

Manual Echidna Black

Posta en marcha e proxectos
con Echidna Black



Xabier Rosas

Índice

| | |
|--|----|
| Coñecendo a placa..... | 4 |
| Selector alimentación 5V..... | 8 |
| Selector alimentación Vin (Alimentación externa)..... | 8 |
| Modo sensores..... | 9 |
| Modo MkMk..... | 9 |
| Descripción dos componentes:..... | 10 |
| LED RGB..... | 11 |
| Son..... | 12 |
| Pulsadores..... | 13 |
| Panca de xogos “Joystick”..... | 15 |
| Sensor de Luz: LDR..... | 16 |
| Sensor de temperatura LM35..... | 17 |
| Micrófono..... | 18 |
| Acelerómetro..... | 19 |
| Entradas (Modo MkMk)..... | 20 |
| Pines de libre disposición..... | 21 |
| Conector ISCP “In-Circuit Serial Programming”..... | 22 |
| Programación (falemos con EchidnaBlack)..... | 23 |
| IDE de Arduino..... | 24 |
| Estrutura do programa “Sketch” de Arduino:..... | 27 |
| Sketch “Títila_I”..... | 30 |
| Subir o programa..... | 31 |
| LED_PWM..... | 32 |
| Operadores aritméticos (A =2, B = 4)..... | 33 |
| Operadores de comparación (A =2, B = 4)..... | 33 |
| Operadores lóxicos (A =2, B = 4)..... | 34 |
| Operadores booleanos (A =2, B = 3)..... | 34 |
| Operadores compostos (A =2, B = 3)..... | 35 |
| For..... | 36 |
| LED_PWM_2..... | 36 |
| Cores_RGB..... | 37 |
| Tons..... | 39 |
| Pulsador_I..... | 42 |
| if...else..... | 42 |
| While..... | 44 |
| Lectura analólica..... | 45 |
| JoyStick_analox..... | 46 |
| Tipos de datos:..... | 48 |
| Proposta A, B:..... | 49 |
| LDR..... | 50 |
| Theremín LDR..... | 51 |
| Temperatura..... | 52 |
| Vúmetro..... | 55 |
| Nivel eixo Y..... | 58 |
| Xogo “Colle a EchidnaBlack sen activar a Alarma.”..... | 60 |
| Piano MkMk..... | 64 |
| Piano_2..... | 67 |
| Acender LEDes dende o ordenata..... | 69 |
| Axusta a iluminación LED dende o ordenata..... | 70 |

| | |
|---|-----|
| Acende LEDes desde o SmartPhone..... | 71 |
| Comandos AT para HC-06..... | 72 |
| Usando elementos externos..... | 76 |
| Servomotor..... | 76 |
| Sensor de proximidade de infravermellos Sharp 0A41SK..... | 79 |
| Sensor distancia con ultrasons HC-SR04..... | 82 |
| Instalar librerías..... | 84 |
| Medida do % de humidade e temperatura co DHT-11..... | 86 |
| Sensor capacitivo de humidade do chan..... | 88 |
| EEPROM..... | 93 |
| Sensor capacitivo de humidade con Límites en EEPROM..... | 94 |
| Receptor de Infravermellos..... | 99 |
| IR_Receptor..... | 101 |
| Emisor de Infravermellos..... | 103 |
| Detectando campos magnéticos..... | 105 |
| Medida de corrente con ACS712..... | 108 |
| Visualizador de catro díxitos. TM1637..... | 110 |
| TM1637_Temperatura..... | 110 |
| NeoPixel WS2812B..... | 114 |
| /* NeoPixel_I..... | 114 |
| Vumetro con tira de NeoPixel WS2812B..... | 115 |
| Pantalla 5 x 5 NeoPixel WS2812B..... | 118 |
| Analizador de espectro 5 x 5 NeoPixel WS2812B..... | 122 |
| Pantalla Oled bus I ² C..... | 125 |
| Pasemos a programación:..... | 127 |
| Oled_Cadro_JOY..... | 128 |
| /* Oled_Cadro_JOY..... | 128 |
| Oled_Medidor_Analox..... | 130 |
| /* Oled_Medidor_Analox..... | 130 |
| Calculo resistencia serie “Rs”..... | 135 |
| Desenladrillar..... | 136 |
| Eres é libre de:..... | 140 |

ISBN: 9798338113295





Tres docentes decidimos crear nosa propia ferramenta para traballar con robótica na aula, usando hardware e software libre. Creamos materiais e contidos didácticos baixo licencia [CERN-OHL-W](#) e [CC BY 4.0](#).

Creamos Echidna Educación como unha asociación sen ánimo de lucro que ten como obxectivos: Promover o ensino da programación, e robótica mediante o uso de ferramentas de código aberto. Facilitar o acceso á programación de dispositivos físicos a través da creación de electrónica de acceso libre “hardware open source”. Deseñado a partir das necesidades e experiencias da aula. Un tempo despois chegaron dous docentes máis configurando o actual equipo @EchidnaSteam



EchidnaBlack é unha placa autónoma compatible con Arduino Nano/ Arduino UNO...

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

Conta cos certificados.



Certificado CE: ISETC.000520201231
Electromagnetic Compatibility
2014/30/EU

EN 55032:2015+A11:2020, EN 55035:2017
The Open Source Hardware Association
ES000010
Directiva CE 2002/95/EC





Coñecendo a placa.

Imos coñecer as partes que componen esta placa de adestramento, e aprenderemos a programar os actuadores integrados e poder ler os sensores, mediante sinxelos exemplos, como controlar unha luz, ler a temperatura para activar unha saída en función da mesma, e outros exemplos que iremos vendo o longo do manual.

Nas explicacións dos compoñentes atoparemos:

Texto en cursiva indicando a conexión do compoñente.

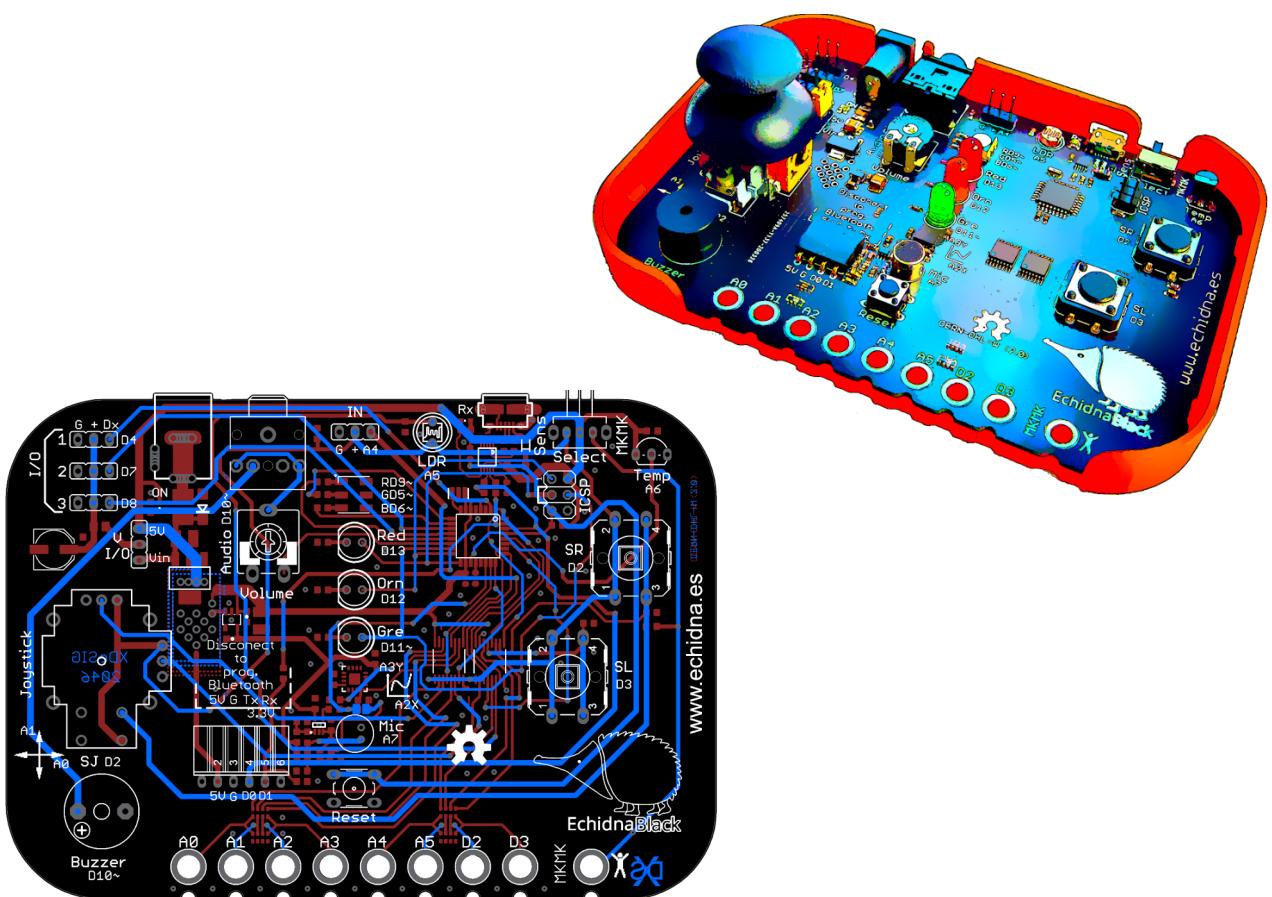
D2 "SL": Entrada dixital

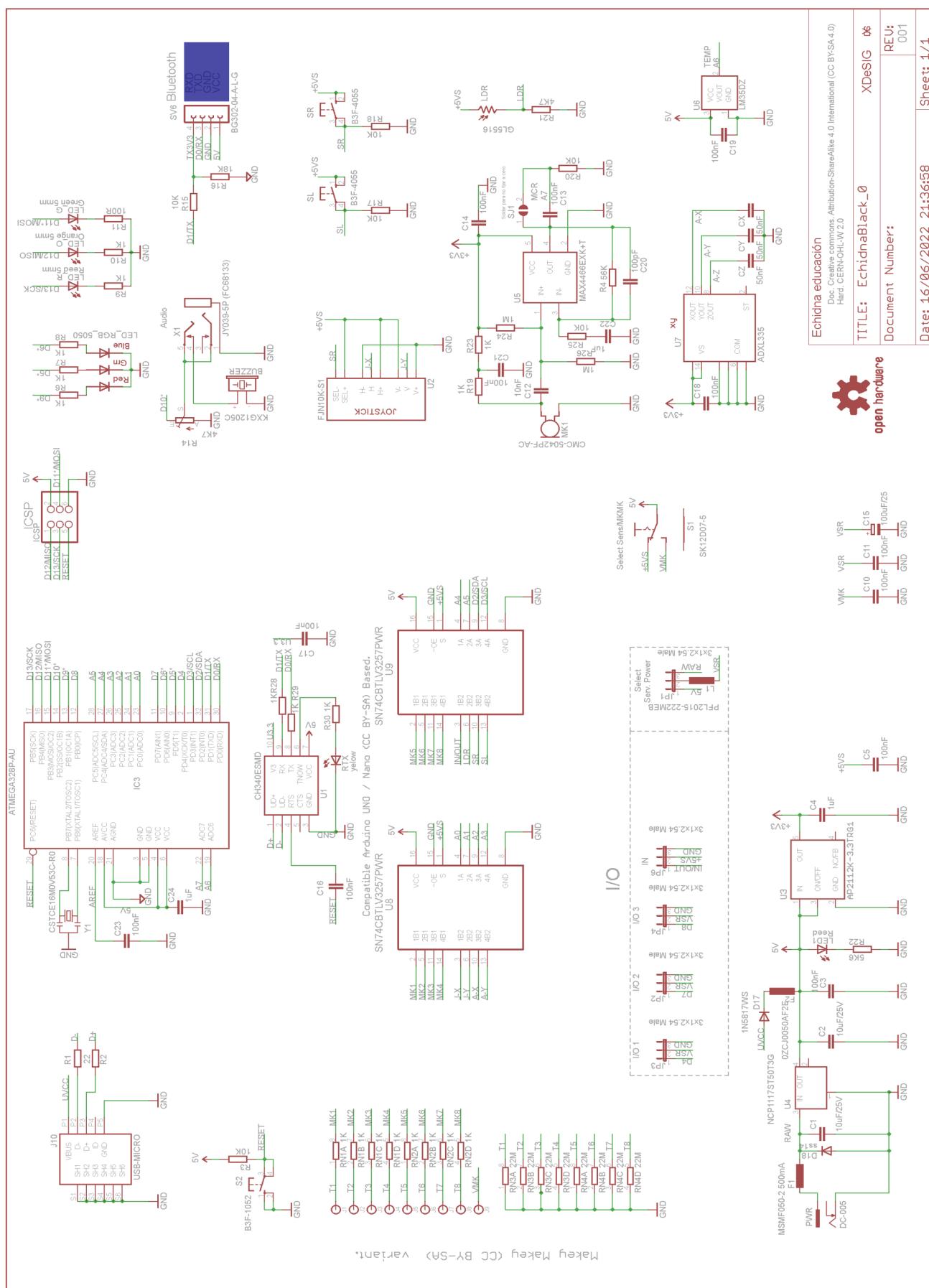
Texto exemplarizando a instrución de lectura/escritura do compoñente.

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

Texto suliñado atallo a datos na interrede.

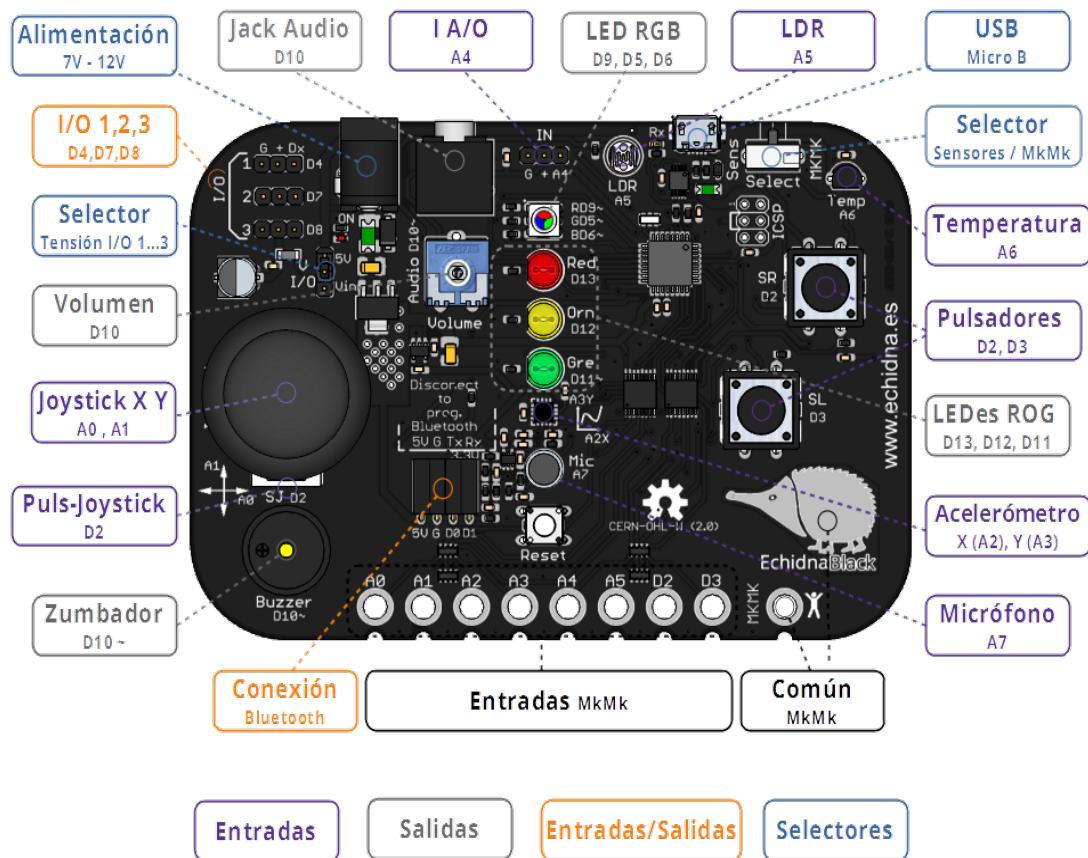
<https://echidna.es/hardware/echidnablack/>





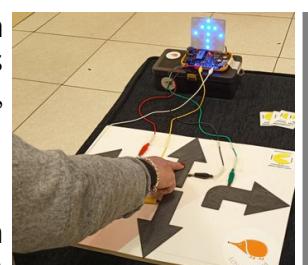


Entradas/ saídas



Conta con múltiples entradas e saídas, algunas son só entradas ou saídas, pero tamén contamos con catro conexións que poden funcionar como entradas e saídas, dependendo da configuración que fagamos no noso programa.

Temos entradas tipo MkMk, que se activan cando pasa unha pequena corrente entre o común MkMK (+5V) e calquera das entradas MkMk, tocando cos dedos, conectando unha gominola, unha froita, plastilina condutora ou pintando teclas con lapis...



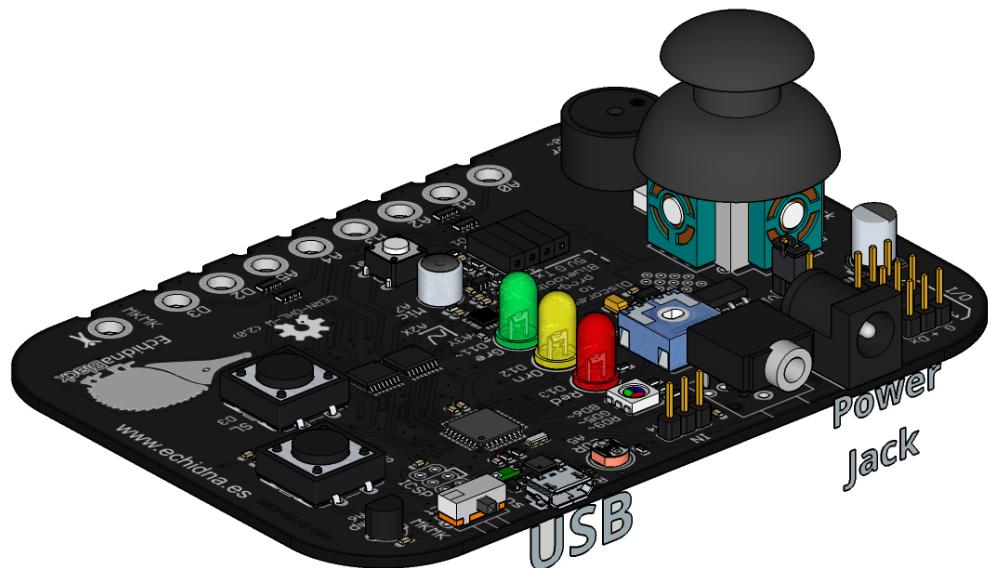
O cerebro é o microcontrolador ATmega328P, con unha arquitectura RISC, podendo realizar un millón de operacións cada segundo, conta con memoria flash de 32KB , memoria Ram de 2Kb e 1KB de EEPROM. Contamos con 23 liñas de entrada/saída das cales 6 tamén son entradas analóxicas de 10 bits. Unha canle de comunicación serie TxR/RxD. Non te preocupes de memorizar a que pin do microcontrolador está conectada cada cousa, a “chuleta” está impresa na propia placa ;-).



