color que queremos en cada sección.

Encendemos los NeoPixels desde cero hasta el valor máximo en ese momento, apagando los NeoPixels entre el valor máximo y el total de la tira.

Finalmente, añadimos una pequeña espera para poder visualizar cada ciclo de encendido de los NeoPixels.



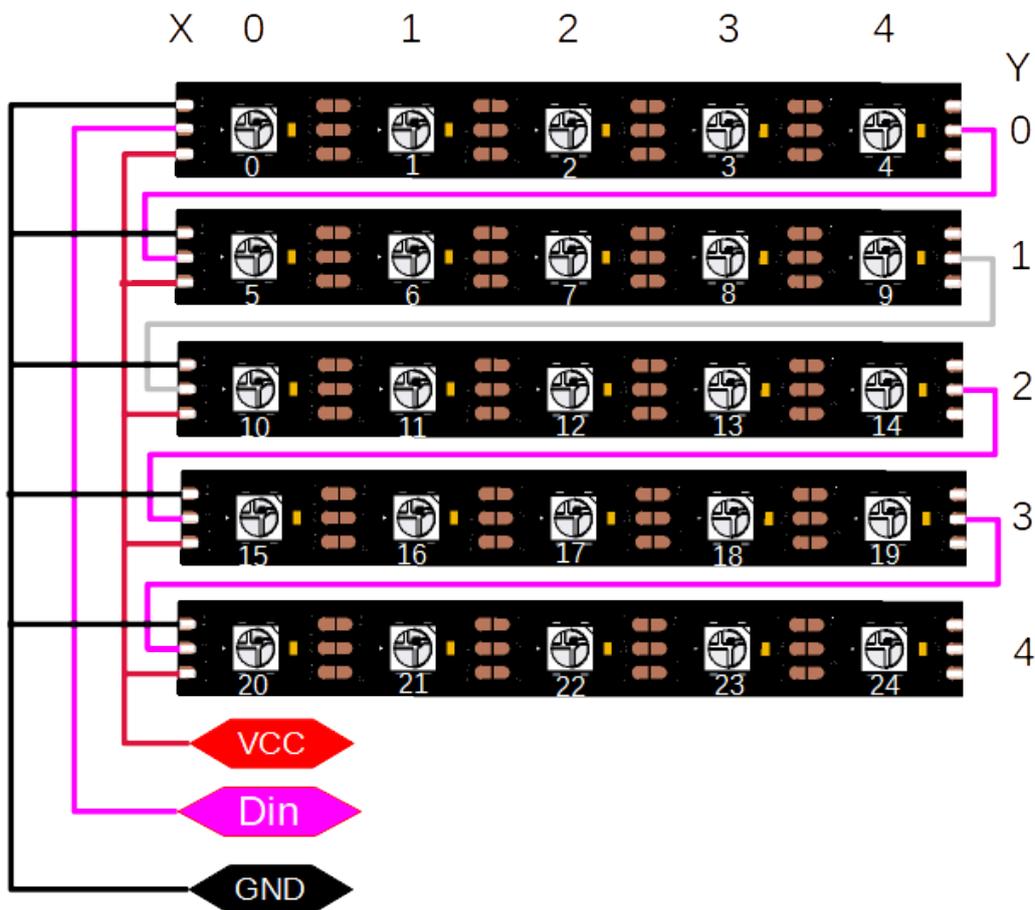
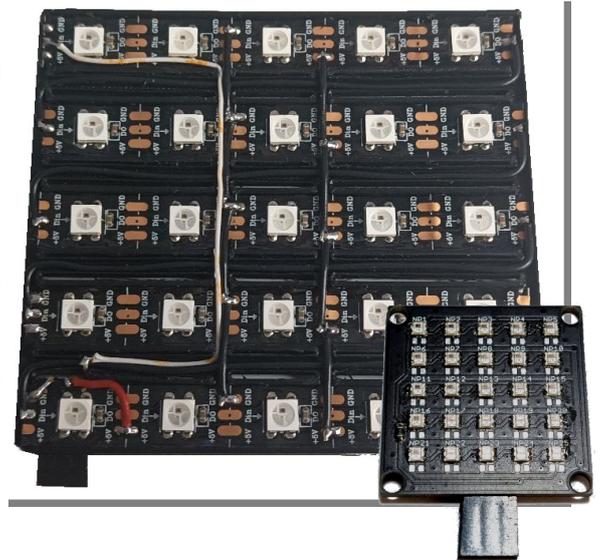


Pantalla 5 x 5 NeoPixel WS2812B

Ya podemos hacer una pequeña pantalla usando cinco tiras de cinco NeoPixels o una PCB con NeoPixels pequeños de 2x2 mm.

Comenzamos conectando la entrada de datos en el NeoPixel superior izquierdo, que será la posición cero ("0"). Desde ahí, se desplazará hacia la izquierda hasta la posición cuatro ("4").

La siguiente tira tendrá las posiciones 5 a 9, la siguiente 10 a 14, y así sucesivamente hasta la posición 24, donde podemos conectar otras tiras o extender la pantalla con más módulos.



Para manejar esta pantalla mediante coordenadas X, Y, usamos la expresión:

$\text{NeoPixel} = X + Y \times 5$ (@caligari), Si queremos iluminar el LED en la posición $X = 3$, $Y = 1$, tendremos que encender el NeoPixel $8 = 3 + (1 \times 5)$



Como ejemplo, vamos a adaptar el programa anterior "NeoPix_Vu_linea" a una pantalla de cinco columnas y cinco filas.

```
/* NeoPix_Vu_linea_Pant
   Visualizador de la amplitud de audio de cinco bandas
   Usando la pantalla de 5 x 5 NeoPixeles
   "https://github.com/xdesig/5x5_NeoPixel_2020"
*/

#include <Adafruit_NeoPixel.h>
#define PIN_NEO A4

word Filas = 5; // Filas
word Columnas = 5; // Columnas

int NUM_LEDS = Columnas * Filas;

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUM_LEDS, PIN_NEO, NEO_GRB +
NEO_KHZ800);

#define MIC A7 // Entrada de audio (Micrófono)
#define SR 2 // Pulsador para subir la amplificación
#define SL 3 // Pulsador para bajar la amplificación

int Amplificacion = 35; // Valor de amplificación
int Sonido; // Volumen de audio
int Amplitud; // Amplitud en LEDs

int EjeY; // Variable de resultados (eje Y) mapeados para mostrar en la pantalla
int EjeX; // Variable para recorrer el eje X
int Pixel; // Variable de pixel a mostrar (pixel = X + Y*5)
int Retardo = 10; // Retardo en milisegundos entre ciclos

int Rojo; // Variable del color rojo
int Verde; // Variable del color verde
int Azul; // Variable del color azul

void setup() {
  analogReference(INTERNAL); // Referencia del convertidor analógico/digital a
1.1Volts
  pinMode(SR, INPUT); // Definimos el rol de cada pin
  pinMode(SL, INPUT);
  pixels.begin(); // Inicializamos los NeoPixeles
}
```





```

void loop() {
  Rojo = random (32); // Elegimos los colores de cada ciclo "colores al azar"
  Verde = random (32);
  Azul = random (16);
  for (EjeX = 0; EjeX < Columnas; EjeX++) { // Recorremos la pantalla en el eje X

    // Leemos la entrada de sonido y la multiplicamos por amplificación
    Sonido = (analogRead(MIC) * Amplificacion);
    if (Sonido >= 1023) { // Comprobamos que no superamos el valor máximo
      Sonido = 1023;
    }
    // Mapeamos los valores de amplitud (filas) e invertimos
    // para iluminarlos desde abajo hacia arriba

    Amplitud = map(Sonido, 0, 1023, Filas, 0);

    for (EjeY = Filas; EjeY >= Amplitud; EjeY--) { // Encendemos los NeoPíxeles
verticales
      Pixel = EjeX + EjeY * Columnas; // Pasamos las coordenadas X y Y a la línea de
píxeles
      // Encendemos cada pixel con su color
      pixels.setPixelColor(Pixel, pixels.Color(Rojo, Verde, Azul));
    }

    // Apagando los píxeles superiores al valor máximo medido de cada columna
    // Usamos "Amplitud-1" para no apagar los NeoPíxeles inferiores

    for (int EjeY = 0; EjeY < Amplitud - 1; EjeY++) {
      Pixel = EjeX + EjeY * Columnas;
      pixels.setPixelColor(Pixel, pixels.Color(0, 0, 0));
    }

    pixels.show(); // Enviamos los datos a la pantalla de NeoPíxeles
  }

  delay(Retardo); // Un pequeño retardo entre ciclos
}

```

Análisis:

Para mostrar los datos en la pantalla la recorreremos de izquierda a derecha con “for (X = 0; X < Col; X++)”, haciendo que el eje X de la pantalla sea el eje de tiempo. Luego tomamos el valor detectado por el micrófono, una vez multiplicado por “Amp”; tenemos en cuenta que los

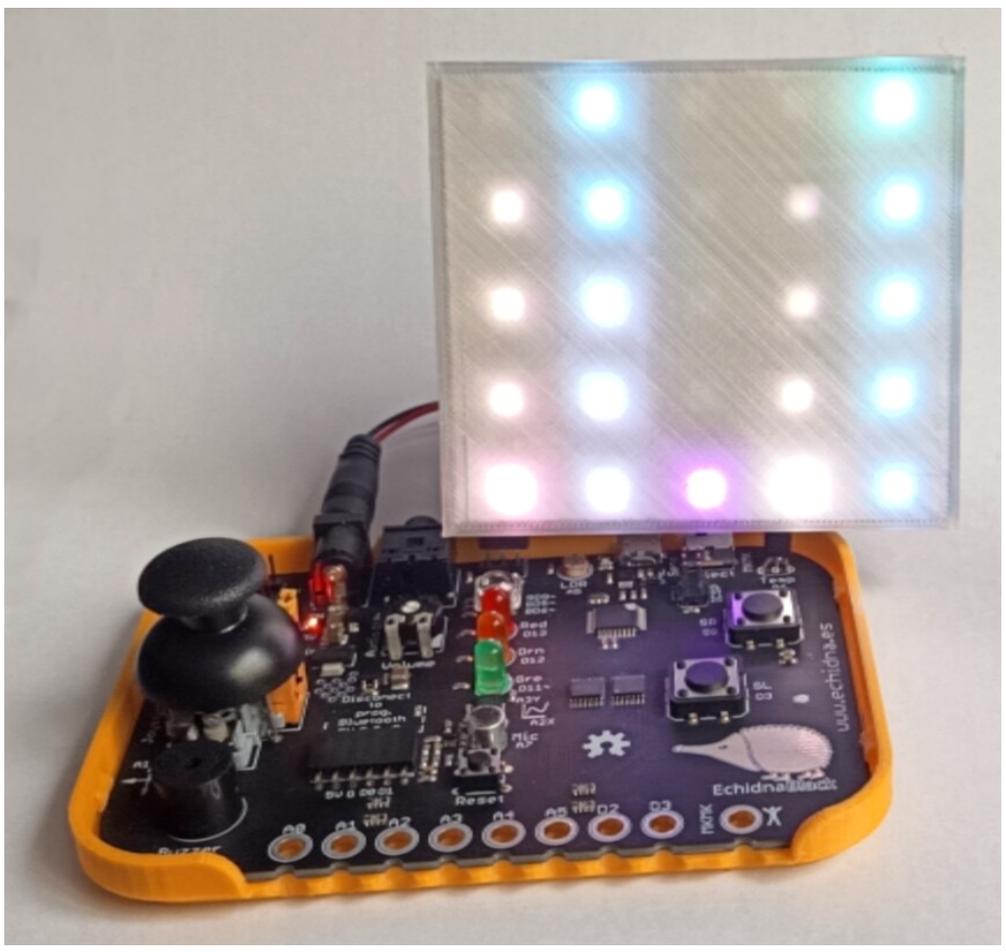


NeoPixels de abajo tienen posiciones más altas que los de arriba, por eso “`A = map(Son, 0, 1023, Fil, 0)`”.

El resultado está invertido, ya que queremos que se iluminen desde abajo hacia arriba.

Con “`for (Y = Fil; Y >= A; Y--)`” recorreremos las filas para encender los NeoPixels, y para apagar los no usados “`for (int Y = 0; Y < A-1; Y++)`”, siempre seguido de la expresión “`Pix = X + Y * Col`”.

Por último, usamos un pequeño retardo para mostrar los NeoPixels encendidos durante más tiempo (podemos bajarlo si necesitamos rapidez).





Analizador de espectro 5 x 5 NeoPixel WS2812B

Siguiendo con la pantalla, podemos hacer un [analizador de espectro](#). Para ello, necesitamos instalar la librería "fix_fft.h".

```

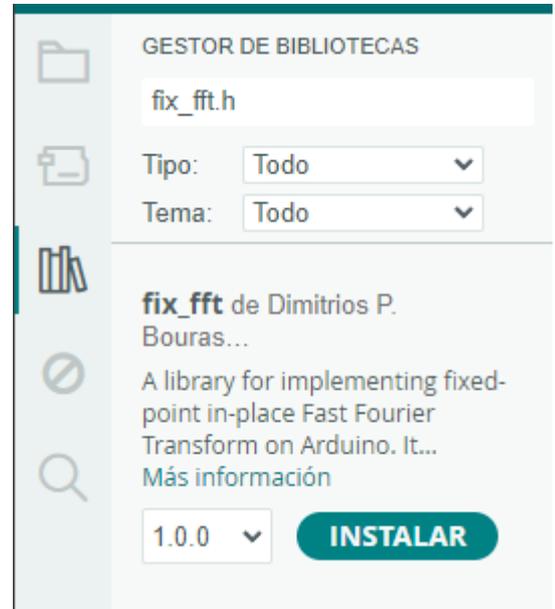
/* Neopix Espectro
Visualizador de espectro de sonido de cinco bandas
Usando la pantalla de 25 NeoPíxeles
Transformada rápida de Fourier (FFT).
Autor: Dimitrios P. Bouras,
Mantenida por Enrique Condes
*/
#include <Adafruit_NeoPixel.h>
#define PIN_NEO A4
#define NUM_LEDS 25
Adafruit_NeoPixel pixels =
Adafruit_NeoPixel(NUM_LEDS, PIN_NEO, NEO_GRB +
NEO_KHZ800);

#include <fix_fft.h> // Biblioteca para calcular la FFT
char re[128], im[128]; // Matrices de muestras / resultados de la FFT
byte i; // Índice
int dat; // Variable de la amplitud de la frecuencia
int F[5] = {2, 3, 6, 9, 15}; // Frecuencias a mostrar en pantalla *73,8Hz
#define MIC A7 // Entrada de audio (Micrófono)
int amp = 25; // Valor de amplificación
int Y; // Variable de resultados (eje Y) mapeados para mostrar en la pantalla
int X; // Variable para recorrer el eje X
int A; // Amplitud en NeoPíxeles
int Pix; // Variable de pixel a mostrar (pix = X + Y*5)
int Rojo; // Variable del color rojo
int Verde; // Variable del color verde
int Azul; // Variable del color azul

void setup() {
  analogReference(INTERNAL); // Ref. Analógica/digital a 1.1Volts
  pixels.begin(); // Inicializamos los NeoPíxeles
}

void loop() {
  // Lectura de sonido y cálculo de la amplitud por frecuencia.
  for (i = 0; i < 128; i++) { // Tomamos 128 muestras de la entrada analógica
(micrófono)
    int sample = amp * (analogRead(MIC));