Alimentación da placa

Temos dúas formas:

1. Mediante a conexión USB. Esta forma é válida para a maioría das funcións, toda a placa recibe unha tensión regulada de 5V proporcionada polo cable USB, xeralmente cun límite de 500mA EchidnaBlack conta con fusible rearmable protexendo o noso ordenador dos posibles sobre-consumos dos nosos experimentos.
2. Mediante o Jack de alimentación. Todas as partes reciben unha tensión regulada internamente de 5V e 1A máx. Danos a posibilidade de seleccionar Vin para alimentar os **pines I/O ca tensión** e potencia que proporciona o **alimentador externo**, esta conexión conta con fusible rearmable de protección (ollo non superar 1A de corrente constante).





Selector de Alimentación I/O

A ponte “Jumper” de alimentación permite seleccionar a alimentación das I/O.

| | |
|-------------|--------------|
| | |
| Selector 5V | Selector Vin |

Selector alimentación 5V:

No caso de querer alimentar as I/O dende os 5 Volts procedentes do regulador integrado, a ponte ten que estar colocado como indica a imaxe da esquerda. Aconsellado para sensores externos que precisen duna tensión estabilizada.

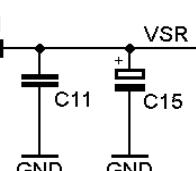
¡Non utilizar a alimentación 5V cando os dispositivos conectados consuman máis de 500 mA! Do contrario sobrepasaríase o regulador.

De ser preciso podemos alimentalos mediante unha alimentación suplementaria, mantendo o GND común entre ambos equipos.

Selector alimentación Vin (Alimentación externa):

Aconsellado para alimentar servos ou outros dispositivos conectados a I/O que consuman máis de 500 mA con alimentación externa no Jack de alimentación.

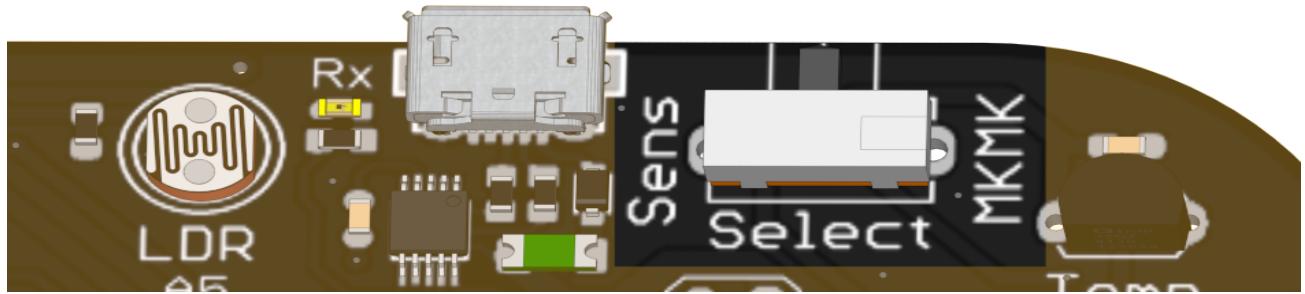
A alimentación nos pinos I/O “ V_{SR} ” conta cun filtro L-C para evitar que cheguen os parasitos eléctricos dos motores o resto dos compoñentes.





Selector Modo Sensores/Modo MkMk

Mediante o conmutador cambiamos entre **Modo Sensores** e **Modo MkMk**.



Modo sensores

No modo sensores temos activos todos os compoñentes da placa exceptuando as conexións MkMk.

Modo MkMk

Neste modo temos activos:

As conexións MkMk.

O sensor de temperatura.

Micrófono.

As saídas:

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



Descripción dos componentes:

Saídas

LED

Acrónimo de “Light Emitting Diode”, baseado no fenómeno da electroluminiscencia. Úsanse como testemuñas (indicadores) e como fonte de iluminación.

O aplicar tensión nos pins do LED este emite luz, podemos controlar o acendido e apagado programando o pin como saída dixital “1” acende, “0” apaga, se está conectado a unha saída PWM “~” podemos axustar o brillo.

Tódolos LEDes conectase con unha resistencia serie “Rs” para controlar a tensión/corrente aplicada. ([exemplo de calculo da resistencia para un led](#) pág. 138).

Temos tres LEDes conectados as seguintes saídas do microcontrolador:

D11~ “Gre” (Verde): Saída Dixital e PWM.

D12 “Orn” (laranja): Saída Dixital.

D13 “Red” (vermello): Saída Dixital.

Para manexalos LEDes podemos facelo de dúas formas, dependendo da saída na que esta conectado:

Dixital: Podemos modificar o estado enviando “0” apagado e un “1” acendido.

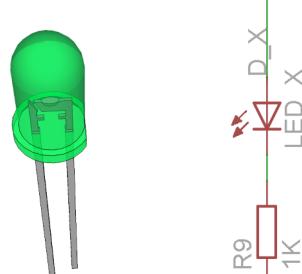
```
digitalWrite(Red, 1); ou digitalWrite(Red, HIGH); //Acendido
```

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

PWM: Na saída D11~ “Gre” podemos enviar o valor de intensidade luminosa entre 0 apagado e 255 acendido completo.

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

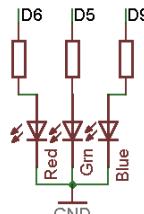
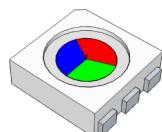
Folla de datos: <http://www.sparkfun.com/datasheets/Components/LED/COM-09590-YSL-R531R3D-D2.pdf>





LED RGB.

Acrónimo de Light Emitting Diode e Red Green Blue, son tres LEDes: Vermello, verde e azul na mesma cápsula.



Podemos controlar o LED RGB dixitalmente, acendendo ou apagando cada LED e axustar a luminosidade mediante PWM.

Control dixital: podemos acender e apagar cada un dos LEDes que forman o RGB, permite formar $2^3=8$ cores diferentes.

Control “analóxico” PWM: podemos controlar o brillo de cada LED, variando o tempo que está encendido e apagado, o que da a sensación de variación luminosa dende 0 a 255 (8 bits), permite realizar $2^8*2^8*2^8= 16,7$ millóns de cores.

Como os LEDes anteriores, conectámoslos con unha resistencia serie “Rs” para controlar a tensión/corrente aplicada.

Este RGB esta conectado a saídas “~”que poden ter control PWM:

D9~ “RGB_R” (vermello): Dixital e PWM

D5~ “RGB_G” (verde): Dixital e PWM

D6~ “RGB_B” (azul): Dixital e PWM

A programación é a mesma que usamos nos outros LEDes:

Dixitalmente (máximo 8 cores):

```
digitalWrite(RGB_R, 1); ou digitalWrite(RGB_G, HIGH);
```

```
digitalWrite(RGB_G, 1); ou digitalWrite(RGB_G, HIGH);
```

```
digitalWrite(RGB_B, 0); ou digitalWrite(RGB_G, LOW);
```

Analoxicamente (máximo 16 millions de cores):

```
analogWrite(RGB_R, 128);
```

```
analogWrite(RGB_G, 128);
```

```
analogWrite(RGB_B, 0);
```

Folla de datos: [http://www.farnell.com/datasheets/2003905.pdf?](http://www.farnell.com/datasheets/2003905.pdf?ga=2.248743815.1083922617.1497294090-1303058009.1492504873)

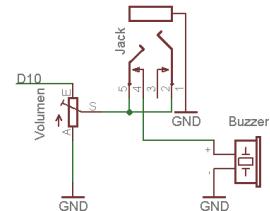
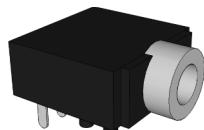
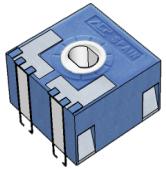




Son.

Dispoñemos de dúas saídas para reproducir son, o zumbador e o jack no que podemos conectar auriculares ou altofalantes autoamplificados. O conectar una clavixa (3,6mm) no jack, desconectase o zumbador.

Podemos axustar a amplitude do son mediante o potenciómetro de Volume.



Na saída D10~ podemos reproducir sinais entre 31 Hz e 20 KHz.

O zumbador o ser excitado por un sinal con unha frecuencia determinada, vibra reproduciendo un son. Temos que ter en conta que a súa frecuencia central é 2,3 KHz, co que si nos afastamos moito desa frecuencia non soará con calidade.

Si na saída do jack conectamos un equipo externo de son permítenos reproducir os sinais con máis calidade.



Si conectamos uns auriculares a alto volume, pode chegar o limiar da **dor nos oídos** e danar os auriculares.



Se baixamos o nivel do volume do “zumbador” pode non resultar audible.

D10~ “Buzz”: Saída PWM (modula en anchura de pulso) .

Activase como unha saída PWM con valor entre 0 e 255 que modula a frecuencia do son.

Temos dúas instrucións que poden facer este cometido

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

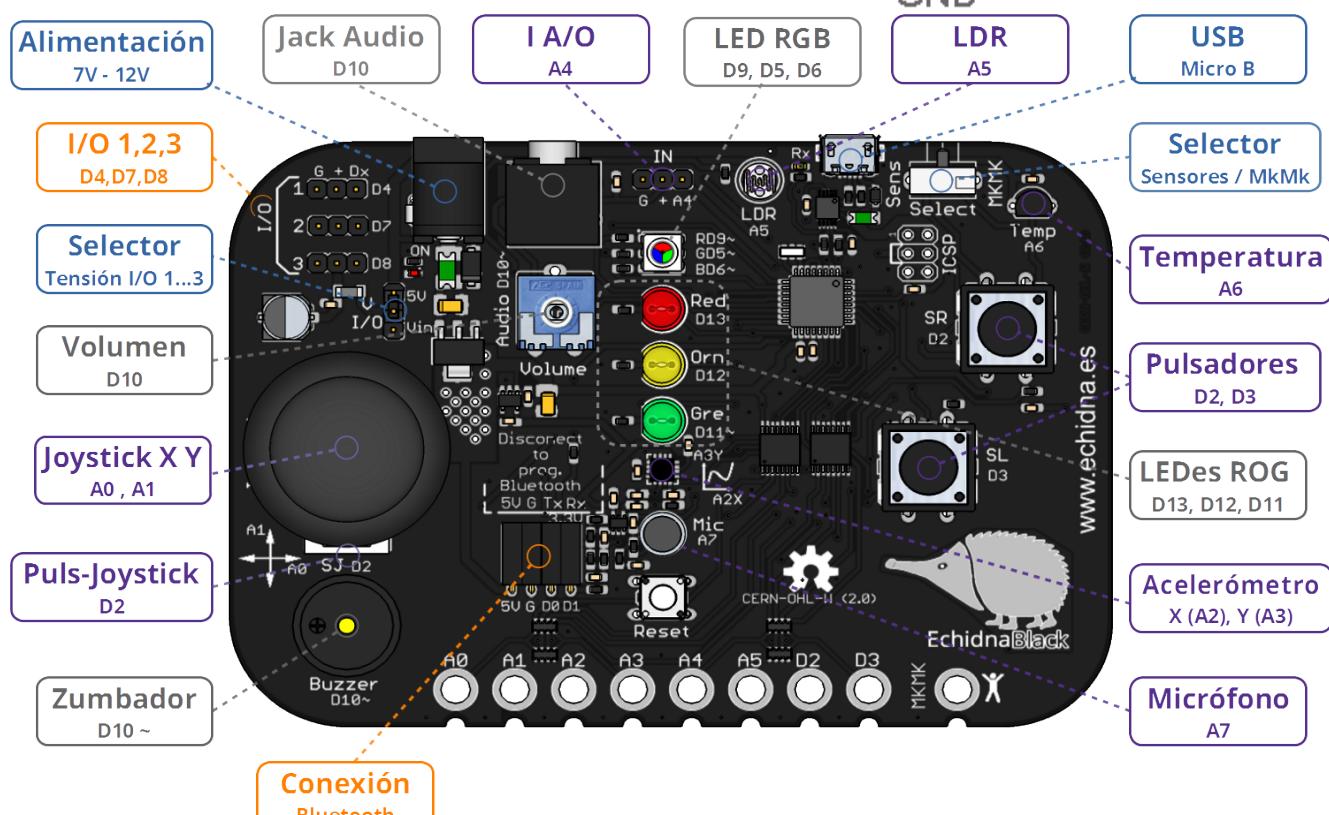
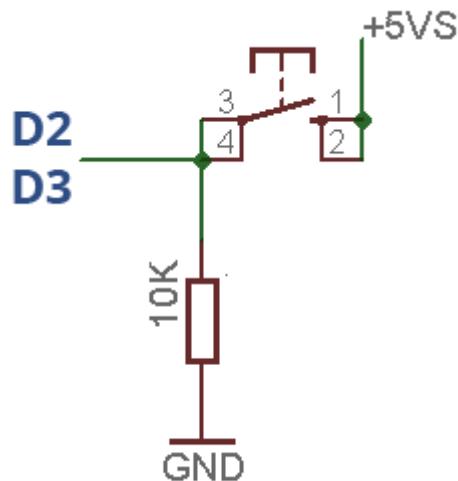
Folla de datos: <https://docs.google.com/file/d/0B1T3xR6vq4KRaG0yTjBBXzB3Y2M/edit>



Entradas (Sensores)

Pulsadores.

Un “pulsador” é un compoñente electromecánico que permite abrir ou pechar un circuíto, conta con un só estado estable.



Entradas

Salidas

Entradas/Salidas

Selectores



Conectado cunha resistencia a masa formando un circuíto denominado pull-down, que proporciona un “0” (0V) sen pulsar e un “1” (5V) cando pulsamos.

Están conectados os pins

D2 “SL” : Entrada dixital

D3 “SR” : Entrada dixital

Para ler os “pulsadores” usamos a instrución de lectura dixital.

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

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

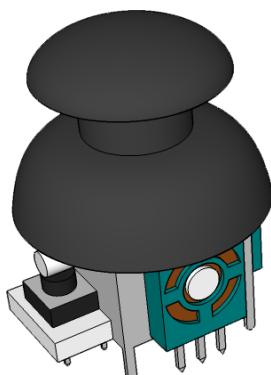


Nota: boolean: Ver páxina 52 “Tipos de datos”

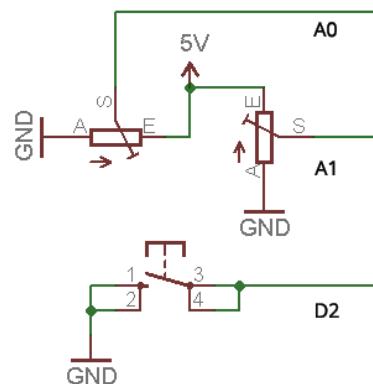


Panca de xogos “Joystick”

É un mando formado por dous potenciómetros un para o eixo X e outro para o eixo Y. Este modelo conta cun “pulsador” a mais.



Imaxe Alfredo B.



Permite converter o movemento de mando nun valor resistencia proporcional, ao conectarlo en modo divisor de tensión, obtemos unha tensión proporcional de 0 – 5V na saída de cada eixo (X, Y).

Na posición de repouso os valores rondan os 2,5V (520 de lectura analólica) *, o “pulsador” do Joystick está conectado co “pulsador” **SR**.

A0 “Joy_X”: Entrada analólica

A1 “Joy_Y”: Entrada analólica

D2 “SJ” = “SR”: Entrada dixital

Lectura e almacenamento dos valores del Joystick:

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

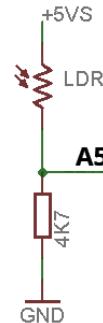
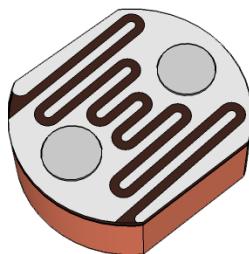
Folla de datos: <http://www.polyshine.cn/uploadFile/FJN10K-2014-01-03-07-24-37.pdf>

Nota: `int`: Ver páxina 52 “Tipos de datos”



Sensor de Luz: LDR.

LDR acrónimo de “Light Dependent Resistor”, é una resistencia onde o seu valor depende da cantidade de luz que lle incide. Xeralmente fabricase con sulfuro de cadmio.



A LDR proporciona valores de resistencia altos (algúns $M\Omega$) con pouca luz e valores baixos con moita luz. Grazas a conectarla cunha resistencia en serie formase un divisor de tensión que consigue inverter a lóxica de funcionamento; de forma que con moita luz proporciona valores altos de tensión $4,9V = 999$ (de lectura analólica) * e valores baixos con pouca luz ($0,0V = 000$).

Non só e sensible a luz visible, a súa resposta óptica vai por baixo dos 400nm (ultravioleta) a os 800nm (infravermellos), sendo máis sensible o redor dos 550nm (verde-amarelo)

* Debido as tolerancias dos componentes estes valores poden ser distintos en cada placa.

A5 “LDR”: Entrada analólica

Lectura e almacenamento dos valores da LDR:

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

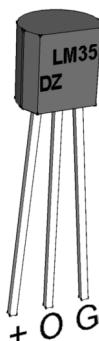
Folla de datos:

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

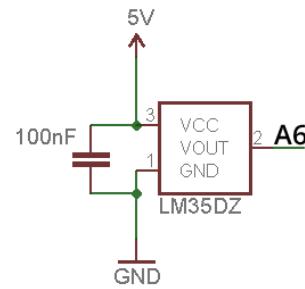


Sensor de temperatura LM35.

Esta calibrado a 10mv cada 1°C , a saída é lineal de -55°C to 150° , aquí só podemos medir temperaturas positivas (non conxeles a Echidna).