```





```
re[i] = sample / 4 - 128; // Escalamos para ajustar a un carácter de -128 a 127
im[i] = 0; // Ahora no tenemos valores imaginarios y hacemos todos iguales a cero
}
fix_fft(re, im, 7, 0); // Enviamos las muestras para la conversión FFT,
devolviendo los resultados reales/imaginarios en las mismas matrices

for (X = 0; X < 5; X++) { // Recorremos el eje X de la pantalla
  dat = sqrt(re[F[X]] * re[F[X]] + im[F[X]] * im[F[X]]); // La magnitud de la
  // frecuencia es la raíz cuadrada de la suma de los cuadrados de las partes real
  e imaginaria
  // de los vectores. Solo recuperamos los valores de las frecuencias elegidas F[x]
  // Mapeamos los valores de amplitud de cada frecuencia a los píxeles verticales
  A = map(dat, 0, 25, 4, 0);

  for (Y = 5; Y >= A ; Y--) { // Encendemos los NeoPíxeles verticales
    Pix = X + Y * 5; // Calculamos las coordenadas X y Y en la línea de píxeles.
    if (Y == 0) { // Ajustamos el color en cada línea
      Rojo = 32;
      Verde = 0;
      Azul = 0;
    }
    if (Y == 1) {
      Rojo = 32;
      Verde = 16;
      Azul = 0;
    }
    if (Y == 2) {
      Rojo = 16;
      Verde = 8;
      Azul = 0;
    }
    if (Y == 3) {
      Rojo = 4;
      Verde = 16;
      Azul = 0;
    }
    if (Y == 4) {
      Rojo = 0;
      Verde = 4;
      Azul = 2;
    }
    pixels.setPixelColor(Pix, pixels.Color(Rojo, Verde, Azul));
  }

  for (int Y = 0; Y < A ; Y++) { // Comprobamos que NeoPixel apagar
    Pix = X + Y * 5 ;
    pixels.setPixelColor(Pix, pixels.Color(0, 0, 0));
  }
}
```



```
}  
pixels.show(); // Enviamos los datos a la pantalla de NeoPixeles  
}  
}
```

Analise:

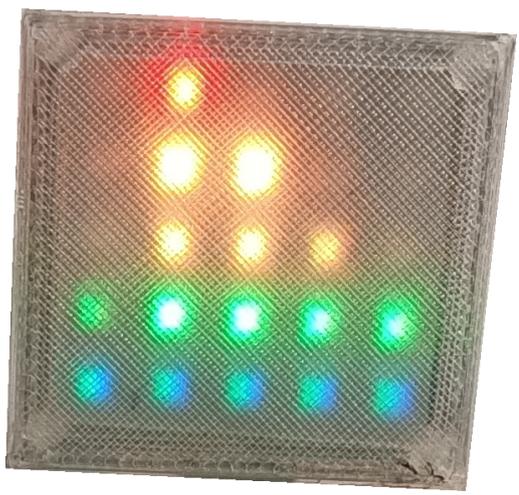
El analizador de espectro de sonido, donde el eje X está en el dominio de la frecuencia y el eje Y representa la amplitud de cada frecuencia analizada, se realiza utilizando la librería "fix_fft.h". Esta librería se emplea para calcular la Transformada Rápida de Fourier (FFT) y descomponer cada grupo de muestras en sus componentes espectrales (frecuencias).

Tomamos 128 muestras del sonido que almacenamos en la matriz "re[128]". Una vez realizado el cálculo, obtendremos los resultados reales en esta matriz y los resultados imaginarios en "im[128]". Posteriormente, calculamos la raíz cuadrada de la suma de los cuadrados de los resultados reales más los cuadrados de las partes imaginarias. Esto se hace para cinco valores, ya que solo tenemos cinco columnas en nuestra pantalla.

Podemos elegir qué frecuencia corresponde a cada columna en la matriz $F[5] = \{2, 3, 6, 9, 15\}$, donde cada número se multiplicará por 73.8Hz.

En el ejemplo tenemos:

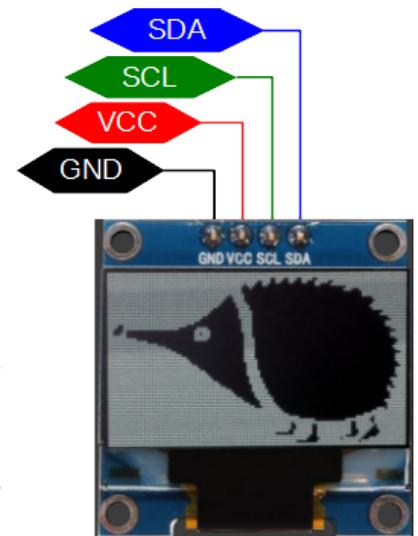
| Columna | 0 | 1 | 2 | 3 | 4 |
|------------|---------|---------|---------|---------|--------|
| Frecuencia | 147,6Hz | 221,4Hz | 442,8Hz | 664,2Hz | 1107Hz |





Pantalla Oled bus I²C

Usaremos la pantalla SSD1306 de 0,96" (2,43 cm) con 128 x 64 píxeles. Es una pantalla de bajo costo que tiene las ventajas de la tecnología OLED y un bus de comunicaciones con solo dos pines: SDA y SCL.



¿Qué es OLED?

OLED es el acrónimo de Organic Light-Emitting Diode (Díodo Orgánico Emisor de Luz), un diodo emisor de luz compuesto por materiales orgánicos electroluminiscentes. Fueron descubiertos por Heeger, MacDarmid y Shirakawa, quienes recibieron el Premio Nobel de Química en el año 2000.

Los OLEDs producen una gran luminosidad debido a su delgadez, pueden fabricarse en formatos grandes, no requieren iluminación externa, consumen menos que otras pantallas, ofrecen un amplio ángulo de visión (alrededor de 170 grados), tienen una larga vida útil de funcionamiento (entre 10,000 y 40,000 horas, aunque los azules duran menos), tiempos de respuesta más rápidos y un gran contraste. Sin embargo, son sensibles a la humedad, lo que puede degradarlos rápidamente.

Más información puede encontrarse en:

"Fundamentos de la Tecnología OLED", editado por P. Chamorro Posada, J. Martín Gil, P. Martín Ramos y L. M. Navas Gracia en la Universidad de Valladolid.

Conexiones:

(Atención, algunas versiones pueden cambiar los pines).

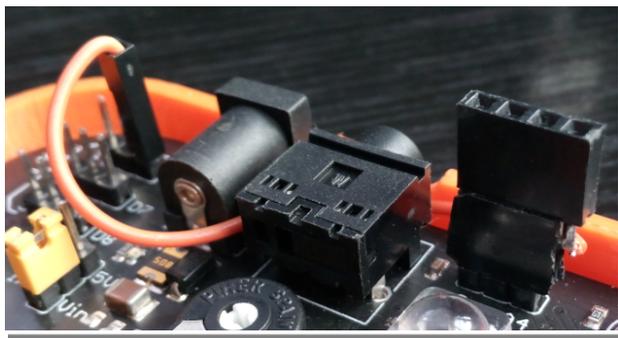
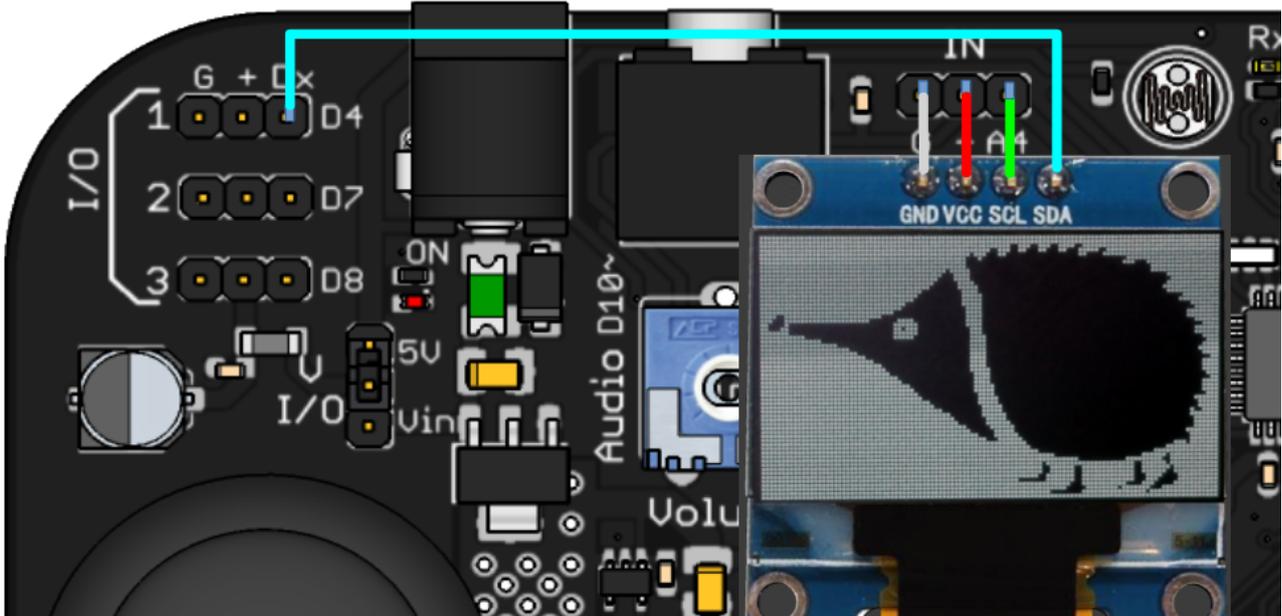
| Pantalla OLED | Echidna |
|--------------------|---------|
| GND | GND |
| VCC | VCC |
| SCL (System Clock) | A4 |
| SDA (System Data) | D4 |





En el conector IN de la EchidnaBlack solo disponemos de GND, VCC y A4, lo que nos obliga a llevar esa conexión con un pequeño cable a (I/O1) D4.

Tenemos la opción de conectar la pantalla en los pines D4, D7 o D8, pero si la conectamos en el conector IN, la pantalla quedará centrada. :-)

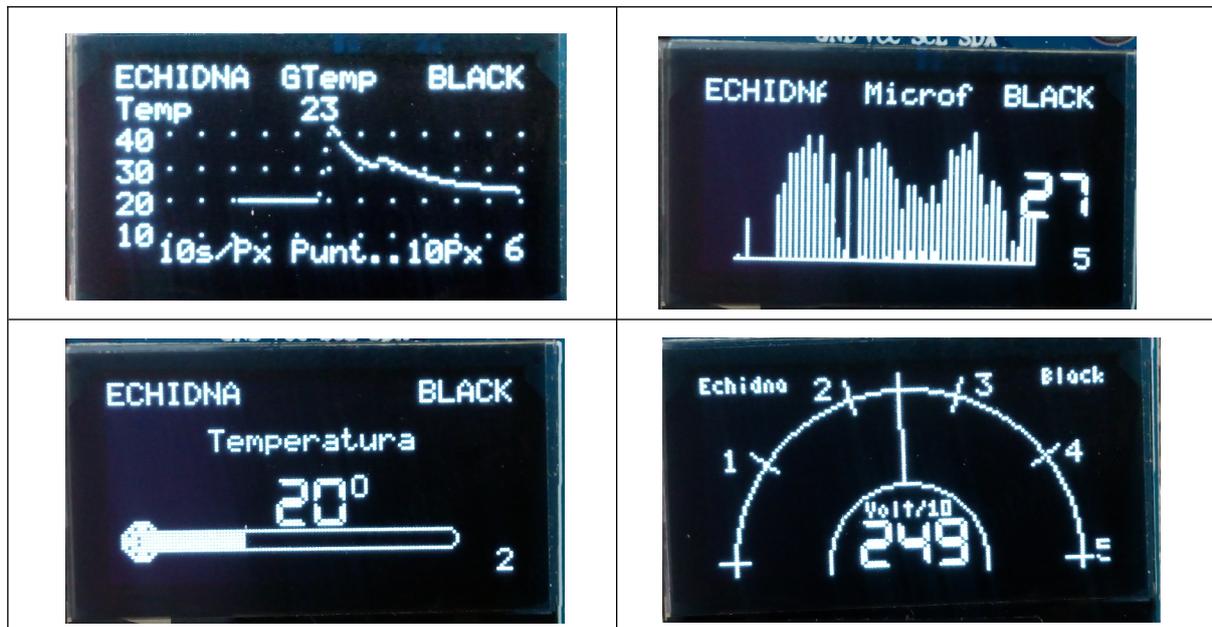




Pasemos a la programación:

Necesitamos una librería que nos permita controlar la pantalla, cambiando los pines de comunicación.