- + = Vcc 4 a 20V
- O = Sinal de saída $+10\text{mV}/^{\circ}\text{C}$ (V_o)
- G = Masa alimentación
- A6 "Temp": Entrada Analólica



Tendo en conta que proporciona 10mv por $^{\circ}\text{C}$. O conversor analóxico-dixital do microcontrolador ATmega328P usa 10bits son 1024 “chanzos” da medida analólica, cunha referencia de 5Vcc podemos obter a temperatura mediante a seguinte operación:

```
int temperatura = (analogRead(A6) * 5.0 * 100.0)/1024; //temp °C
```

Para coñecer a temperatura, primeiro leemos o valor do sensor e logo aplicamos a fórmula para converter cada 10 mV en $^{\circ}\text{C}$

Tendo unha referencia de 5Vcc perdemos bastante precisión xa que poderíamos medir de 0°C a 500°C o que excede moito o rango de medida de este sensor, usando a referencia de tensión interna do conversor analóxico-dixital no ATmega328P “1,1V” podemos medir de 0 a 110°C que se aproxima máis o rango do LM35 e as medidas que imos realizar.

```
analogReference(INTERNAL); // Referencia analox. a 1.1V
float temperatura = (analogRead(A6) * 1.1 * 100.0)/1024; //temp °C
```

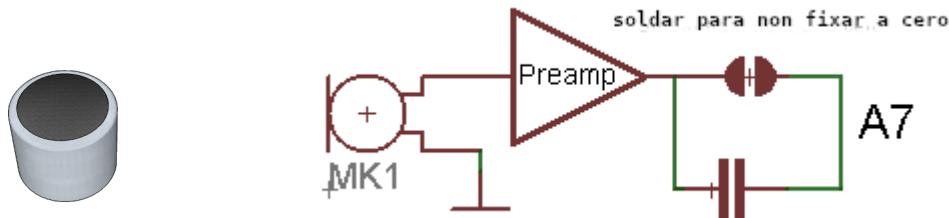
Folla de datos: <http://www.ti.com/lit/ds/symlink/lm35.pdf>

Nota: float: Ver páxina 52 “Tipos de datos”



Micrófono.

É un transdutor acústico-eléctrico piezoeléctrico que entrega un sinal eléctrico de similares características do son que recibe.



Varia dende cero a un valor de 3V aprox., mostrado na figura 1, se realizamos unha pequena soldadura no selector da cara inferior (entre as patas dianteiras da fig. Echidna) o sinal que nos proporciona agora está centrada en 1,75V= 360 de sinal analóxica, o resultado mostrase na figura 3.

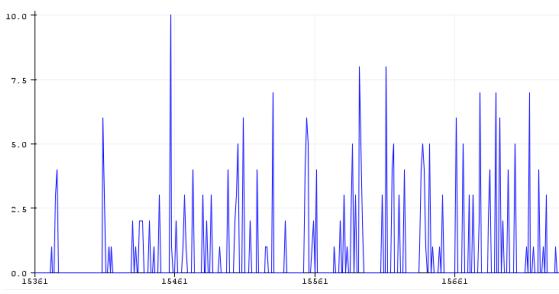


Fig. 1

A7 “Mic”: Entrada Analólica

O igual que no caso anterior podemos usar referencia interna de 1,1V para mellorar sensibilidade:

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

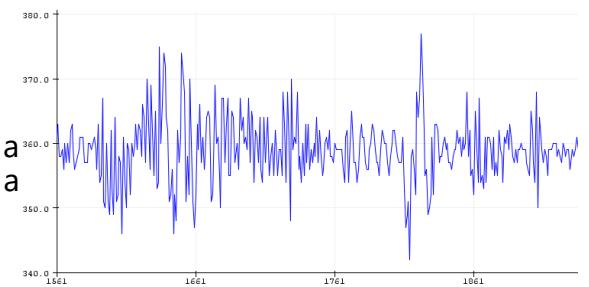


Fig. 3

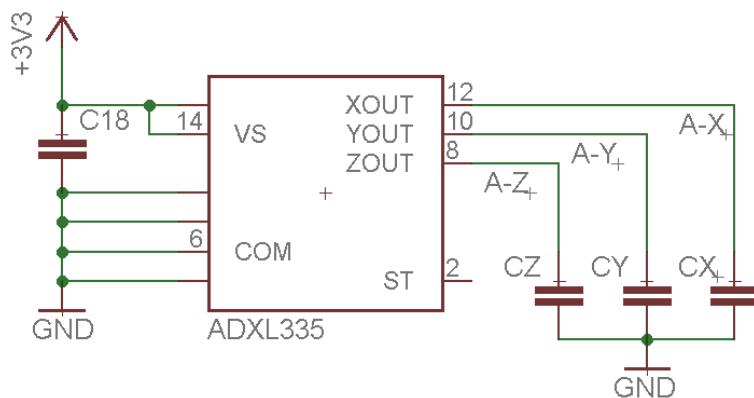
Folla de datos: <https://www.mouser.es/datasheet/2/670/cmc-5042pf-ac-1776337.pdf>



Acelerómetro.

Sensor de aceleracións, está baseado en condensadores diferenciais dentro dunha estrutura micro-mecanizada.

Proporciona una tensión dependente da aceleración en cada eixo.



Mide a aceleración nos eixos (X, Y e Z) nun rango de +- 3g, na posición de reposo proporciona un valor de 1,75V = 359 (de lectura analólica)* e no extremos os valores 1,39V = 285* - 2,10V = 428*.

En EchidnaBlack non esta conectada a saída Z

* Debido as tolerancias dos componentes estes valores poden ser distintos en cada placa.

A2 "Ace_X": Entrada analólica

A3 "Ace_Y": Entrada analólica

Lectura e almacenamento dos valores del sensor:

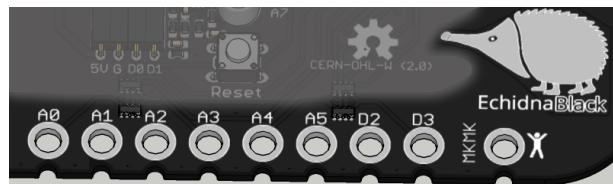
```
int Valor_Ace_X = analogRead(A2);
int Valor_Ace_y = analogRead(A3);
```

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



Entradas (Modo MkMk).

Dispoñemos de 8 conexións MkMk. Cada conexión é parte dun divisor de tensión con unha resistencia moi elevada, polo que o conectar un elemento que conduce electricidade, inda que sexa tenuemente, o cerrar o circuíto co punto común, a tensión da entrada sube e pode ser detectada. A tensión limiar de detección depende de cada material / persoas. (O logo Echidna tamén se comporta como Común).



Podemos ler as entradas MkMk como si dunha entrada dixital se tratara, pero tamén podemos ler as entradas A0..A5 analoxicamente e comparadas cun valor limiar para tomar decisións.

A0 MkMk0: Entrada analóxica

A1 MkMk1: Entrada analóxica

A2 MkMk2: Entrada analóxica

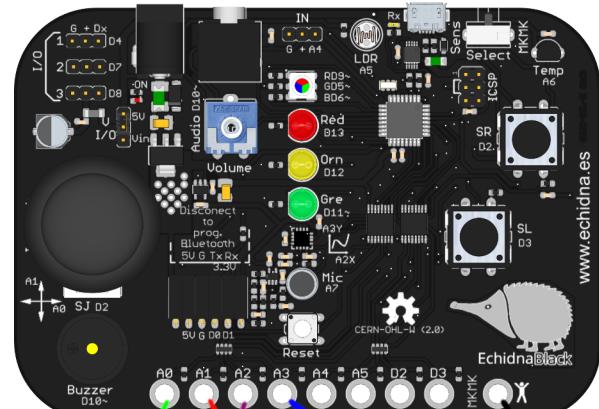
A3 MkMk3: Entrada analóxica

A4 MkMk4: Entrada analóxica

A5 MkMk5: Entrada analóxica

D2 MkMk6: Entrada dixital

D3 MkMk7: Entrada dixital



Lectura dixital:

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



Lectura analóxica:

```
int umbral = 600;
int valorMkMk0 = analogRead(A0);
if (valorMkMk0 >= umbral){
// aquí o que queiras fazer
}
```



Pines de libre disposición.

Contamos con catro entradas/saídas de libre disposición, tres dixitais D4, D7, D8 e unha entrada analóxica e ou saída dixital A4.



Cada I/O e IN conta con tres pines:

$G = GND$ (negativo, común).

$+ = +V$ (positivo)

Para I/O ver "[Selector de Alimentación](#)"

Para IN +5Vcc

$Dx = \text{entrada / saída}$

$A4 = \text{Entrada analóxica ou dixital, saída dixital.}$

Podemos conectar diferentes dispositivos, como servo, magnetómetro, higrómetro, pantalla e un longo etcétera.

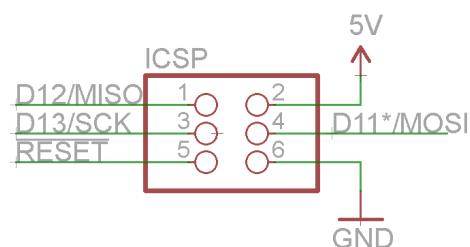
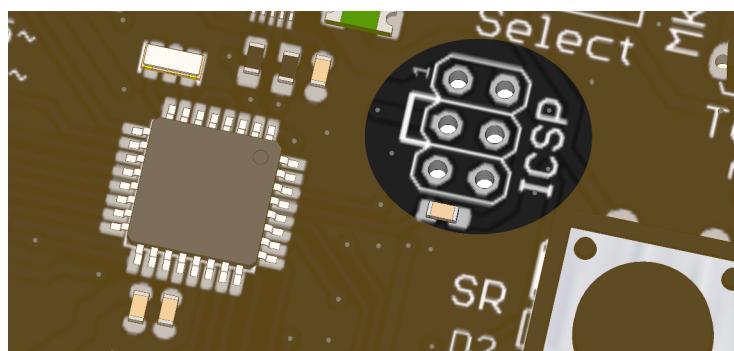
A disposición de pines elixiuse para ter compatibilidade con moitos dispositivos, pero non todos son compatibles pin a pin, é preciso comprobar a disposición das conexións dos teus dispositivos, para evitar escangallalos.



Conector ICSP “In-Circuit Serial Programming”.

Por defecto EchidnaBlack ven preprogramada co cargador de arranque “bootloader” OPTIBOOT (GPLv2 WRT), que nos permite enviarlle os programas vía USB, e compatible con tódolos IDE de Arduino (ArduinoNano).

Mediante o conector ICSP podemos reprogramar o microcontrolador tendo acceso a memoria de programa (flash) sin ter que utilizar o bootloader, podemos cambiarlle o cargador de arranque, ou por outro Firmware. (Para o traballo normal non é preciso utilizalo.)



MISO (D12) – Master In Slave Out



MOSI (D11) – Master Out Slave In

SCK (D13) – Signal Clock

RESET – Reinicio do microcontrolador.

5V (VCC) – Voltaxe

GND – Terra (común)

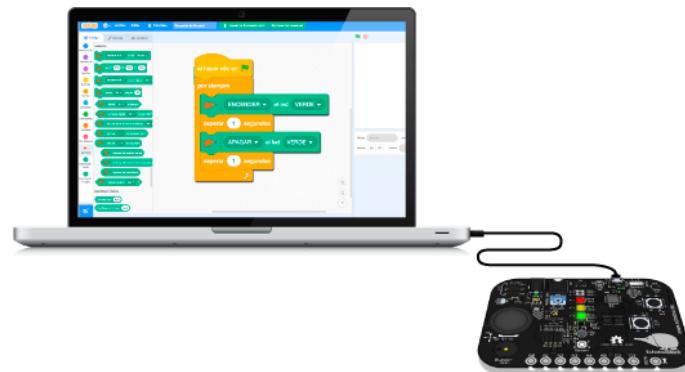
Pode ser que algunha vez (rara) non se cargue correctamente o programa que queremos e o microcontrolador se quede “Briquedoado” bloqueado e xa non podamos programalo, podemos “Desbriquelo” volvendo a subir o bootloader mediante o conector ICSP. Ver “[Desenladrillar](#)“



Programación (falemos con EchidnaBlack)

Temos bastantes formas para programala:

- Mediante contornas gráficas, é preciso pasarlle algún programa específico o 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>”
- ...



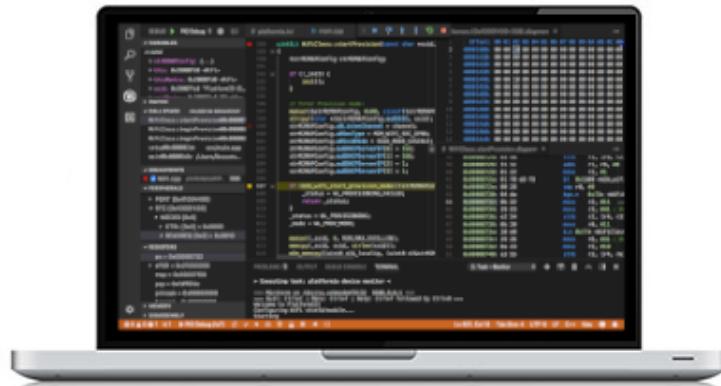
- Mediante liñas código
 - ASM
 - C
 - Wiring “<http://wiring.org.co>”
 - Arduino (C++ modificado) é o que imos usar neste manual.

Para escribir os programas e pasarllos a EchidnaBlack precisamos usar unha diversidade de programas: editor, compilador e programador, pero temos uns programas que xa integran todo o necesario para poder programar comodamente,



chámense contornas de de desenvolvemento integrado (Integrated Development Environment) IDE, entre outras temos:

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



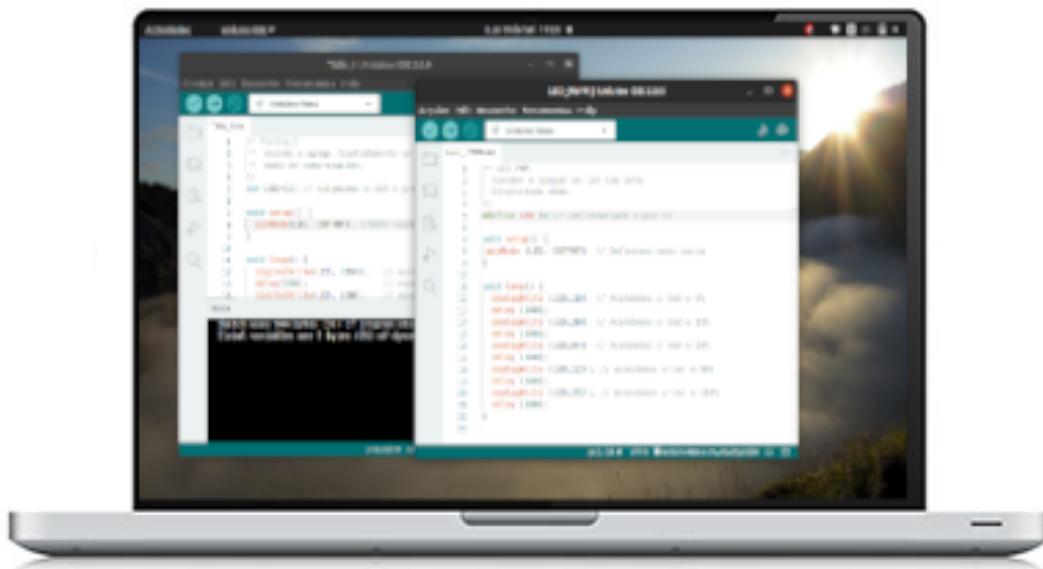
PlatformIO o meu parecer e a más

aconsellable das IDE, pero neste manual imos usar o **Arduino IDE 2.0.0**, aconséllote que probes outras IDEs e usa a que más cómoda atopes (ollo, algúna e de pago).

IDE de Arduino

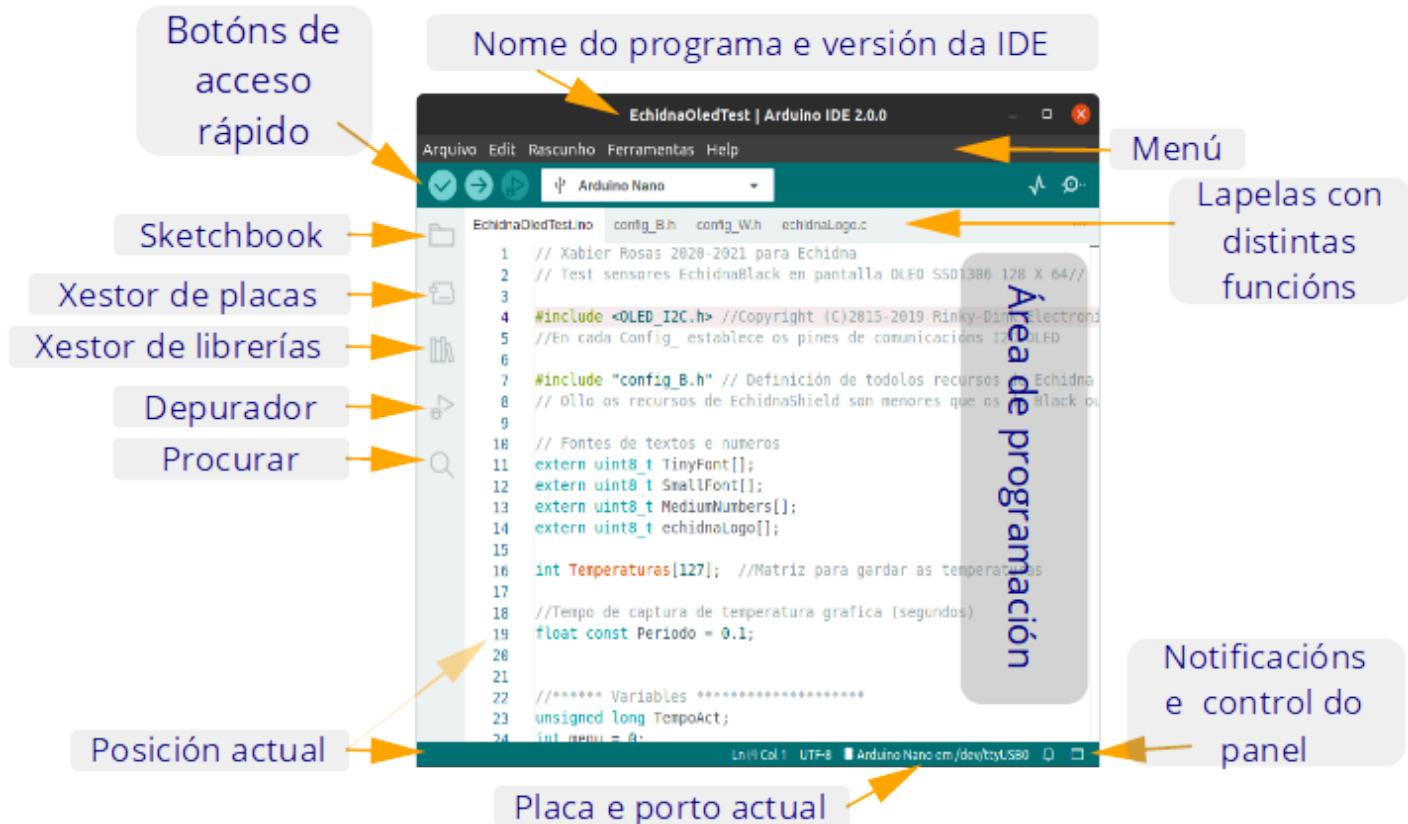
Descarga a e instala no ordenador (a instalación variará dependendo do sistema operativo que teñas)

Na Interrede temos moitos manuais de instalación dos que podes votar man.