Podemos usar la librería "OLED_I2C.h" (2019) de Henning Karlsen, con licencia CC BY-NC-SA 3.0. Es una librería muy sencilla, no cuenta con las soluciones gráficas de la librería de Adafruit "Adafruit_GFX.h", pero con un poco de imaginación podemos realizar algunas gráficas interesantes.



Ejemplos

Librería:

https://github.com/jlegas/OLED_I2C/archive/refs/heads/master.zip

Una vez descargada, la instalaremos con [Sketch] → [Incluir librería] → [Añadir librería .ZIP...].

OLED_I2C

Multi-platform library for I²C 128x32 and 128x64 pixel SSD1306 OLEDs

Manual



vamos a empezar con un simple ejemplo que pinta un cuadro en la pantalla y un círculo siguiendo el movimiento del Joystick,

Oled_Cuadro_JOY

```

/* Oled_Cuadro_JOY
Dibuja un cuadro límite del joystick
*/

#include <OLED_I2C.h> //(C)Rinky-Dink Electronics
OLED myOLED(4, A4); //establece los pines de comunicación I2C OLED A4 y D4

//***** Establece las entradas de la señal a medir *****
#define EntradaX A0
#define EntradaY A1

//***** Establece los valores mínimos y máximos
const int VXmin = 0;
const int VXmax = 1023;
const int VYmin = 0;
const int VYmax = 1023;
const int Radio = 15; //Radio de la circunferencia
int PosX = 64; //coordenada centro X
int PosY = 32; //coordenada centro Y
int MedidaX; //variable para la lectura del valor X
int MedidaY; //variable para la lectura del valor Y
int i; // Variable común

void setup()
{
  myOLED.begin(); //inicializa el visualizador OLED 128x64
  Serial.begin(115200); // Inicializa la comunicación serie
}

void loop () {
  //***** Lee la entrada analógica *****
  MedidaX = analogRead(EntradaX);
  MedidaY = analogRead(EntradaY);

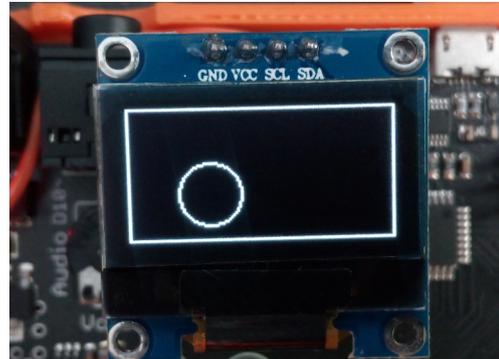
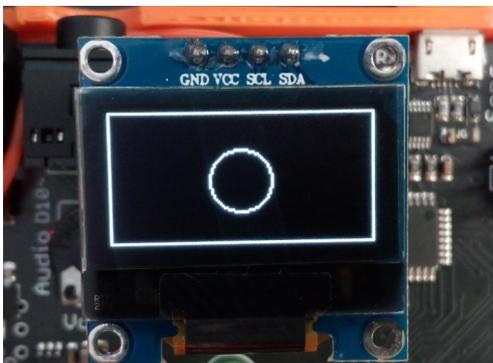
  //***** Envía el valor vía serie *****
  Serial.print (MedidaX);
  Serial.print ("\t");
  Serial.println (MedidaY);

  //***** Escalamos los valores X e Y al tamaño de la pantalla
  PosX = map(MedidaX, VXmin, VXmax, 0+Radio, 128-Radio);
  PosY = map(MedidaY, VYmin, VYmax, 64-Radio, 0+Radio);

```



```
//***** Dibuja el rectángulo *****  
myOLED.drawRect(0, 0, 127, 63); // coordenadas inicio x,y fin x,y  
  
//***** Dibuja el círculo en la posición dada por el joystick *****  
myOLED.drawCircle(PosX, PosY, Radio);  
//***** Presenta toda la información que tiene en la memoria OLED **  
myOLED.update();  
delay(1);  
  
//***** Borra el visualizador *****  
myOLED.clrScr();  
}
```



🔍 Análisis:

En el escalado (mapeado) de las posiciones X e Y se tiene en cuenta el radio de la circunferencia para que la circunferencia no salga del cuadro.

Dibujar con esta librería es muy fácil, aquí solo usamos:

`myOLED.drawRect(iniX, iniY, finX, finY)` para el rectángulo.

`myOLED.drawCircle(PosX, PosY, Radio)` para el círculo.

El resto de comandos son:

`OLED myOLED(4, A4)` establece los pines de comunicación I2C para el OLED, A4 y D4.

`myOLED.begin()` inicializa el visualizador OLED.

`myOLED.update()` copia el contenido de la memoria en la pantalla.

`myOLED.clrScr()` borra la pantalla



Oled_Medidor_Analog

```
/* Oled_Medidor_Analog
```

```

    Medidor analógico de aguja en pantalla OLED SSD1306 128 X 64
*/
#include <OLED_I2C.h> //(C)Rinky-Dink Electronics,BY-NC-SA 3.0
OLED myOLED(4, A4); //establece los pines de comunicación I2C OLED
#define Entrada A5 // La entrada de señal a medir
extern uint8_t TinyFont[]; // Fuentes de textos y números
extern uint8_t SmallFont[];
extern uint8_t MediumNumbers[];
extern uint8_t BigNumbers[];

// Modificando los siguientes parámetros se puede cambiar de
// posición y tamaño la representación analógica.
//Radio de la semicircunferencia del cuadrante (valor clave).
const int Radio = 90; // por defecto 90.
const int Radiop = 65; //radio de la semicircunferencia pequeña donde comienza la
    aguja para evitar el N°. (65)
const int Xcentro = 62; //coordenada centro X del cuadrante. (62)
const int Ycentro = 100; //coordenada centro Y del cuadrante. (100)
const int AngMin = 45; //Ángulo mínimo para la línea de la escala. (45)
const int AngMax = 135; //Ángulo máximo para la línea de la escala. (135)
const int Nmarcas = 4; //Número de marcas que queremos. (4)

//*****
int dotX; // variables para los puntos de la escala
int dotY;
int marXini; // variables de inicio y final de las marcas en el cuadrante
int marYini;
int marXfin;
int marYfin;
int Xini; // variables de inicio y final de la aguja
int Yini;
int Xfin;
int Yfin;
int i; // Variable común
const float Pi = 3.1415927; //Pi para calcular ángulos de grados a radianes.
float Alfa; // ángulo en el cuadrante expresado en radianes
float AlfaMin = (AngMin - 180) * ( Pi / 180);
float AlfaMax = (AngMax - 180) * ( Pi / 180);
// calcula el paso entre los ángulos mínimo y máximo.
float Paso = (AlfaMax - AlfaMin) / Nmarcas;
float Marca [Nmarcas]; //establece la matriz para las marcas
//Ojo con los valores, máximo de "Nmarcas"