Unha vez instalado o IDE Arduino imos a coñecelo:





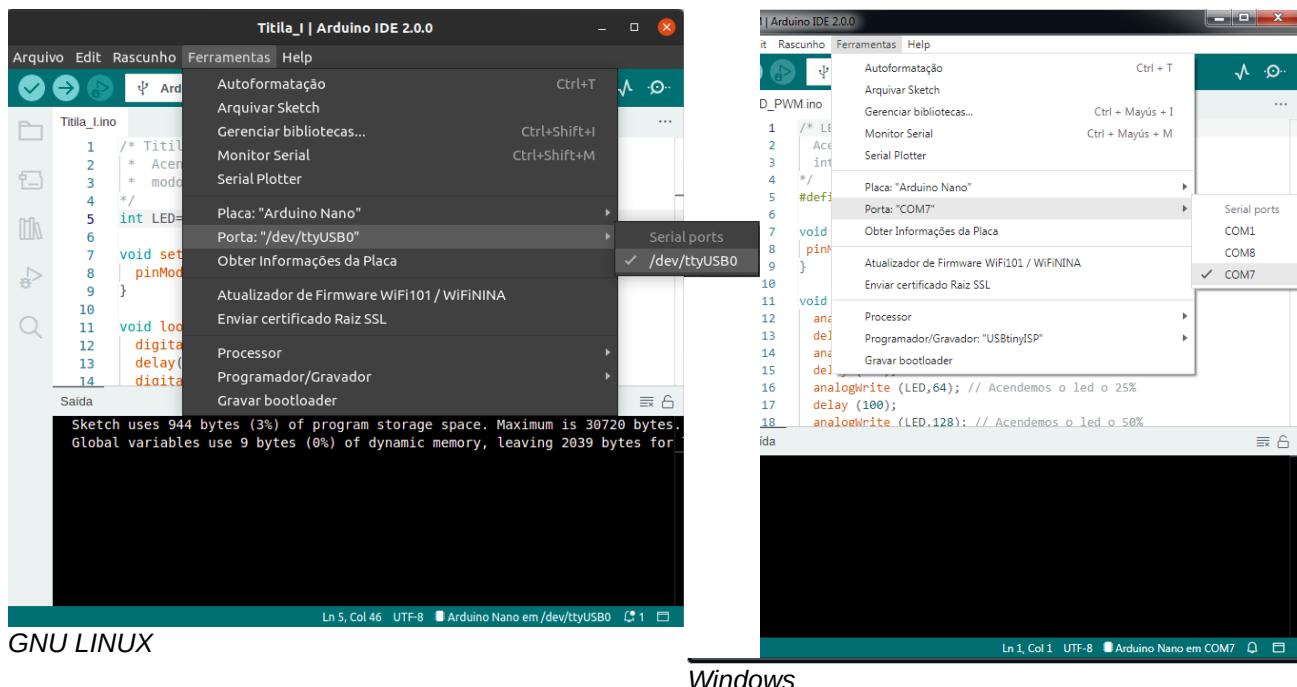
Para subir os programas a nosa EchidnaBlack temos que seleccionar a placa Arduino Nano e o porto serie correspondente, este pode variar en cada conexión USB que utilicemos.

Usamos o chip de comunicacóns, CH341.

Nos sistemas operativos GNU Linux, non é preciso instalar ningún controlador “Driver” USB para comunicarnos con EchidnaBlack,

En MAC OS a veces non se recoñece o dispositivo ou aparece unha mensaxe de **Kernel Panic**. Para solucionalo descargamos a versión actualizada do “driver” [CH341](#) (pedirá reinicio) temos permitir a sua execución en permitir apps descargadas.

En Windows tamén temos que instalar o controlador [CH341](#)





Estrutura do programa “Sketch” de Arduino:

Todos os Sketch de Arduino teñen dúas funcións básicas:

1. void setup()

É unha función que se executa unha soa vez, cando se inicia o sketch. Usase para inicializar comunicacións, axustar o modo dos pins, inicio do uso de dispositivos, librerías...

2. void loop()

Fai precisamente o que o seu nome suxire, é un bucle, executa consecutivamente as instrucións do programa.

The screenshot shows the Arduino IDE interface with the title bar "Titila_I | Arduino IDE 2.0.0". The menu bar includes "Arquivo", "Edit", "Rascunho", "Ferramentas", and "Help". The toolbar has icons for file operations like Open, Save, and Print. The board selection dropdown shows "Arduino Nano". The code editor displays the following sketch:

```
Titila_I.ino
1  /* Titila_I
2  * Acende e apaga repetidamente un led e EchidnaBlack
3  * modo de comprobación.
4  */
5  int LED=12; // asignamos o LED o pin D12 (Led laranxa)
6
7  void setup() {
8      pinMode(LED, OUTPUT); //Modo saída
9  }
10
11 void loop() {
12     digitalWrite(LED, HIGH); // acende o LED HIGH = "1" = +5V
13     delay(100); // espera 100 mili segundos
14     digitalWrite(LED, LOW); // apaga o LED LOW = "0" = 0V
15     delay(500); // espera 500 mili segundos
16 }
17
```

The status bar at the bottom shows "Ln 12, Col 32 UTF-8" and "Arduino Nano em /dev/ttyUSB0".

Comentar o código é unha boa practica, Axúdanos a elaborar o programa e entendelo pasado un tempo. Os comentarios non afectarán a operación normal do



programa, non se compilan son só para nós, a liña que escribamos tras “//” é un comentario. Se precisamos escribir máis liñas podemos establecer un bloque de comentario comenzando por “/*” e rematando por “*/”

Cando escribimos un programa, necesitamos a empaquetar un conxunto de instrucións (funcións, bucles..), pónense entre chaves “{}”.

Usase “;” para dicirlle o compilador que esta instrución rematou.

As entradas e saídas dos microcontroladores son configurábeis e normalmente dixitais (só entenden ceros e uns):

$$\text{“0” Cero} = -0,5V \text{ a } 0,3^*Vcc \quad \text{“1” Un} = 0,7^*Vcc \text{ a } Vcc + 0,5V$$

Algunhas entradas saídas poden ser analóxicas (entenden valores que van de 0V a Vcc) no ATmega328P que ten a EchidnaBlack, temos oito (8) entradas analóxicas e ningunha saída analólica, pero si contamos con seis (6) saídas PWM de 8 bits.

Para axustar como se van comportar usamos:

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

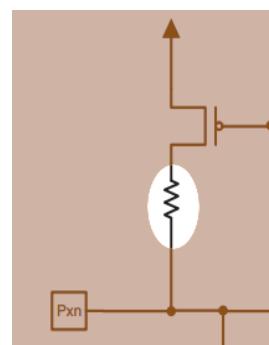
pin: especifica o numero de pin (0..13) (A0..A7)

modo: configura o modo de funcionamento, que en Arduino é:

OUTPUT: saída.

INPUT: entrada.

INPUT_PULLUP: entrada con unha resistencia de 10KOhms conectada a +VCC, así non temos que por unha resistencia externa.



Para ler unha entrada:

```
digitalRead(pin);
```

Lectura dixital, devolta un “1” ou “0”, pin de 0..13 e A0..A5.

```
analogRead(Ax);
```

Lectura analólica , devolta un valor de 10 bits A0..A71



Escribir nunha saída:

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

Saída dixital pin 0..13 e A0..A5, valor pode ser un 0=LOW ou 1=HIGH.

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

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

Se poñemos a saída PWM = 0 é o mesmo que apagado

Se o valor é 128 a saída terá a metade do tempo en 0 e a outra metade en 1. o mesmo que 50%

A esta relación chamámoslle ciclo de traballo “duty cycle”, permite controlar a iluminación dun led, a velocidade de xiro dun motor...

50% duty cycle C.C.Thewrightstuff



75% duty cycle



25% duty cycle





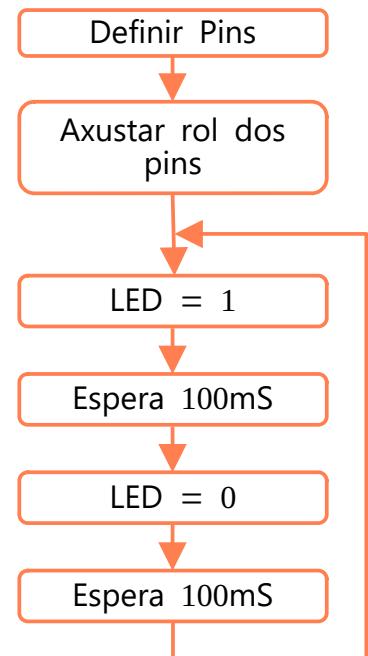
Sketch “Titila_I”.

Xa podemos facer un pequeno programa que acenda e apague un led. Iniciamos o IDE Arduino e escribimos o programa.

```
/* Titila_I
 * Acende e apaga repetidamente un LED
 * A modo de comprobación.
 */
#define LED 12 // asignamos o LED o pin D12 (Led laranxa)

void setup() {
    pinMode(LED, OUTPUT); //Modo saída
}

void loop() {
    digitalWrite(LED, HIGH); // acende o LED HIGH = "1" = +5V
    delay(100); // espera 100 milisegundos
    digitalWrite(LED, LOW); // apaga o LED LOW = "0" = 0V
    delay(500); // espera 500 milisegundos
}
```



💡 Analise:

“#define LED 12” Asigna a constante 12 a LED, As constantes definidas non ocupan espacio de memoria, o compilador substituirá a referencia LED co 12 no momento da compilación. Así si quixeramos cambiar o valor do pin asignado o LED , só temos que cambialo na cabeceira do programa, e non ir cambiando liña a liña.

`delay(100);`

Esta é unha declaración de atraso, o que significa que LED pódese acender ou apagar durante 100 milisegundos, fíxate que poñemos un atraso de 500mS tras apa-gar o LED, así o apagado dura máis que o acendido.



Subir o programa.

Agora podemos verificar que a sintaxe é correcta, pincha en verificar, tamén podes pincha no botón subir, que tamén verifica a sintaxe, se non temos erros o subira a EchidnaBlack.

Xa temos a EchidnaBlack conectada o ordenador, seleccionamos placa ArduinoNano, procesador ATmega328 e o porto de comunicacóns asignado.

The screenshot shows the Arduino IDE 2.0.0 interface. The title bar says "Titila_I | Arduino IDE 2.0.0". The menu bar includes "Arquivo", "Edit", "Rascunho", "Ferramentas", and "Help". The toolbar has icons for save, upload, and refresh. The central code editor window displays the "Titila_I.ino" sketch:

```
1  /* Titila_I
2   * Acende e apaga repetidamente un led e EchidnaBlack
3   * modo de comprobación.
4   */
5  int LED=12; // asignamos o LED o pin D12 (Led laranxa)
6
7  void setup() {
8    pinMode(LED, OUTPUT); //Modo saída
9  }
10
11 void loop() {
12   digitalWrite(LED, HIGH); // prende o LED HIGH = 5V
13 }
```

The "Saida" (Serial Monitor) window at the bottom shows the output of the avrdude command:

```
avrduude: reading on-chip flash data:
Reading | ##### | 100% 0.13s
avrduude: verifying ...
avrduude: 944 bytes of flash verified
avrduude done. Thank you.
```

At the bottom of the IDE window, status information includes "Ln 5, Col 46 UTF-8", "Arduino Nano em /dev/ttyUSB0", and a connection icon.

Podemos ver na imaxe que o programa subiuse correctamente, e usa moi pouca memoria 932 bytes da memoria de programa e 9 bytes de memoria ram.

Nota: Atoparemos en moitos exemplos a asignación de entradas/saídas como si de un valor se tratara “`int LED=12;`”, isto ocupara memoria, neste programa non importa temos moito sitio, pero non é preciso sacrificar espacio de memoria para unha simple substitución en tempo de compilación “`#define LED 12`”.

- Proba a acender o resto dos LEDes que ten EchidnaBlack.



LED_PWM.

Agora toca manexar a intensidade de algúns LEDes conectados a saídas que permiten PWM son os marcados con “~” : RD9, GD5, BD6, e Gre D11.

```
/* LED_PWM
   Acender e apagar un led con unha
   intensidade dada.
*/
#define LED 11 // Led conectado o pin 11

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

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

⌚ Analise:

Este exemplo non é o máis axeitado para manexar a intensidade dos LEDes, xa que temos que escribir cada un dos valores. Para mellorar isto usaremos a estrutura de programación de repetición **for**.



Operadores comúns:

Operadores aritméticos (A =2, B = 4)

| Operador | Nome | Descripción | Exemplo |
|----------|----------------|---|------------------|
| “ = ” | Asignación | Almacena o valor na variable da esquerda. | A = B dará A = 4 |
| + | Suma | Agrega dous operandos. | A + B dará 6 |
| - | Resta | Resta o segundo operando do primeiro. | A - B dará -2 |
| * | Multiplicación | Multiplica dous operandos. | A * B dará 8 |
| / | División | Divide numerador por denominador. | B / A dará 2 |
| % | Módulo | Resto despois da división. | B % A dará 0 |

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

| Operador | Nome | Descripción | Exemplo |
|----------|----------------|---|---------------------|
| == | Igual a | Comproba si os operando son iguais. | (A == B) dará falso |
| != | Non igual | Comproba si os operando non son iguais. | (A != B) dará certo |
| < | Menor que | Comproba si o operando da esquerda e menor que o da dereita. | (A < B) dará certo |
| > | Maior que | Comproba si o operando da esquerda e maior que o da dereita. | (A > B) dará falso |
| <= | Menor o igual | Comproba si o operando da esquerda e menor ou igual que o da dereita. | (A <= B) dará certo |
| >= | Maior ou igual | Comproba si o operando da esquerda e maior ou igual que o da dereita. | (A >= B) dará falso |



Operadores lóxicos ($A = 2, B = 4$)

| Operador | Nome | Descripción | Exemplo |
|-------------------------|------|---|--------------------------|
| <code>&&</code> | E | Operador lóxico E (And) si ambos son distintos de cero devolta certo. | $(A \&\& B)$ dará certo |
| <code> </code> | OU | Operador lóxico Ou (Or) si calquera dos operadores son distintos de cero devolta certo. | $(A B)$ dará certo |
| <code>!</code> | Non | Operador lóxico Non (Not) usase para inverter o estado lóxico. | $!(A \&\& B)$ dará falso |

Operadores booleanos ($A = 2, B = 3$)

| Operador | Nome | Descripción | Exemplo |
|-----------------------|---------------------|---|--|
| <code>&</code> | E | O operador E (And) binario copia un bit no resultado si existe en ambos operandos | $(A \& B)$ dará 2 que é 10 en binario |
| <code> </code> | OU | O operador OU (Or) binario copia un bit no resultado si existe en calquera dos operandos | $(A B)$ dará 3, 11 en binario |
| <code>^</code> | XOU | O operador binario XOU(XOR) copia o bit si está configurado nun operando pero non en ambos. | $(A ^ B)$ dará 1, 1 en binario |
| <code>~</code> | NON | O operador complementario cambia os uns por ceros. | $(\sim A)$ dará 253 , en binario 1111 1101 |
| <code><<</code> | Despraza a esquerda | Despraza a esquerda o numero de bits indicado polo operando da dereita. | $A << 2$ dará 8, en binario 1000 |
| <code>>></code> | Despraza a dereita | Despraza a dereita o numero de bits indicado polo operando da dereita. | $A >> 2$ dará 0, en binario 0000 |



Operadores compostos (A =2, B = 3)

| Operador | Nome | Descripción | Exemplo |
|---------------------|-------------------------|--|--|
| <code>++</code> | Incremento | O operador incrementará o valor enteiro nunha unidade. | <code>A++</code> dará 3 |
| <code>--</code> | Decrecemento | O operador decrecemento o valor enteiro nunha unidade. | <code>A--</code> dará 1 |
| <code>+=</code> | Suma composta | Suma o operando dereito o operando esquierdo e asina o resultado o operando esquierdo. | <code>B += A</code> é equivalente a <code>B = B + A</code> |
| <code>-=</code> | Resta composta | Resta o operando dereito do operando esquierdo e asina o resultado o operando esquierdo. | <code>B -= A</code> é equivalente a <code>B = B - A</code> |
| <code>*=</code> | Multiplicación composta | Multiplica o operando dereito o operando esquierdo e asina o resultado o operando esquierdo. | <code>B *= A</code> é equivalente a <code>B = B * A</code> |
| <code>/=</code> | División composta | Divide o operando dereito do operando esquierdo e asina o resultado o operando esquierdo. | <code>B /= A</code> é equivalente a <code>B = B / A</code> |
| <code>%=</code> | Módulo composto | Colle o módulo usando os operandos e asina o resultado o operando esquierdo. | <code>B %= A</code> é equivalente a <code>B = B % A</code> |
| <code> =</code> | Operación OU composto | Realiza a operación OU (Or) bit a bit e deixa o resultado no operando esquierdo | <code>A = 2</code> é o mesmo que <code>A = A 2</code> |
| <code>&=</code> | Operación E Composto | Realiza a operación E (AND) bit a bit e deixa o resultado no operando esquierdo | <code>A &= 2</code> é o mesmo que <code>A = A & 2</code> |

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



For.

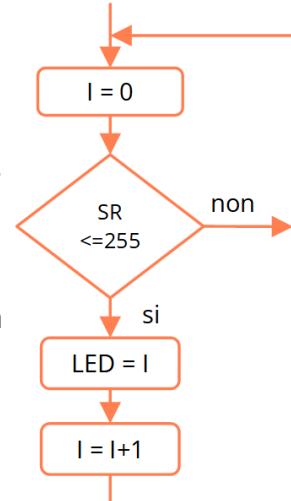
A declaración **for** úsase para repetir un bloque de sentenzas encerradas entre chaves un número determinado de veces.

Ten tres partes separadas por (;). A inicialización da variable local prodúcese unha soa vez e a condición se comproba cada vez que se termina a execución das instrucións dentro do bucle. Se a condición segue cumpríndose, as instrucións do bucle vólvense a executar. Cando a condición non se cumpre, o bucle termina.

Calquera dos tres elementos de cabeceira pode omitirse, áinda que o punto e coma é obligatorio. Tamén as declaracions de inicialización, condición e expresión pode ser calquera estamento válido en lingua xe “C” sen relación coas variables declaradas.

Síntaxe:

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



LED_PWM_2

Incremento do brillo de un dos LEDes RD9, GD5, BD6, e Gre D11. Mediante estrutura de repetición **for**.

```
/* LED_PWM, Variar a intensidade dun led con modulación da anchura de pulso
usando un bucle for.
```

```
*/
```

```
#define LED 11 // Led conectado o pin 11

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

void loop() {
  // Incrementamos 0 dende 0 ata 255 en pasos de 1
  for (int i = 0; i <= 255; i++)
  {
    analogWrite(LED, i); // Escribimos o valor i "PWM" na saída
    delay(10); // facemos unha pausa de 10mS entre incremento
  }
}
```



Cores_RGB.

Podemos acender o led RGB con cores o chou, podemos ter 16 777 216 cores.

```
/* Cores_RGB
cores o chou
*/
#define RGB_R 9 //Led RGB vermelho
#define RGB_G 5 //Led RGB verde
#define RGB_B 6 //Led RGB azul

void setup() {
pinMode (RGB_R, OUTPUT); // Definimos como saída
pinMode (RGB_G, OUTPUT); // "
pinMode (RGB_B, OUTPUT); // "
}

void loop() {
analogWrite(RGB_R, random(255)); // PWM aleatorio vermella
analogWrite(RGB_G, random(255)); // " verde
analogWrite(RGB_B, random(255)); // " azul
delay(500);
}
```

💡 Analise:

Neste programa usase a instrución “random()”, devolta un valor pseudoaleatorio.

Sintaxe:

| | |
|--------------------|---|
| random(max); | Nos devolta un valor que vai de 0 a max |
| random(min , max); | Nos devolta un valor que vai de mim a max |

Probamos a pulsar o botón de “Reset”. Que sucede?

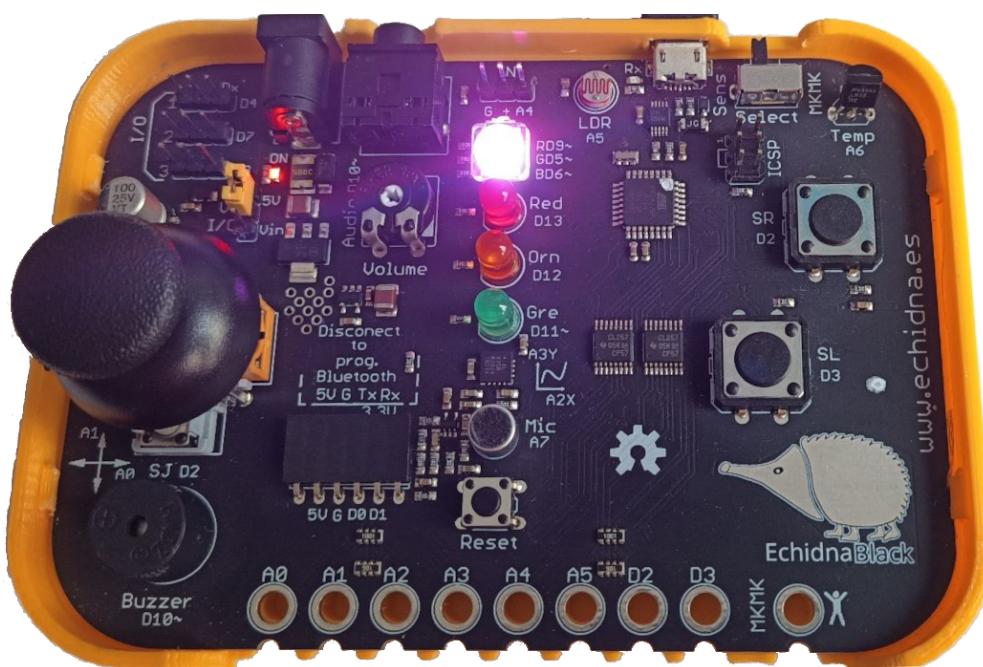


Veremos como sempre fai a mesma secuencia de cores, non é aleatorio real, para corrixir iso, temos que usar unha semente que poidamos coller do “mundo real”, por exemplo dunha entrada analóxica non usada `randomSeed(analogRead(A4))`; nesa entrada non temos nada conectado (ou si?), e utilizar ese valor descoñecido como semente:

Incluímos a seguinte liña tras a definición do modo dos pins

```
void setup() {
    pinMode (RGB_R, OUTPUT); // Definimos modo saída
    pinMode (RGB_G, OUTPUT); // "
    pinMode (RGB_B, OUTPUT); // "
    randomSeed(analogRead(A4)); //Semente aleatoria baseada en A4
}
```

Comprobamos agora pulsando o botón de Reset, que xa non comenza sempre igual.





Tons.

Podemos usar a técnica PWM facer sons (tons), pero temos unha instrución adicada o son “**tone()**” que xera unha onda cadrada dunha frecuencia axustable con un ciclo de traballo do 50%, tamén temos unha instrución para parar “**noTone()**”.

Sintaxe:

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

```
/* Buzzer_previo, Exemplo de son co buzzer
*/
#define Buzzer 10 //Zumbador D10~
void setup() {
    pinMode (Buzzer, OUTPUT); //Definimos como saída
    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 o son no pin D10
}
void loop() { // non é preciso por nada, non precisamos repetir
}
```

💡 Analise:



Podemos ver no código que as instrucións de ton están na función `setup()`, isto se fai para que só execute os tons unha vez, na función `loop()` non temos instrucións xa que realmente non é preciso facer nada.

Isto pode escribirse usando unha chamada a unha nova función `void loop()` que non precisa de parámetros .

```
/* Buzzer
Son co Buzzer
*/
#define Buzzer 10 //Zumbador D10~

void setup() {
    pinMode (Buzzer, OUTPUT); //Definimos como saída
    son(); //chamamos a función son
}

void loop() {
    //nada non precisamos 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 o son no pin D10  
}
```

💡 Analise:

Na función **setup()** chamamos a **son()**, e unha vez rematada segue a **loop()**, onde se queda facendo nada “:-)”.

Por certo, te sonan esas notas?

Nos exemplos que trae Arduino temos un exemplo de melodía “[Ficheiro/Exemplos/02.Digital/toneMelody](#)”, só temos que cambiar o pin de saída de 8 polo 10 (liñas 37 e 44).

Nota: Para mostrar os números das liñas no IDE [Ficheiro] [Preferencias]

```
toneMelody      pitches.h  
31  // iterate over the notes of t  
32  for (int thisNote = 0; thisNot  
33  
34    // to calculate the note dur  
35    //e.g. quarter note = 1000 /  
36    int noteDuration = 1000 / nc  
37    tone(8, melody[thisNote], nc  
38  
39    // to distinguish the notes,  
40    // the note's duration + 30%  
41    int pauseBetweenNotes = note  
42    delay(pauseBetweenNotes);  
43    // stop the tone playing:  
44    noTone(8);  
45  }  
46 }
```



Pulsador_I.

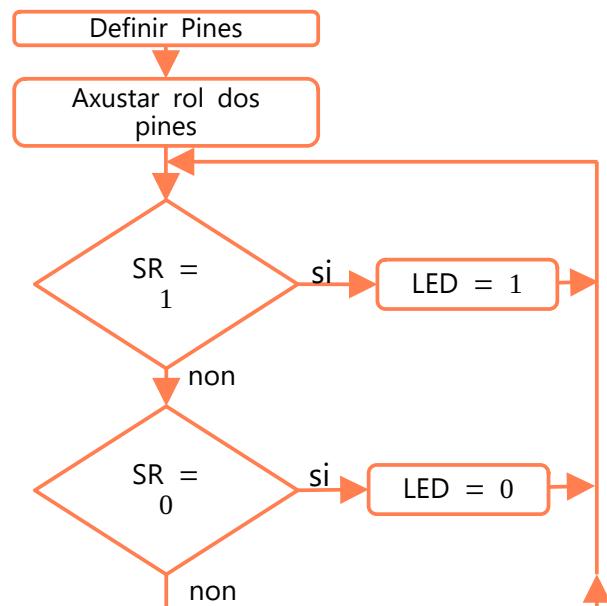
Programa que acende un led cando prememos o pulsador (neste exemplo sen variables). Lectura dixital 0/1

Para este exemplo usamos unha estrutura de control **If**:

É un estamento que se utiliza para probar si unha determinada condición se alcanzou, por exemplo comprobar si unha igualdade é certa, e executar unha serie de declaracíons (operacíons) que se escriben dentro de chaves {}. Si é falso (a condición non se cumple) e non executa as operacíons que están dentro das chaves.

```
/* Pulsador_I
Acender un led cando se prema o pulsador
sen variables
*/
#define SR 2 // Pin do pulsador
#define LED 11 // pin do led
void setup() {
    pinMode(LED, OUTPUT); //modo saída
    pinMode(SR, INPUT);   //modo entrada
}
void loop() {
    if (digitalRead (SR) == HIGH) { //comproba si
        SR é igual a "1"
        digitalWrite(LED, HIGH); //acende o LED
    }
    if (digitalRead (SR) == LOW) { //comproba si
        SR é igual a "0"
        digitalWrite(LED, LOW); //apaga o LED
    }
}
```

Analise:



Neste exemplo facemos dúas comprobacíons sobre dúas lecturas consecutivas da mesma entrada. Non é o máis aconsellable, pero podemos usar a seguinte estrutura.

if...else.

Vén ser un estrutura que se executa en resposta a idea:



Si se cumpre a condición fai isto, se non fai isto outro.

Por exemplo, deséxase probar unha entrada dixital, e facer unha cosa si a entrada é alta ou facer outra cousa si a entrada non é alta. Como fixemos no caso anterior imos acender o LED cando o pulsemos SR = 1, e apagar no caso contrario.

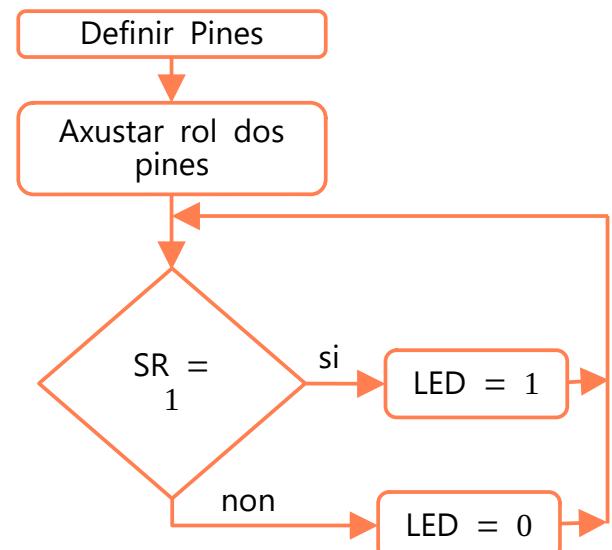
```
/* Pulsador_2
Acender un led cando se prema o pulsador
apagalo en caso contrario
*/
#define SR 2 // Pin do pulsador

#define LED 11 // pin do led

void setup() {
    pinMode(LED, OUTPUT); //modo pin como saída
    pinMode(SR, INPUT); //modo pin como saída
}

void loop() {
    //si o pulsador é "1" acende o led
    if (digitalRead (SR) == HIGH) {

        digitalWrite(LED, HIGH);
    }
    // no caso contrario apaga o led
    else {
        digitalWrite(LED, LOW);
    }
}
```





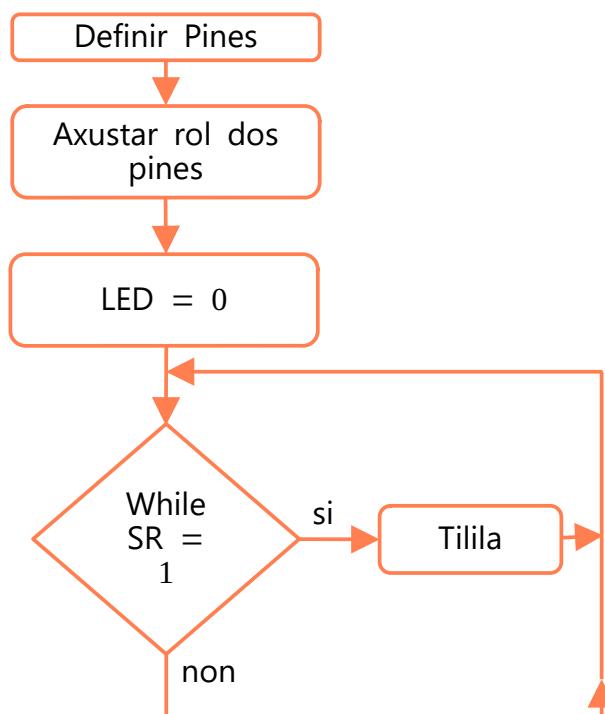
While

Imos facer titilar o LED cando se pulse SR, usando a estrutura While. Este bucle execútanse de forma continua, e infinitamente, sempre que condición entre o paréntese sexa verdadeira, cando a expresión dentro do paréntese () convértese en falsa deixase de executar.

```
/* Titila_II
titila cando se pulse SR usando a estrutura While
*/
#define L_Orn 12 //Led laranxa
#define SR 2 //Pulsador SR

void setup() {
    pinMode (SR, INPUT); // Definimos modo entrada
    pinMode (L_Orn, OUTPUT); // Definimos modo saída
    digitalWrite(L_Orn, LOW); //Apaga o LED
}

void loop() {
    // mentres o pulsador SR valga 1 facemos titilar
    while (digitalRead(SR)==1) {
        digitalWrite(L_Orn, HIGH); //Acende o LED
        delay(50); //Espera 50 misegundos
        digitalWrite(L_Orn, LOW); //Apaga o LED
        delay (100); //espera 100 misegundos
    }
}
```





Lectura analólica.

Imos comenzar a ler os sensores analóxicos.

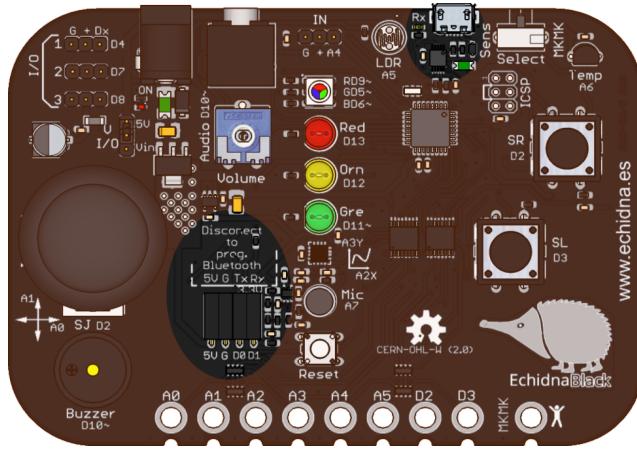
Para facer más cómoda a medida imos enviar os datos pola conexión USB e velos co ordenador.

EchidnaBlack conta con comunicación serie mediante conexión USB e zócolo para adaptador Bluetooth.

Lembra que para subir os sketchs é aconsellable non ter pinchado o adaptador Bluetooth, xa que comparten as mesmas conexións

Para usar a comunicación serie veremos as instrucións **Serial.begin()**;

Serial.print();, **Serial.println();**, temos máis instrucións relacionadas ca comunicación serie que iremos vendo en outros exemplos. Agora nos centraremos nas fundamentais para enviar datos e poder visualizalos no ordenador.



nas fundamentais para enviar datos e

Sintaxe:

Serial.begin(Velocidade); //configura e inicializa a canle serie

Velocidade en bits por segundo (Baudios),

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

Podemos usar outras velocidades para adaptar os terminais de comunicacóns, EchidnaBlack está testada de 300 ata 2 000 000 Baudios.

Tamen podemos axustar o resto dos parámetros de comunicación:

Serial.begin(Velocidade, configuración);

Configuración: SERIAL_ e Numero de datos por palabra, paridade par ou impar e bits de stop un ou dous, por defecto envía 8 bits, sen paridade e un bit de stop.

Serial.begin(Velocidade, SERIAL_8N1); é o mesmo que **Serial.begin(9600);**

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



Unha vez configurada a comunicación xa podemos enviar datos, estes poden ser numéricos ou cadeas de texto.

Sintaxe:

```
Serial.print(ValorSensor); // envía o ValorSensor
Serial.print("\t"); // envia una separación tabulada entre valores
// envía o ValorSensor e unha liña nova de separación
Serial.println(ValorSensor);
```

JoyStick_analogx.

```
/* Joystick_analogx
 * Lectura analólica dos eixos X,Y do Joystick
 */
#define Joy_X A0 // Eixo x conectado -> A0
#define Joy_Y A1 // Eixo x conectado -> A0

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

    // Non é preciso configurar os pinos como entrada,
    // pero é unha boa practica para lembrar o rol de cada un.
    pinMode (Joy_X, INPUT); // Definimos como entrada
    pinMode (Joy_Y, INPUT); // "

}

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



Analise:

No exemplo vemos que aparece unas variables “Valor_X e Valor_Y” precedidas de “int”. As variables son espazos de memoria onde imos almacenar datos que poden variar no tempo, temos que declarar como serán eses datos, neste exemplo int (enteiro) usa 16 bits (2 Bytes) os valores van de -32,767 a 32,767, podemos declarar a variable de forma local, só para a función na que a declaramos ou de forma global si a declaramos antes do `setup()`, podemos darle un valor inicial “`int Valor_X =0`”.

Podemos comprobar os datos transmitidos no monitor serie que ten integrado o IDE Arduino.

Lembra axustar os parámetros de comunicación do terminal como o que puxemos no programa `“Serial.begin(9600);”`.

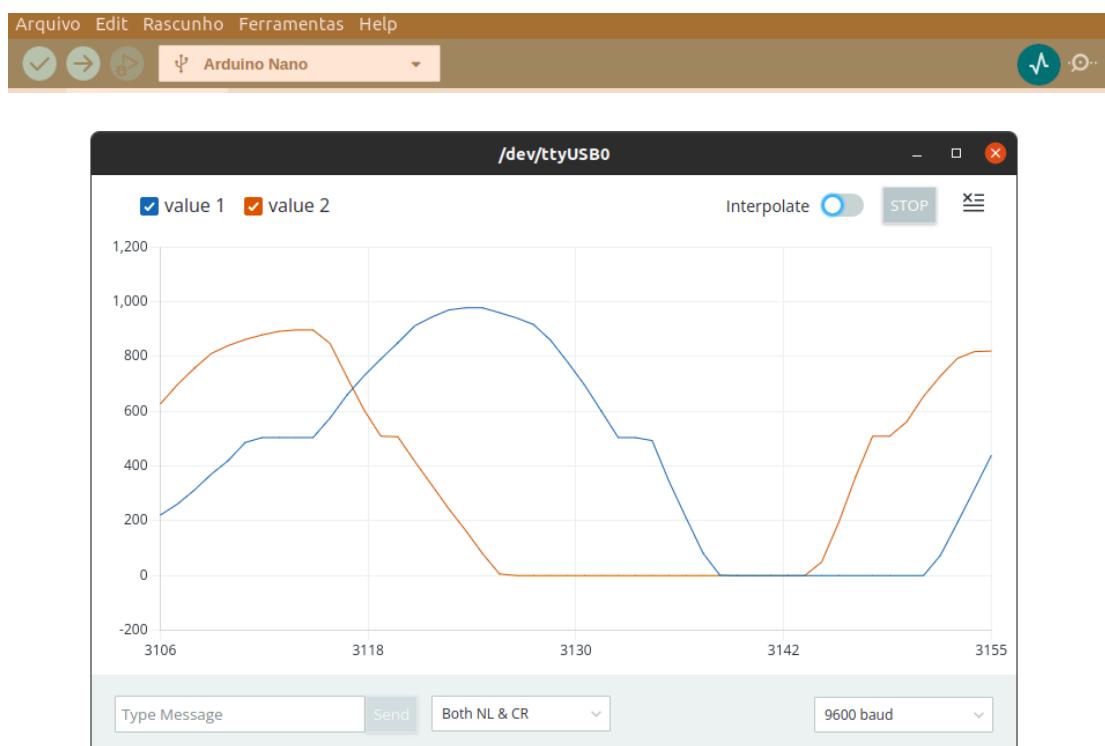
Vemos os datos numéricos separados polo tabulador.