```



```
// textos a representar en el cuadrante
char const *valores [] = { "", "256", "512", "768", ""};
const int despra[] { 0, 10, 0, 10, 0}; // Desplazamiento X para los valores

void setup(){
  myOLED.begin(); //inicializa el visualizador OLED 128x64
  //myOLED.invert(true); //invierte el display
  Serial.begin(115200); // inicializa la comunicación serie
  Serial.println("Inicializando");
  myOLED.setFont(SmallFont); // fuente de letra pequeña
  myOLED.print("Iniciando", CENTER, 05);
  myOLED.update();
  // calculamos los ángulos donde representar las marcas y los valores de las mismas
  Alfa = AlfaMin;
  for (i = 0; i < (Nmarcas); i++) {
    Marca[i] = Alfa;
    Alfa = Alfa + Paso; // incremento o decremento para adaptarlo a los textos
  }
}

void loop () {
  int Medida = analogRead(Entrada); // lee la entrada analógica
  Serial.println (Medida); // envía el valor por serie
  myOLED.setFont(BigNumbers);
  myOLED.printNumI(Medida, CENTER, 40); //visualiza el valor de la entrada analógica
  Pint_text(); // pinta el texto en el cuadrante
  DibuCirc(); // dibuja el círculo del cuadrante
  PintMarcas(); // dibuja las marcas del cuadrante
  PintValores(); // dibuja los valores

  //***** Convierte a media el ángulo alfa
  int MedidaM = map(Medida, 0, 1023, 45, 135); // medida a un máximo de 180º
  Alfa = (MedidaM - 180) * ( Pi / 180); // calcula el ángulo de la medida

  //***** Dibuja la aguja en el cuadrante
  Xini = Radiop * cos (Alfa) + Xcentro; // calcula los puntos de inicio
  Yini = Radiop * sin (Alfa) + Ycentro;
  Xfin = Radio * cos (Alfa) + Xcentro; // calcula los puntos de fin
  Yfin = Radio * sin (Alfa) + Ycentro;
  myOLED.drawLine(Xini, Yini, Xfin, Yfin); // Dibuja la línea de la aguja
  myOLED.update(); // presenta toda la información que tiene la memoria del OLED
  delay(1);
  myOLED.clrScr(); // borra el contenido del visualizador
}

//***** Pintando los puntos de la escala *****
void DibuCirc() {
```



```

for (float F = AlfaMin; F < AlfaMax; F = F + 0.04) {
dotX = (Radio - 5) * cos (F) + Xcentro; // calcula los puntos X y Y
dotY = (Radio - 5) * sin (F) + Ycentro;
myOLED.setPixel(dotX, dotY);
}
}

//***** Pintando los valores *****
void PintValores() {
for (i = 0; i < Nmarcas; i++) {
myOLED.setFont(SmallFont);
marXfin = (Radio + despra[i] ) * cos (Marca[i]) + Xcentro - 10;
marYfin = (Radio ) * sin (Marca[i]) + Ycentro - 5;
myOLED.print(valores[i], marXfin, marYfin);
}
}

//***** Dibuja las marcas del cuadrante *****
void PintMarcas() {
for (float F = AlfaMin; F < AlfaMax; F = F + Paso) {
// Comienza a 10 puntos menos del radio para no solaparse
marXini = (Radio - 10) * cos (F) + Xcentro;
marYini = (Radio - 10) * sin (F) + Ycentro;
// Finaliza a 5 puntos por encima del círculo dibujado
marXfin = Radio * cos (F) + Xcentro;
marYfin = Radio * sin (F) + Ycentro;
myOLED.drawLine(marXini, marYini, marXfin, marYfin); // Dibuja las marcas
}
}

//***** Pinta los textos Echidna STEAM *****
void Pint_text() {
myOLED.setFont(SmallFont); // Fuente pequeña
myOLED.print("Echidna", LEFT, 55);
myOLED.print("STEAM", RIGHT, 55);
}
}

```



🔍 Análisis:

Modificando los siguientes parámetros se puede cambiar la posición y el tamaño de la representación analógica: Radio, RadioP, Xcentro, Ycentro, Angmin, AngMax, Nmarcas.

Por ejemplo, poniendo Radio = 70 dejando espacio superior para otras indicaciones.



En valores `[] = { "", "256", "512", "768", "" }`; podemos poner nuestros valores.

Para ajustar la posición "X" de esos valores usamos la `despra[] { 0, 10, 0, 10, 0 }`;

Para dibujar el cuadrante y el inicio/fin de la aguja calculamos los ángulos en radianes y calculamos el coseno del ángulo Alfa para la coordenada X y el seno para la coordenada Y.

Podemos descargar los programas del [repositorio de EchidnaShield](#).

La electrónica y la programación abren un mundo de posibilidades, y el verdadero aprendizaje comienza cuando experimentas, pruebas e incluso cometes errores para mejorar. La clave está en comprender los conceptos y utilizarlos para resolver problemas reales.

Recuerda que este manual no es un punto final, sino una puerta de entrada a nuevas ideas y desarrollos. Anímate a seguir investigando, probando nuevas configuraciones y compartiendo tus proyectos con la comunidad.

Ahora es tu turno: **innova, crea y lleva tus diseños al siguiente nivel** 🚀

Más valen proyectos compartidos que mil en un cajón, @obijuan?

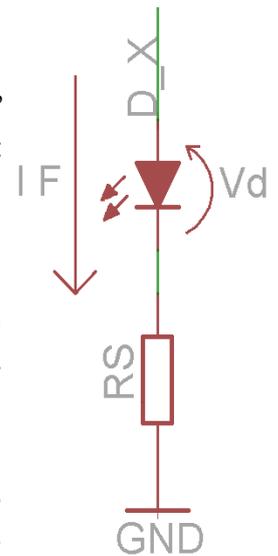


Calculo resistencia serie "Rs"

Cálculo de la resistencia en serie "Rs"
Para calcular la resistencia en serie "Rs", tendremos en cuenta:
Tensión que proporciona en la salida (Dx).

Para este ejemplo supondremos que el circuito está conectado a una de las salidas del ATmega328P que está alimentado a 5Vcc, la tensión en la salida para valor alto es 4,1V mínimo y 5V máximo. Tipo de LED.

El LED será de 5mm de diámetro rojo, que tiene las siguientes características: Corriente directa $I_F = 20\text{mA}$. Tensión directa $V_d = 1,9\text{V}$. Cálculo de Rs.

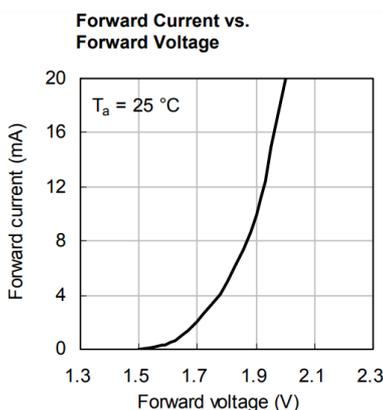


La tensión que cae en la resistencia será la tensión de salida del Atmega (D_X) que llamaremos V_i , menos la tensión necesaria para el funcionamiento del LED V_d . La corriente que circula por la resistencia es la misma que la del LED I_F .

$$R_S = \frac{V_i - V_d}{I_F} = \frac{5\text{V} - 1,9\text{V}}{10\text{mA}} = 310\Omega$$

Para no forzar las salidas del microcontrolador y que los LED duren mucho, podemos reducir la corriente que pasa por él a unos 5mA, lo que implica aumentar la resistencia a 620Ω, donde el valor estándar es 680Ω.

Recuerda que cada fabricante y tipo de LED tiene valores diferentes de tensión VF y corriente IF de funcionamiento, por lo que deberemos consultar las hojas de características.



ELECTRICAL / OPTICAL CHARACTERISTICS at $T_A=25^\circ\text{C}$

| Parameter | Symbol | Emitting Color | Value | | Unit |
|---|------------------------------|---------------------|-------|------|----------------------------|
| | | | Typ. | Max. | |
| Wavelength at Peak Emission $I_F = 10\text{mA}$ | λ_{peak} | High Efficiency Red | 627 | - | nm |
| Dominant Wavelength $I_F = 10\text{mA}$ | λ_{dom} [1] | High Efficiency Red | 617 | - | nm |
| Spectral Bandwidth at 50% Φ REL MAX $I_F = 10\text{mA}$ | $\Delta\lambda$ | High Efficiency Red | 45 | - | nm |
| Capacitance | C | High Efficiency Red | 15 | - | pF |
| Forward Voltage $I_F = 10\text{mA}$ | V_F [2] | High Efficiency Red | 1.9 | 2.3 | V |
| Reverse Current ($V_R = 5\text{V}$) | I_R | High Efficiency Red | - | 10 | μA |
| Temperature Coefficient of λ_{peak} $I_F = 10\text{mA}$, $-10^\circ\text{C} \leq T \leq 85^\circ\text{C}$ | $TC_{\lambda_{\text{peak}}}$ | High Efficiency Red | 0.12 | - | $\text{nm}/^\circ\text{C}$ |
| Temperature Coefficient of λ_{dom} $I_F = 10\text{mA}$, $-10^\circ\text{C} \leq T \leq 85^\circ\text{C}$ | $TC_{\lambda_{\text{dom}}}$ | High Efficiency Red | 0.06 | - | $\text{nm}/^\circ\text{C}$ |
| Temperature Coefficient of V_F $I_F = 10\text{mA}$, $-10^\circ\text{C} \leq T \leq 85^\circ\text{C}$ | TC_V | High Efficiency Red | -1.9 | - | $\text{mV}/^\circ\text{C}$ |

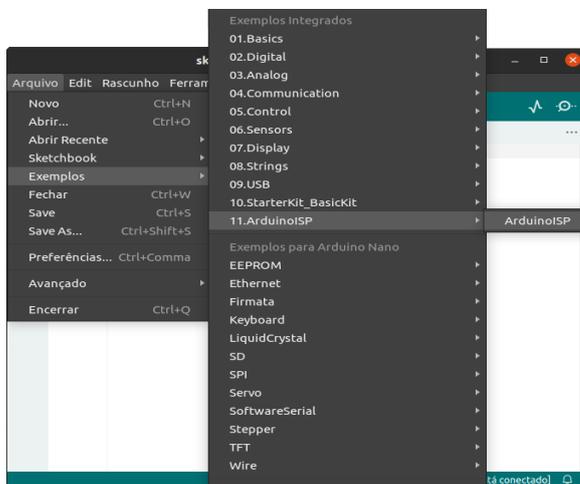
Notes:
1. The dominant wavelength (λ_d) above is the setup value of the sorting machine. (Tolerance: $\lambda_d: \pm 1\text{nm}$.)
2. Forward voltage: $\pm 0.1\text{V}$.
3. Wavelength value is traceable to CIE127-2007 standards.
4. Excess driving current and / or operating temperature higher than recommended conditions may result in severe light degradation or premature failure.

Desenladrillar.

Para volver a la vida la EchidnaBlack después de un "brickeo", tenemos que grabar de nuevo el bootloader mediante un programador ICSP (In-Circuit Serial Programming). Si no tenemos un programador, podemos usar un Arduino o otra EchidnaBlack.

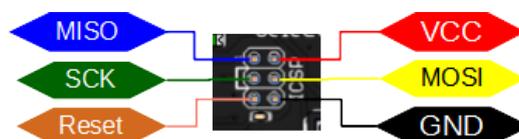
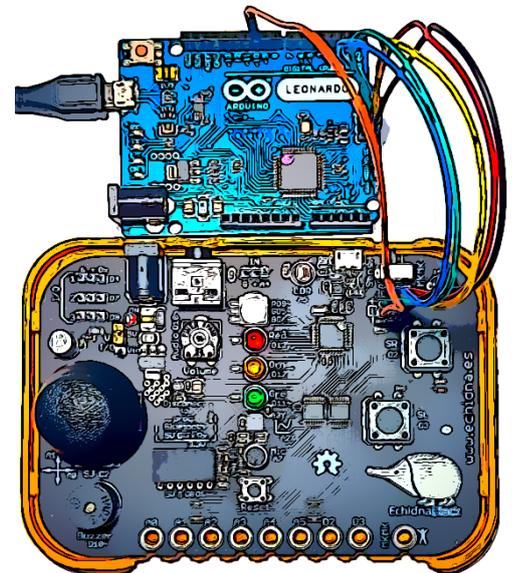
Vamos a comenzar con un Arduino Nano, Uno, Leonardo...

1. Conectamos el Arduino al ordenador y seleccionamos el tipo de Arduino.
 1. Subimos el Sketch (programa) que tenemos en: [Archivo] [Ejemplos] [ArduinoISP]



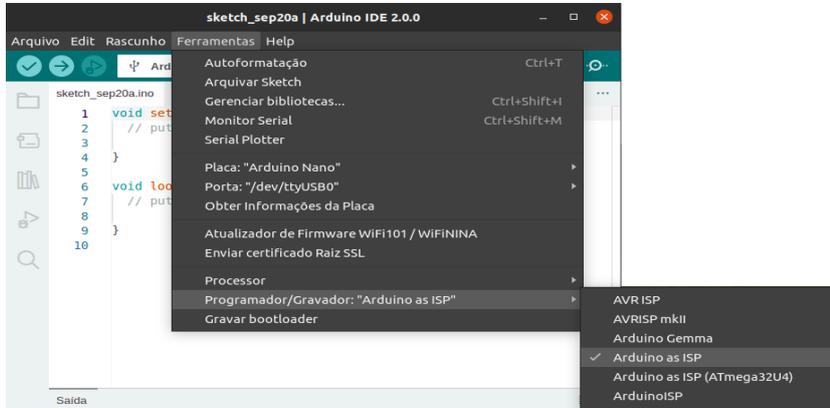
2. Realizamos las conexiones entre el Arduino y la EchidnaBlack.

| Arduino | EchidnaBlack |
|---------|--------------|
| MISO | MISO |
| MOSI | MOSI |
| SCK | SCK |
| D10 | RESET |
| Vcc | Vcc |
| GND | GND |