Outra forma de ver os datos que nos ofrece o IDE é o plotter serie.

The screenshot shows the Arduino IDE interface with the file "Joystick_analog.ino" open. The code defines two integer variables for joystick axes and initializes the serial port at 9600 Bps. Below the code is the "Monitor Serial" window displaying a series of numerical values separated by tabs, representing the analog readings from the joystick axes. The window has a status bar indicating the baud rate is 9600 and the connection is to an Arduino Nano over USB.

```
/* Joystick_analog
 * Lectura analóxica dos eixos X,Y do Joystick
 */
#define Joy_X A0 // Eixo x conectado -> A0
#define Joy_Y A1 // Eixo y conectado -> A0
void setup() {
    // Inicia comunicación serie a 9600 Bps
    Serial.begin(9600);
}
// Non é necesario configurar no minal con entrada
```

| Timestamp | Value 1 (Joy_X) | Value 2 (Joy_Y) |
|-----------|-----------------|-----------------|
| 3106 | 200 | 600 |
| 3118 | 480 | 500 |
| 3128 | 950 | 900 |
| 3130 | 500 | 0 |
| 3132 | 480 | 0 |
| 3142 | 0 | 50 |
| 3155 | 450 | 800 |





Tipos de datos:

(ollo non están todos) (Var é nome da variable, val e o valor que damos a variable)

- `bool` var = val; //Usa un(1) bit, binario 0 ou 1, falso (false)ou verdadeiro (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; //Enteiro usa 16 bits (2 Bytes) os valores van de -32767 a 32767
- `word` var = val; //Ocupa 16 bits sen signo de 0 a 65535
- `uint16_t` var = val; //Almacena 16 bits, 0 a 65535
- `unsigned int` var = val; //Enteiro 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; //Longo sen signo, 32 bits(4 bytes), 0 a 4,294,967,295
- `char` var = val; // Almacena un carácter, os caracteres literais // escribense entre comiñas simples 'A' // para varios caracteres van entre comiñas dobles "ABC"

**Proposta A, B:**

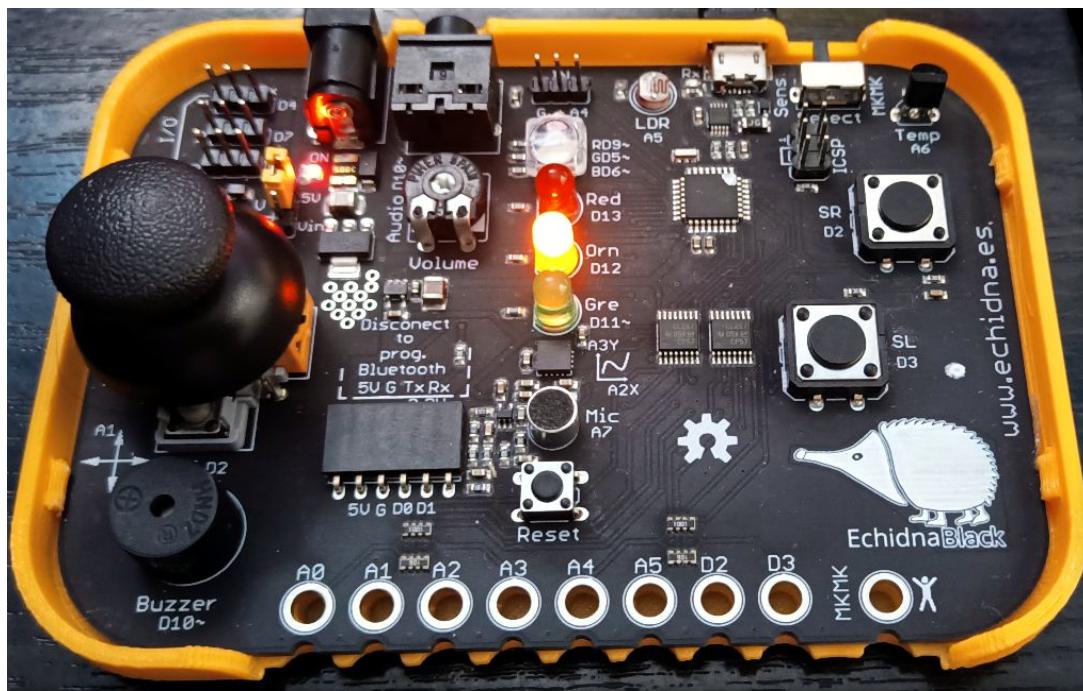
- A) Escribe un programa que active un led cando superamos un valor unha entrada analólica, por exemplo Acende o led Vermello cando A1 >= 550.

...

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

- B) escribe un programa dunha baliza, que funcione segundo a seguinte lóxica:

| Condición/LEDes | Vermello | Laranxa | Verde |
|-----------------|----------|---------|-------|
| A1 >= 600 | 1 | 0 | 0 |
| 400 < A1 < 600 | 0 | 1 | 0 |
| A1 <= 400 | 0 | 0 | 1 |





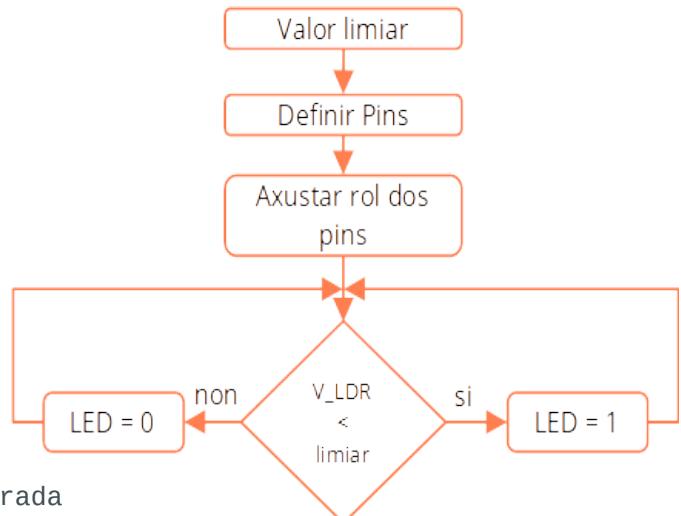
LDR.

Neste exemplo imos facer un interruptor crepuscular, que acenderá un led cando a luz baixe de un valor dado.

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

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

void loop() {
    // Leemos aa entrada
    int Valor_LDR = analogRead(LDR);
    // comproba si o valor está por baixo
    if ( Valor_LDR < limiar) {
        digitalWrite (L_Orn, HIGH); // Acende o LED
    }
    else {
        digitalWrite (L_Orn, LOW); // Apaga o LED
    }
    // un pequeno retardo para estabilizar as medidas.
    delay(1);
}
```



💡 Analise:

Unha vez que temos o valor da LDR, o comparamos co valor de activación “limiar”, se o valor da luz é inferior a este valor acendemos o LED, no caso contrario permanece apagado. Este sketch fai a función dun Interruptor crepuscular.

X É aconsellable usar valores de histérese veremos un exemplo no programa Temperatura



Theremín LDR.

Imos manexar o zoador manipulando a luz que incide na LDR emulando o famoso “Theremin” ;-).

```
/*TheremLDR
Un pequeno programa, ca LDR manexamos o zoador e un par de LEDes
*/
//Entradas / saídas
#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); //Axuste do modo
pinMode (Buz ,OUTPUT);
pinMode (RGB_R ,OUTPUT);
}
void loop() {
Valor_LDR = analogRead(LDR); //leemos a LDR
//escala o Valor da LDR a frecuencia
Frecuencia = map(Valor_LDR, 0, 1023, 400, 2500);
BrilloR = map(Valor_LDR, 0, 1023, 0, 128); //escala Valor_LDR o brillo do LED
BrilloG = map(Valor_LDR, 0, 1023,16, 64); //escala para colorear
//si se escurece apaga o zoador
if (Valor_LDR<50){ Frecuencia= 0;}
tone(Buz, Frecuencia, 10); //emite a frecuencia dependente da LDR
analogWrite(RGB_R, BrilloR); //luz Vermella
analogWrite(RGB_G, BrilloG); //luz Verde
delay(1); //unha espera para estabilizar a conversión AD
}
```



Temperatura.

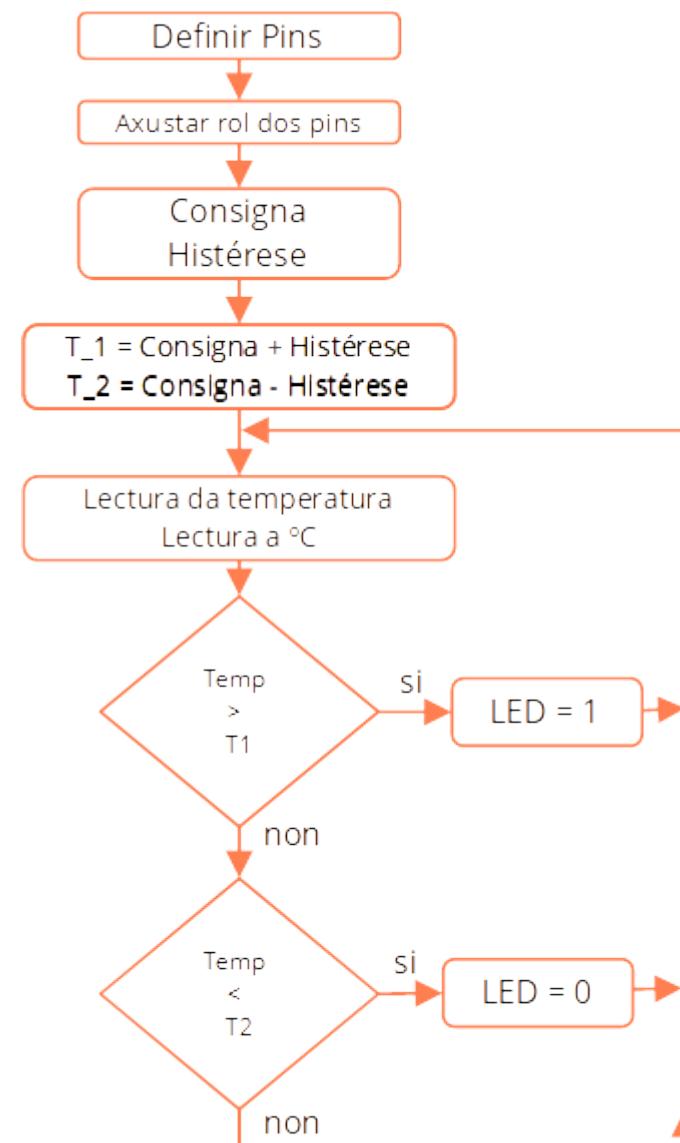
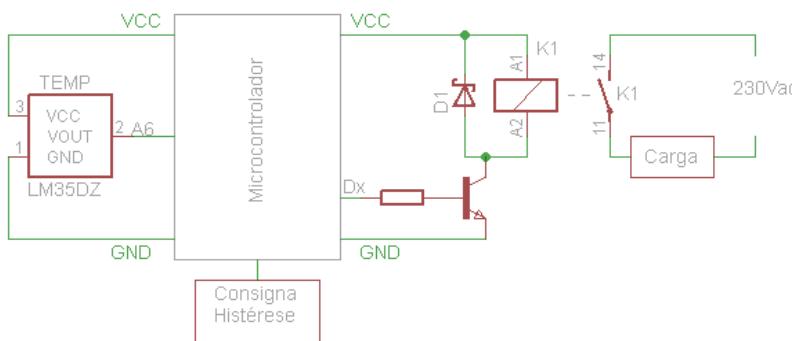
Usando o sensor de temperatura LM35 (LM35_Pin) acenderemos un led cando a temperatura supere un valor e enviaremos o valor vía serie o ordenador.



Teremos en conta unha pequena histéreses para non activar repetidamente a saída cando a temperatura este cerca do punto de conmutación, o que faría que se poida queimar o posible relé ou contactor asociado. (que activara a saída)

Así temos dúas temperaturas T_1 temperatura de activación e T_2 temperatura de desactivación a diferencia entre elas pode chamarse histérese de temperatura.

O esquema representa un circuíto de control de unha carga (refrigerador...) mandada mediante o noso programa que manda a un contactor que se encarga de manexar a potencia que precisa a carga.



*/ Termóstato LM35

Comparación da temperatura con unha dada como Consigna

para acender un led (activar unha saída)

*/

```
#define L_Gre 11 // pin do LED verde -> D11
#define LM35 A6 // Pin do LM35 -> A6
```

```
int Consigna = 27; //Temperatura elixida °C
//diferencia en grados sobre activación e
desactivación
```

```
int Diferencia = 2;
```



```
int Histerese = (Diferencia/2);
int T_1 = Consigna + Histerese; //temperatura de activación
int T_2 = Consigna - Histerese; //temperatura de desactivación
int temperatura; //temperatura medida
char *Estado = "Desactivado"; //cadea de texto do estado

void setup() {
analogReference(INTERNAL); //referencia analólica a 1.1V
Serial.begin(9600); //config. porto serie
pinMode (LM35, INPUT); //pin LM35 como entrada
pinMode (L_Gre, OUTPUT); //saída para LED
digitalWrite (L_Gre, LOW); //apaga o LED, estado inicial
}

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

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

if (temperatura < T_2) { //si baixa do valor
    digitalWrite (L_Gre, LOW); //apaga o LED
    Estado = "Desactivado";
}

// envia as mensaxes 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("Histérese: ");
```



```

Serial.print(Histerese);
Serial.print("°C");
Serial.print("\t");
Serial.print("Estado: ");
Serial.println(Estado);

delay(1000); //tempo de espera de 1s
}

```

Analise:

- **Consigna**, onde depositamos o valor da temperatura, o redor do que queremos establecer a comutación.
- **Histérese**, ten o valor que se incrementa e decrementa o valor de consigna para establecer os dous puntos de comutación.
 - Diferencia = 1; Histérese = (Diferencia/2);
 - $T_1 = \text{Consigna} + \text{Histérese};$
 - $T_2 = \text{Consigna} - \text{Histérese};$
- ***Estado**, é unha variable que vai ter os caracteres para enviar vía serie o estado de activado ou desactivado.
- Como explicamos antes ca instrución `analogReference(INTERNAL)` axustamos a referencia de conversión analoxía a dixital a 1,1V, para obter unha maior precisión.
- O calculo da temperatura en °C é “temperatura = (lectura * 1.1 * 100) / 1024;” onde temos en conta a referencia de tensión de 1,1V e a resolución de 10 bits da conversión analoxía a dixital $2^{10}=1024$.
- Realiza dúas comparacións para establecer o estado.
 - Unha co valor $T_1 \rightarrow$ Activar e outra $T_2 \rightarrow$ Desactivar
- Xa só queda enviar os datos de funcionamento vía serie.
 - `Serial.print("Temp Actual: ");`
 - `Serial.print(temperatura);`



The screenshot shows the Arduino Serial Monitor window. The title bar says "Saída Monitor Serial". The main area displays a series of temperature readings from an Arduino Nano connected via USB. The data is presented in pairs of rows:

| Temp Actual: [value] | Consigna: [value] | Histérese: [value] | Estado: [value] |
|----------------------|-------------------|--------------------|-----------------|
| 27°C | 27°C | 2°C | Desactivado |
| 27°C | 27°C | 2°C | Desactivado |
| 27°C | 27°C | 2°C | Desactivado |
| 27°C | 27°C | 2°C | Desactivado |
| 27°C | 27°C | 2°C | Desactivado |
| 27°C | 27°C | 2°C | Activado |
| 32°C | 27°C | 2°C | Activado |
| 32°C | 27°C | 2°C | Activado |
| 31°C | 27°C | 2°C | Activado |
| 30°C | 27°C | 2°C | Activado |
| 30°C | 27°C | 2°C | Activado |
| 29°C | 27°C | 2°C | Activado |
| 30°C | 27°C | 2°C | Activado |
| 30°C | 27°C | 2°C | Activado |
| 29°C | 27°C | 2°C | Activado |

At the bottom of the monitor window, it says "Ln 1, Col 1 UTF-8" and "Arduino Nano em /dev/ttyUSB0".

Vúmetro.

Imos simular un vúmetro ou medidor de volume, usando tódolos LEDes como indicador e o micrófono como captador de son.

```
/* Vumetro
Usa os LEDes L_Gre, L_Orn, L_Red, RGB_R, RGB_G e RGB_B
como indicadores do nivel.