3. Seleccionamos como placa Arduino Nano Procesador ATmega328.
4. Elegimos el programador "Arduino as ISP".



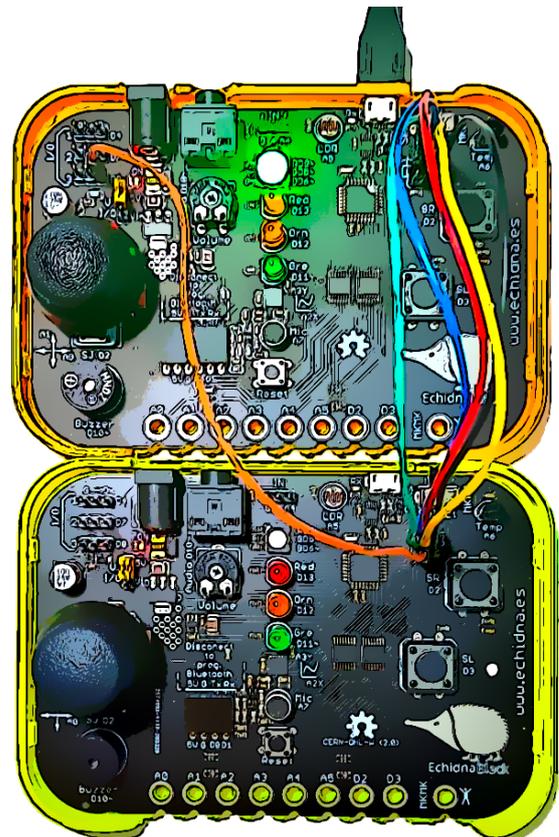
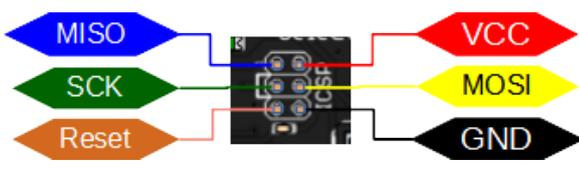
5. Ya podemos "Pinchar" en [Grabar bootloader].

Tras un minuto o algo más, depende del sistema operativo, ya tendremos la EchidnaBlack con el bootloader correcto.

Si no tenemos un Arduino, pero sí otra EchidnaBlack, podemos convertirla en un programador ISP, seguiremos los pasos anteriores con unas pequeñas modificaciones en las conexiones y el programa.

Conexiones:

| Echidna Black Como programador | EchidnaBlack DESTINO |
|--------------------------------|----------------------|
| MISO | MISO |
| MOSI | MOSI |
| SCK | SCK |
| D8 | RESET |
| Vcc | Vcc |
| GND | GND |



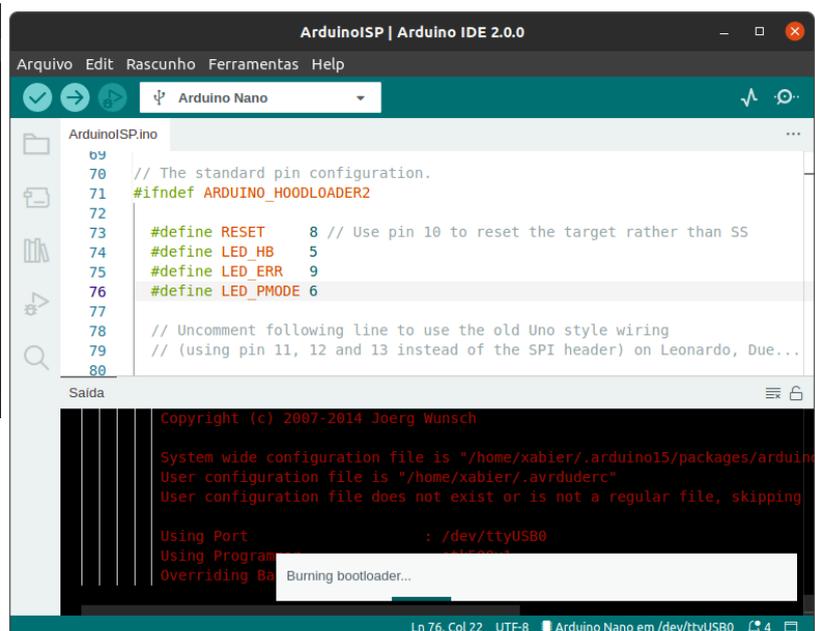
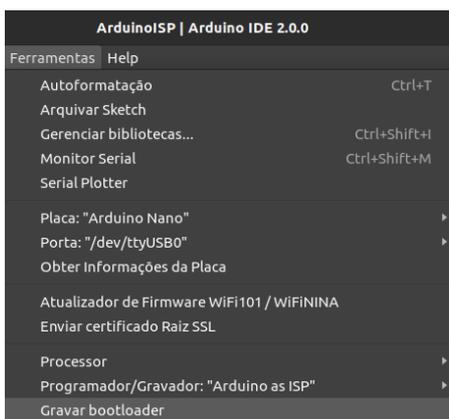


Cambios en el "Sketch" ArduinoISP, en las líneas 73 a 76:

```
#define RESET      8
#define LED_HB     5
#define LED_ERR    9
#define LED_PMODE 6
```

```
ArduinoISP.ino
70 // The standard pin configuration.
71 #ifndef ARDUINO_HOODLOADER2
72
73 #define RESET      8 // Use pin 10 to reset the target rather than SS
74 #define LED_HB     5
75 #define LED_ERR    9
76 #define LED_PMODE 6
77
78 // Uncomment following line to use the old Uno style wiring
79 // (using pin 11, 12 and 13 instead of the SPI header) on Leonardo, Due...
80
81 // #define USE_OLD_STYLE_WIRING
82
83 #ifdef USE_OLD_STYLE_WIRING
84
85 #define PIN_MOSI  11
86 #define PIN_MISO  12
87 #define PIN_SCK   13
88
89 #endif
90
91 // HOODLOADER2 means running sketches on the ATmega16U2 serial converter chip
92 // on Uno or Mega boards. We must use pins that are broken out:
```

Seleccionamos la placa Arduino Nano, procesador Atmega328P, el puerto serie correspondiente, subimos ArduinoISP a EchidnaBlack (programador), una vez subido ya podemos grabar el bootloader en EchidnaBlack (Destino).





Atribución 4.0 Internacional (CC BY 4.0)

Usted es libre de:



- Compartir: copiar y redistribuir el material en cualquier medio o formato.
- Adaptar: rehacer, transformar y recrear el material para cualquier propósito, incluso comercialmente.

El licenciante no puede revocar estas libertades mientras usted cumpla con los términos de la licencia.

Bajo los siguientes términos:

- Atribución: Debe dar el reconocimiento apropiado, proporcionar un enlace a la licencia e indicar si se hicieron cambios. Puede hacerlo de cualquier manera razonable pero no de manera que sugiera que el licenciante lo apoya a usted o su uso.
- Sin restricciones adicionales: No puede aplicar términos legales ni medidas tecnológicas que impidan legalmente a otros hacer algo que la licencia permite.

Notas:

- No es necesario cumplir con la licencia para los elementos del material que estén en dominio público o cuando su uso esté permitido mediante una excepción o limitación aplicable.

No se ofrecen garantías. La licencia puede no proporcionar todos los permisos necesarios para el uso previsto. Por ejemplo, otros derechos como [publicidad, privacidad o derechos morales \(intelectuales\)](#).

Algunas cosas:
[Añadir puntos, 1.1, ...](#)
[Imágenes en tablas](#)
[Plantilla para formato](#)

Redactado y traducido en Estrimia, febrero 2025 (002)