*/
#define RGB_B 6 //Led RGB azul
#define RGB_G 5 //Led RGB verde
#define RGB_R 9 //Led RGB vermello
#define L_Red 13 //Led vermello
#define L_Orn 12 //Led laranxa
#define L_Gre 11 //Led verde
#define Mic A7 //Micrófono
//LEDes nunha matriz para simplificar o manexo
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 os LEDes como saídas
    for (int i = 0; i < 7; i++) { //percorremos a matriz
        pinMode(ledPins[i], OUTPUT); //e definimos cada led como saída
    }
    pinMode (Mic, INPUT);
    //Cambiamos a referencia analólica a 1.1V
    analogReference(INTERNAL);
```

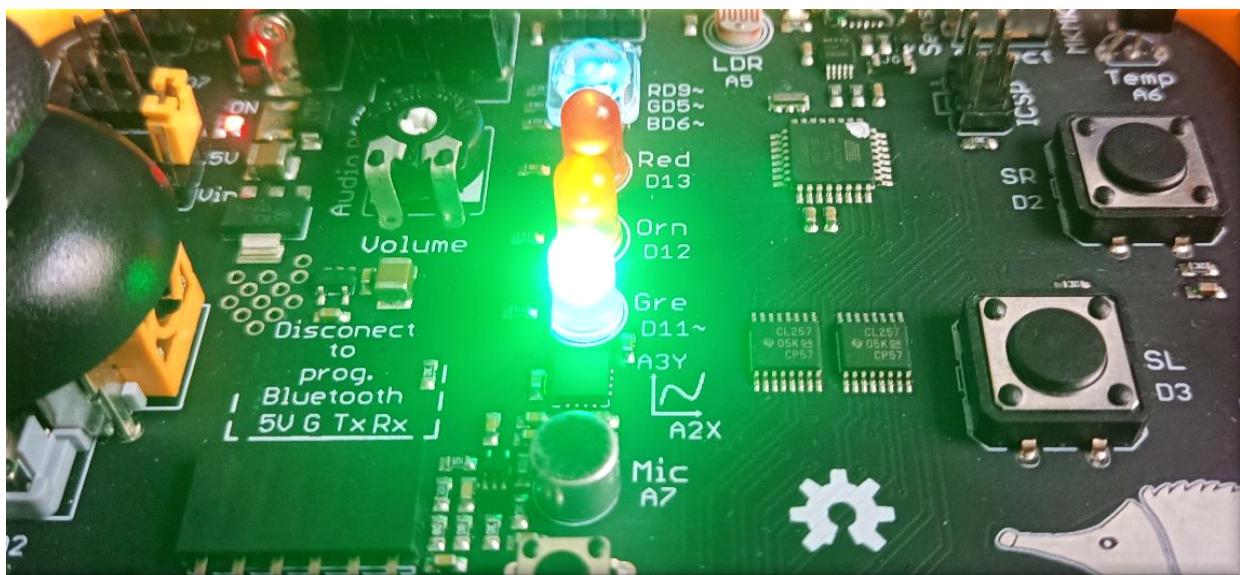


```

}

void loop()
{
    //Captura o valor analóxico do micrófono
    int sinal = analogRead(Mic);
    // Comproba que non nos pasamos de V_Max
    if (sinal > V_Max) {
        sinal = V_Max;
    }
    // converte os valores do MIC a un nº de 0 a 7 (LEDes)
    int LED_maior = map(sinal, 0, V_Max, 0, 7);
    //Acende os LEDes ata o valor máis alto
    for (int i = 0; i < LED_maior; i++){
        digitalWrite(ledPins[i], HIGH);
    }
    //Apaga ata o LED más baixo
    for (int i = 7; i > LED_maior; i--) {
        digitalWrite(ledPins[i], LOW);
    }
    delay(1);
}

```





💡 Analise:

- Usamos unha matriz onde gardamos os LEDes en orde, para poder percorrer as saídas en orde.
- As matrices podemos definilas de varias formas:
 - int ledPins[7] ; os valores sen definir pero si reserva o espazo para sete datos enteros.
 - int ledPins[] = {0, 1, 2, 3, 4, 5, 6} ; Non dicimos a cantidad de datos, é o compilador quen conta o numero de datos e reserva o espazo cos datos entre as chaves{} separados por comas.
 - int ledPins[7] = {0, 1, 2, 3, 4, 5, 6} ; Aquí decidimos cuntos datos ten a matriz.
- No principio do programa definimos a matriz dos LEDes “int ledPins[] = {255, L_Gre, L_Orn, L_Red, RGB_R, RGB_G, RGB_B} ;” comenzando con 255, é un espazo de memoria non usado, no programa ten a finalidade de ser como unha saída fantasma e que non se vexa ningún Led acendido cando non teñamos sinal de entrada “analogWrite (ledPins[0], 16);”
- Percorrer unha matriz unidimensional usando un bucle “for” ou con calquera outro método é doadoo.
 - No setup() usamos for (int i = 0; i < 7; i++) { pinMode(ledPins[i], OUTPUT); }, para configurar os pines conectados os LEDes como saídas. Unha vez que xa temos o valor do sinal de son o escalamos o nº de LEDes disponíveis (0-7), usando a instrución map:
`map(Valor, dende baixo, dende alto, a baixo, a alto);`
Valor e a variable de entrada que ten os datos a escalar, “dende baixo” ou “dende alto” son os valores límites que queremos como entrada, “a baixo” ou “a alto” que son os valores que queremos como saída.
 - A función map() non devolta números fraccionarios, inda que o calculo o indique. Pode usarse para inverter valores x = (y, 1, 50, 50, 1)
- Acendemos os LEDes con for (int i = 0; i < LED_maior; i++) { digitalWrite(ledPins[i], HIGH); } dende o 0 ata o LED maior,
- Apagamos o resto (dende o LED maior ata o Led 6 con for (int i = 7; i > LED_maior; i--) { digitalWrite(ledPins[i], LOW);}



Nivel eixo Y.

Simulamos un nivel ca axuda da saída Y do acelerómetro cando está a nivel o LED Laranxa está iluminado Acenderanse os LEDes Vermello ou verde dependendo a inclinación

```
/* Nivel_Y
Nivel no eixo Y, indicadores LEDes
*/
#define Red 13 //Led vermello
#define Orn 12 //Led laranxa
#define Gre 11 //Led laranxa
#define Ace_Y A3 //Entrada acelerómetro Y
word Valor_Y; //Variables de lectura do acelerómetro
//Estos valores poden cambiar dependendo de cada EchidnaBlack
word Repouso_Y = 359; //Valor de repouso, A nivel

void setup() {
    Serial.begin(9600);
    pinMode(Red, OUTPUT); //modo saída para RGB vermello
    pinMode(Orn, OUTPUT); //modo saída para LED laranxa
    pinMode(Gre, OUTPUT); //modo saída para LED verde
    //Definimos entradas ;-
    pinMode(Ace_Y, INPUT); //acelerómetro Y
}

void loop() {
    Valor_Y = analogRead(Ace_Y);
    Serie();
    //si o valor Y supera o repouso + 2
    if (Valor_Y >(Repouso_Y+2)) {
        digitalWrite (Red, 1); //acendemos o LED vermello
        digitalWrite (Orn, 0); //apagamos o resto
        digitalWrite (Gre, 0);
    }
    //si o valor Y esta a nivel
    if (Valor_Y <(Repouso_Y+2) and Valor_Y >(Repouso_Y-2)) {
        digitalWrite (Red, 0);
        digitalWrite (Orn, 1); //acendemos o LED Laranxa
    }
}
```



```
digitalWrite (Gre, 0); //apagamos o resto
}
if (Valor_Y <(Repouso_Y-2)) {
digitalWrite (Red, 0);
digitalWrite (Orn, 0);
analogWrite (Gre, 32); //acendemos o LED verde apagamos o resto
} //escribimos analoxicamente 32, por que este led
} //iluminase moito, asi so acende o 1/4 do total

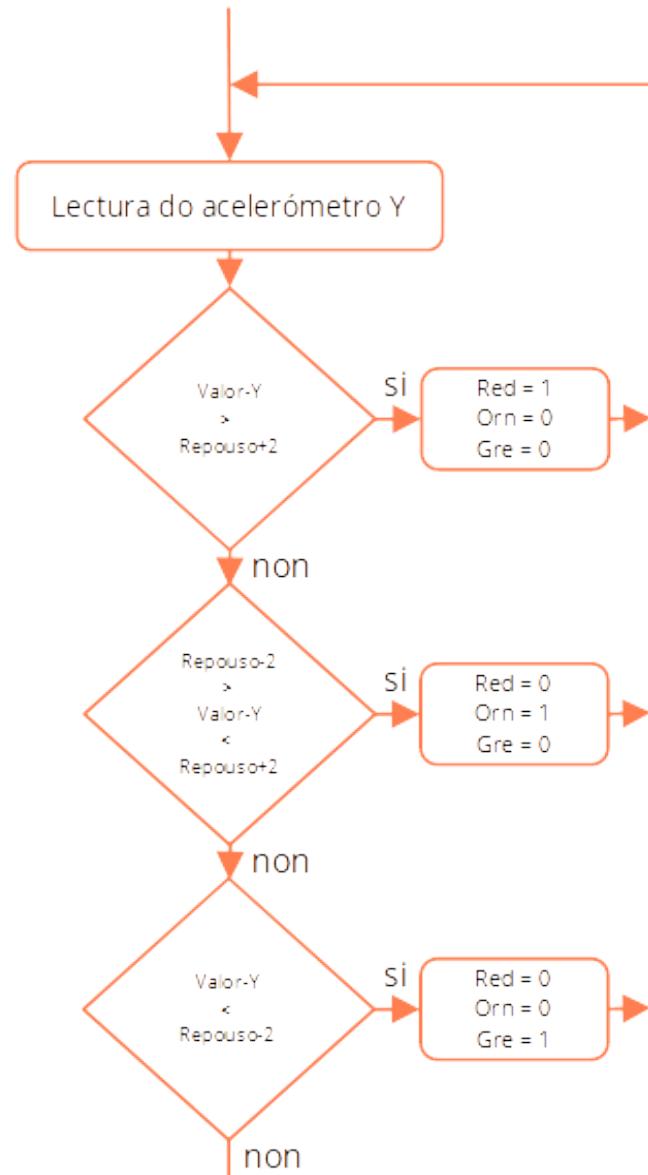
void Serie() {
Serial.print("Valor_Y:");
//envía os datos vía serie
Serial.print(Valor_Y);
Serial.println("");
}
```

Analise:

Neste exemplo, unha vez feita a lectura do valor Y do acelerómetro, realizamos tres comparacións: Si supera-mos o valor de repouso +2, activamos o LED vermello, si o valor está entre o valor de repouso -2 e o valor de repouso +2, acendemos o LED laranxa e si o valor e menor que o de repouso -2, acendemos o LED verde.

Unha ampliación que podemos realizar sería memorizar o valor de repouso premendo o pulsador Sr ou SL

Isto podemos usalo para facer un xogo de colher con moito coidado a EchidnaBlack sen inclinala. :-)





Xogo “Colle a EchidnaBlack sen activar a Alarma.”

```
/* Vibracións
```

Alarma movemento usamos o acelerómetro para activara alarma zoador e led RGB vermello, cando tentamos mover Echidna Cando titila o LED laranxa nos invita a pulsar SR para memorizar a a posición inicial, e comenzar o xogo. más info. vía serie */

```
int limiar_X = 5; //Limites de alarma
int limiar_Y = 5;
```

```
#define RGB_R 9 //Led RGB vermello
#define Orn 12 //Led laranxa
#define Gre 11 //Led laranxa
#define Buz 10 //Zoador, saída son
#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 do acelerómetro
word Valor_Y;
word mostras = 100; //numero de lecturas consecutivas
word media_X = 0;
word media_Y = 0;
```

```
void setup() {
    Serial.begin(9600);
    pinMode(RGB_R, OUTPUT); //modo saída para RGB vermello
    pinMode(Orn, OUTPUT); //modo saída para LED laranxa
    pinMode(Gre, OUTPUT); //modo saída para LED verde
    pinMode(Buz, OUTPUT); //modo saída para o zoador

    //Definimos como entradas ;-
    pinMode(Ace_X, INPUT); //acelerómetro X
    pinMode(Ace_Y, INPUT); // " Y
    pinMode(SR, INPUT); //pulsador SR
```



```
Serial.println("");
Serial.print("Pulsa SR para memorizar a 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(); // Chama a función Serie(), para enviar os valores.

    //Si excedemos os límites inferior ou superior activamos a alarma
    if (Valor_X > (media_X + limiar_X) or Valor_X < (media_X - limiar_X)) {
        alarma();
    }
    if (Valor_Y > (media_Y + limiar_Y) or Valor_Y < (media_Y - limiar_Y)) {
        alarma();
    }
    //Se non temos alarma apaga o LED e zoador
    digitalWrite (RGB_R, LOW);
    noTone(Buz);
}

// función alarma
void alarma() {
    Serial.println("Alarma"); //Envía a palabra "Alarma"
    digitalWrite (RGB_R, HIGH); //activa o LED vermello
    tone(Buz, 2500); //Emite unha frecuencia de 2500Hz
    delay(100); //pausa de 100 mSeg
}

void memoriza_XY() {
    //medidas repetidas para facer unha media
    for (int i = 0; i < mostras; i++) {
        Valor_X = analogRead(Ace_X);
        media_X = media_X + Valor_X;
```



```
delay (1); //atraso para a conversión analóxica dixital
Valor_Y = analogRead(Ace_Y);
media_Y = media_Y + Valor_Y;
delay (1); //atraso para a conversión analóxica dixital
}
//calcula a media aritmética
media_Y = media_Y / mostras;
media_X = media_X / mostras;
//Envia os 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); //Acende o LED verde
delay(2000); //espera 2 segundos
analogWrite(Gre, 32); //Acende o LED verde con más intensidade
}

//envía os valores memorizados e os actuais
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("");
}
```



Analise:

Neste exemplo a máis das estruturas que xa estivemos vendo nos exemplos anteriores, volvemos a usar a estrutura de control `while ()`. Estes bucles execútanse de forma continua, e infinitamente, ata que a expresión dentro do paréntese `()` convértese en falsa.

De esta forma esperamos a que se prema SR “`while (digitalRead (SR) == 0) { digitalWrite (Orn, HIGH); ... }`” cando a comprobación “`(SR) == 0`” non se cumpla saímos do bucle `while` e executamos a seguinte instrución que fai unha chamada a función `memoriza_XY()` onde nun bucle `for` realizamos varias medidas das entradas `Ace_X` e `Ace_Y`. Para facer unha media aritmética que almacenamos nas variables `media_X` e `media_Y`, que nos serven de valoras para comparar cas medidas actuais, usando un marxe dado por `limiar_X` e `limiar_Y`, danos o xogo no que non se dispara a alarma.

Se queres poñelo xogo máis difícil, baixa os valores `limiar_X` e `limiar_Y` ou aumenta o numero de mostras da media aritmética.

```
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
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 4:2



Piano MkMk.

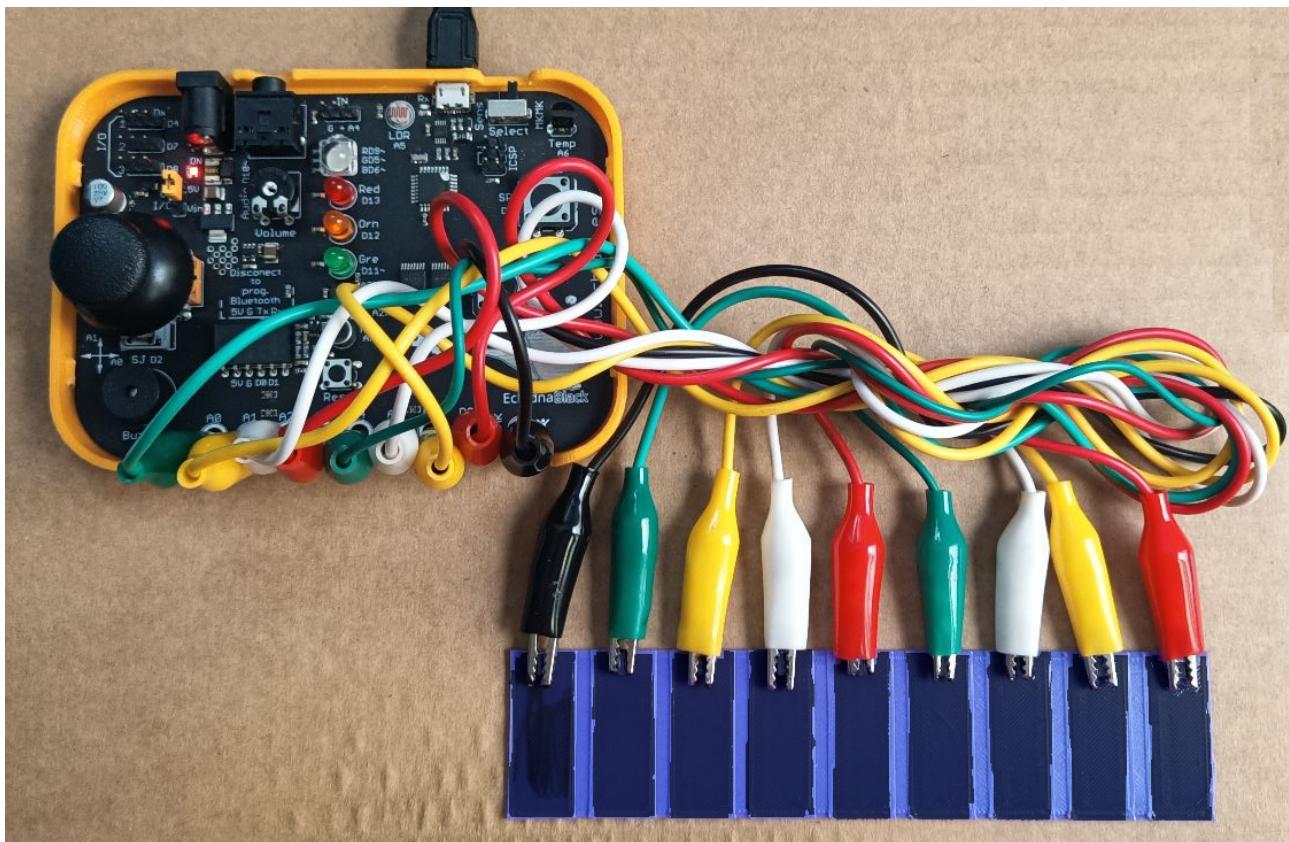
Faremos soar o zoador usando as entradas MkMk, como un Mini-Piano ;-)

Agora imos xogar un pouco no modo MKMK. Podemos facer case todo o anterior, cambiando os pulsadores por elementos que conduzan ago a corrente eléctrica como se comentou no "[Modo MkMk](#)".

Precisamos tantos elementos conductivos como teclas queiramos, con un máximo de 8,

Podemos usar teclas pintadas con lapis sobre un papel, canto más ancho e grosso as pintemos más conducirán a corrente e serán más doidas de detectar, tamén gominolas a maioría conducen un pouco a corrente.

Conectaremos as "teclas" mediante cables, as pinzas de crocodilo facilitan moito o conexiónado.



Impreso en PLA, teclas pintadas con grafito en spray



```
/* Piano
 * Seleccionar o modo MkMk
 * Úsanse as entradas A0..A5, D2 e D3 como entradas
 * D1 será a saída de son
 * Sempre que pechamos o circuíto entre o común e a tecla conectada a entrada
 * Emítese un ton con un tempo establecido
 */
#define Tempo 500 //duración do ton en milisegundos
#define Buz 10 //Zoador, saída son

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 saída
}

void loop() {
    if (digitalRead (A0)==1){ //leemos a entrada A0
        tone(10,262,Tempo); //se é = 1 emite un ton
        delay(Tempo); //pausa
    }
    if (digitalRead (A1)==1){ // " A1
        tone(Buz,294,Tempo);
        delay(Tempo);
    }
    if (digitalRead(A2)==1){
        tone(Buz,330,Tempo);
        delay(Tempo);
    }
    if (digitalRead (A3)==1){
        tone(Buz,349,Tempo);
        delay(Tempo);
    }
    if (digitalRead (A4)==1){
```



```

tone(Buz, 392, Tempo);
delay(Tempo);

}

if (digitalRead(A5)==1){
    tone(Buz, 440, Tempo);
    delay(Tempo);
}

if (digitalRead (2)==1){
    tone(Buz, 494, Tempo);
    delay(Tempo);
}

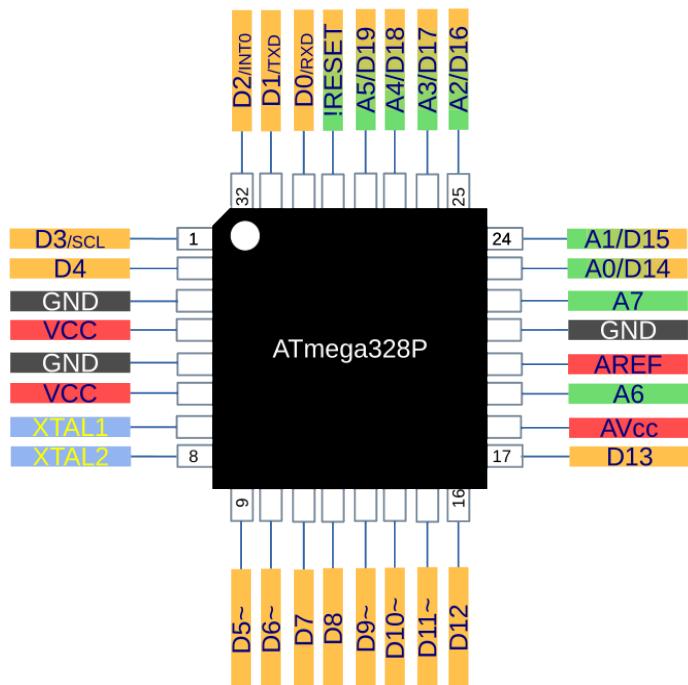
if (digitalRead (3)==1){
    tone(Buz, 523, Tempo);
    delay(Tempo);
}

}

```

💡 Analise:

Estamos facendo unha lectura dixital das entradas analóxicas, case que todos os pinos do Atmega328P pode lerse dixitalmente, salvo A6 e A7 que son entradas analóxicas exclusivamente.





Podemos facer o mesmo programa se ter que escribir tanta liñas, usando uns bucles **for** e unhas matrices.

Piano_2.

```
/* Piano_2
 * Seleccionar o modo MkMk
 * Úsanse as entradas A0..A5, D2 e D3 como entradas
 * D1 será a saída de son
 * Sempre que pechamos o circuito entre o común e a tecla conectada a entrada
 * Emítense un ton con un tempo establecido
 */
int MkMk[] = {A0, A1, A2, A3, A4, A5, 2, 3} ; //Entradas MkMk
int Ton[] = {262, 294, 330, 392, 440, 494, 523 };//frecuencias dos tons

#define Tempo 500          //duración do ton en misegundos
#define Buz 10      //Zoador, saída son

void setup() {
    for (int i = 0; i < 8; i++) {
        pinMode(MkMk[i], INPUT); //Define como entradas as MkMk
    }
    pinMode(Buz, OUTPUT); //modo saída
}

void loop() {

    for (int i = 0; i < 8; i++)           //Percorremos a matriz

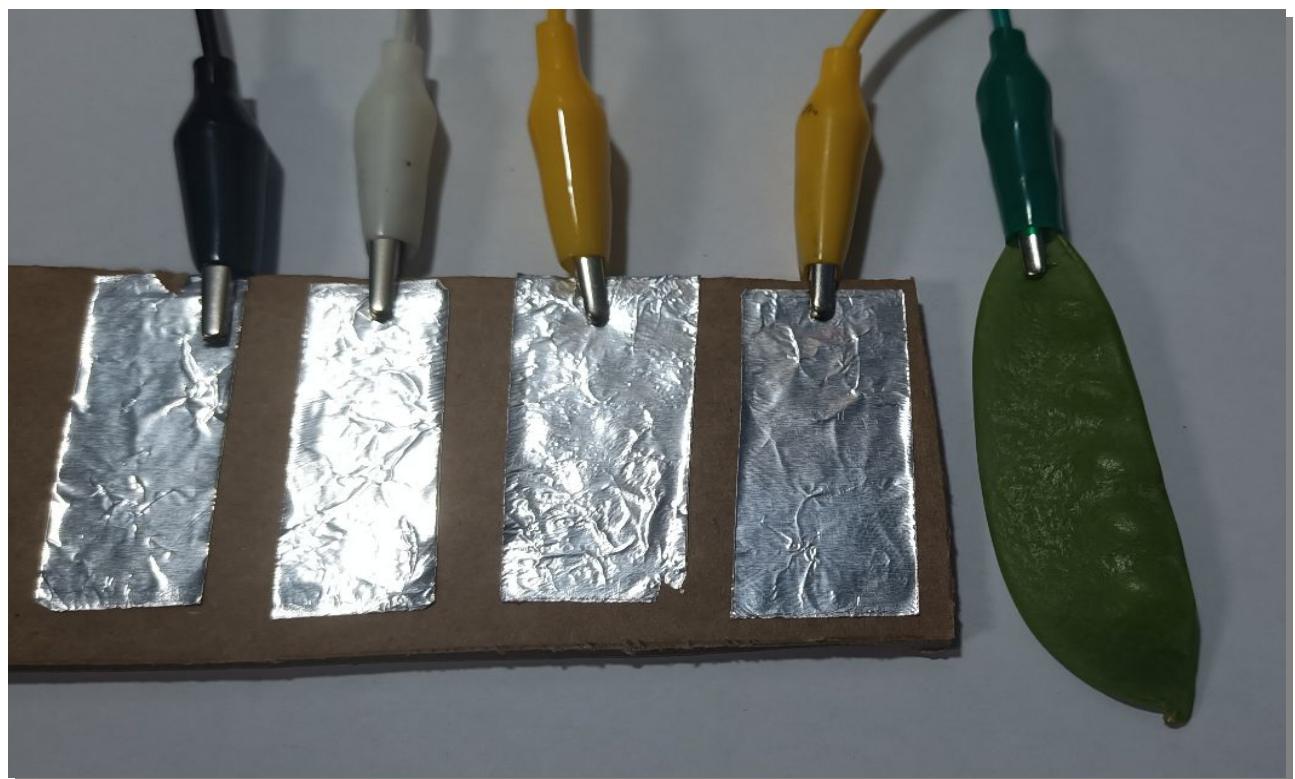
        if (digitalRead (MkMk[i]) == 1) {   //Leemos a entrada MkMk[]
            tone(Buz, Ton[i], Tempo);      //se é = 1 emite un ton
            delay(Tempo);                 //pausa
        }
}
```



X Se precisamos detectar algo que conduza moi pouco podemos facer unha lectura analóxica e comprobar o nivel mínimo de disparo, e ter distintos limiares.

Exemplo:

```
if (analogRead (MkMk[3]) > limiar {      //Leemos a entrada A3
    tone(Buz, Ton[i], Tempo);      //se é = 1 emite un ton
    delay(Tempo);                //pausa
}
```





Acender LEDes dende o ordenador.

Ata o momento vimos como enviar datos dende a placa o ordenador, pero o sentido contrario permítenos controlar cousas na EchidnaBlack mediante a conexión serie dende o ordenata.

```
/* Led_Via_Serie
   Acende /apaga un LED mediante a consola serie
   1 = acende, 2 = apaga
*/
#define RGB_B 6 // Pin do LED azul
char Rec_Ser; // Variable de caracteres da recepción serie

void setup() {
    pinMode(RGB_B, OUTPUT); // Pin LED como saída
    Serial.begin(115200);
    Serial.println("Escribe un '1' para acender ou '2' para apagar" );
    digitalWrite(RGB_G, LOW);
}

void loop() {
    if (Serial.available()) { // Comproba que temos conexión establecida
        Rec_Ser = Serial.read(); // Pasa o dato recibido a variable Rec_Ser
        Serial.println(Rec_Ser); // Reenvía o dato recibido

        if (Rec_Ser == '1') { // Si o carácter recibido é '1' código ASCII 49
            digitalWrite(RGB_B, HIGH); // Acende o LED
        }
        else if (Rec_Ser == '2') { // Si o carácter recibido é '2' ASCII 50
            digitalWrite(RGB_0, LOW); // Apaga o LED
        }
    }
}
```

💡 Analise:

Estamos comprobando si o carácter recibido é igual a “1” que se corresponde co código ASCII 49, si queremos ver os códigos ASCII que tecleamos só temos que cambiar “`char Rec_Ser;`” por “`int Rec_Ser;`”



Axusta a iluminación LED dende o ordenata.

Agora queremos controlar a intensidade luminosa dun led teremos que converter os caracteres enviados a valor numérico usando “`Serial.parseInt()`” que pasa os díxitos recibidos a dato enteiro omitindo os caracteres iniciais que non sexan “`_`”

```
/* Dim_Led_Via_Serie
Axusta % luminosidade LED mediante a consola serie
0 = apagado, 100 = máximo
*/
#define RGB_G 5 // Pin do LED verde
int Rec_Ser; // Variable de caracteres da recepción serie
void setup() {
    pinMode(RGB_G, OUTPUT); // Pin LED como saída
    Serial.begin(115200);
    Serial.println("Escribe un número entre 0 e 100" );
    digitalWrite(RGB_G, LOW);
}
void loop() {
    if (Serial.available()) { // Comproba que temos conexión establecida
        Rec_Ser = Serial.parseInt(); // Pasa o dato recibido a variable Rec_Ser

        Serial.println(Rec_Ser); // Reenvia o dato recibido
        Rec_Ser = map(Rec_Ser, 0, 100, 0, 255); //escala os valores de entrada
        analogWrite (RGB_G, Rec_Ser);
    }
}
```

💡 Analise:

`Rec_Ser` recibe o dato xa convertido en numero “`int`” que podemos usar para controlar a intensidade luminosa do LED.

Usamos ”`map`” para pasar a % (0-100) os valores PWM (0-255) para o LED.



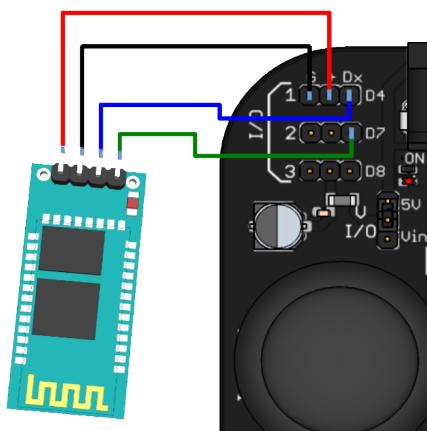
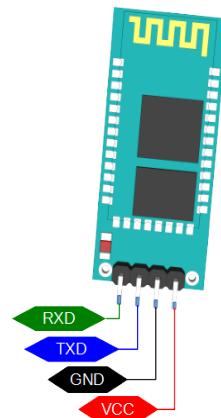


Acende LEDes dende o SmartPhone

Usando un adaptador Bluetooth [HC05](#) ou [HC06](#) e unha APP podemos usar os dous programas anteriores sen facer cambios para controlar o LED dende o SmartPhone.

Adaptador HC-06 ten catro conexións “RXD” recepción de datos, “TXD” transmisión de datos, “GND” masa ou negativo de alimentación e “VCC” Positivo de alimentación.

Temos que programar o adaptador Bluetooth para axustalo as nosas necesidades. Montaremos unha nova canle serie en dous dos pinos de libre disposición, usando a librería “SoftwareSerial” .



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

```
/* SoftwareSerial
 * Abre unha canle serie nos pinos especificados
 * Usamos dous dos de libre disposición
 * D4 como RXD,
 * D7 como TXD
 */
#include <SoftwareSerial.h> //Librería incluída no IDE
#include BaudRate 9600 // Velocidade de comunicacóns
SoftwareSerial SoftSerial(4, 7); // Asina os pinos 4 RXD, 7 TXD

void setup() {
    Serial.begin(BaudRate); // Inicializa o porto de comunicacóns
    Serial.println("Canle serie funcionando"); // Mensaxe de cortesía
    SoftSerial.begin(BaudRate); // Inicializa o porto de comunicacóns software

}
void loop() {
    if (SoftSerial.available()) { // Si ten datos SoftSer.os pasa a canle normal
        Serial.write(SoftSerial.read());
    }
}
```



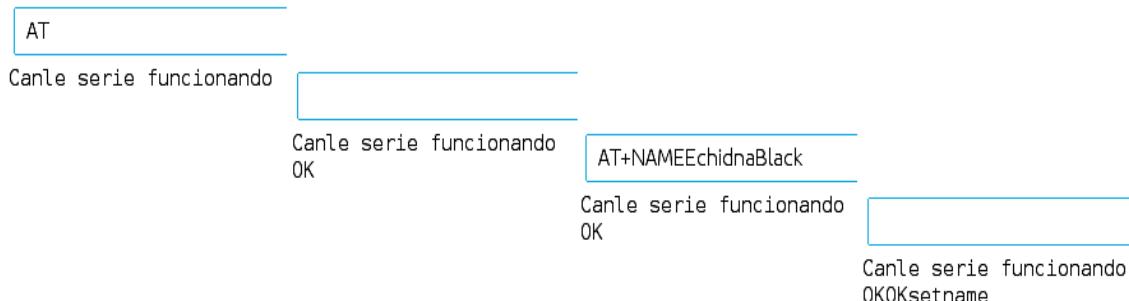
```

        }
        if (Serial.available()) {
            SoftSerial.write(Serial.read()); // Si ten datos os pasa a canle Softw
        }
    }
}

```

Unha vez feitas as conexións, o led do módulo Bluetooth tilila, indicando que está listo, pero non conectado por radio, este é o modo para poder programalo, enviamos o programa a EchidnaBlack e abrimos a terminal serie (9600 bps), nos mostrará amansaxe de cortesía "**Canle serie funcionando**", para comprobar que temos comunicación enviamos o comando de atención "AT", nos contestará "OK" (se non contesta a primeira insiste que o SoftSserial non é moi rápido)

Para cambiar o nome usamos "AT+NAMEName" contestará "OKsetname"



Agora só nos queda programar a velocidad "AT+BAUD8" a 115200 baudios ou bps



Contesta OK e uns caracteres non recoñecibles, eses caracteres son transmitidos a 115200bps, o programa e a consola serie estan funcionado a 9600, e non os interpretan ben.

Temos listo o módulo Bluetooth programado, xa podemos usalo na EchidnaBlack.

Comandos AT para HC-06.

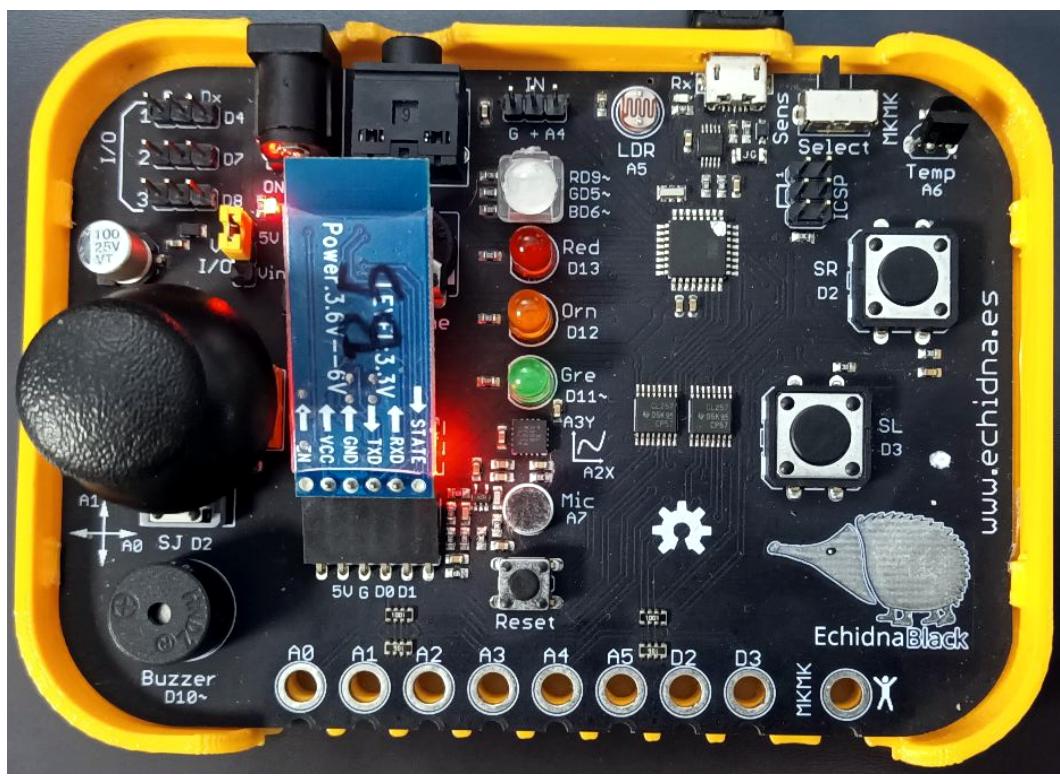
| | Envío | - | Resposta | Exemplo |
|----------|---------------|---|-------------|-----------------------|
| Test: | AT | - | OK | |
| Nome: | AT+NAME<NOME> | - | Oksetname | (AT+NAMEEchidnaBlack) |
| Pin: | AT+PIN<Pin> | - | OKsetPIN | (AT+PIN1234) |
| Versión: | AT+VERSION | - | OK<Versión> | OKLinvor1.8 |



Velocidade: AT+BAUD<X> - OK<baudrate> (AT+BAUD8)

X =:

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

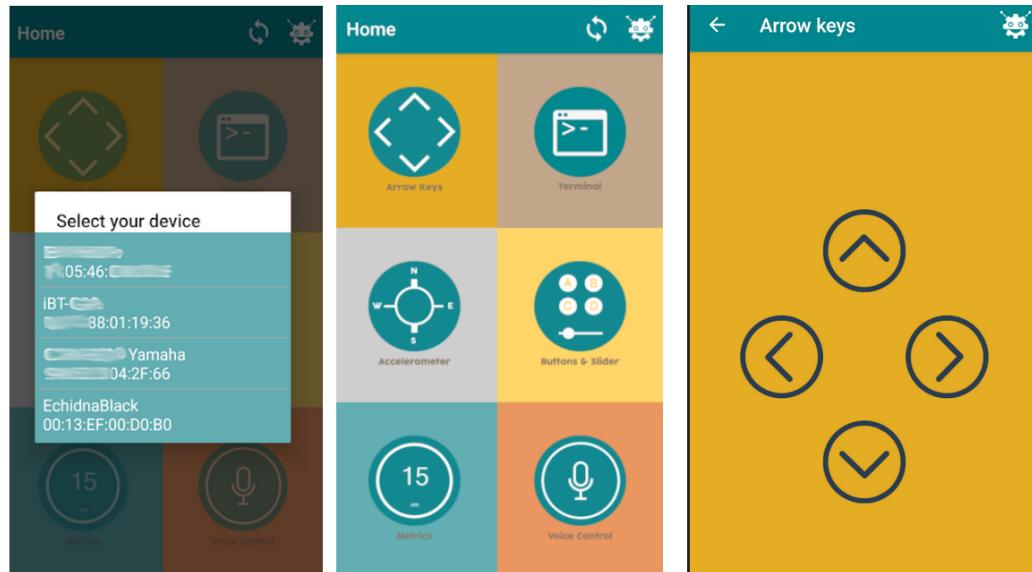
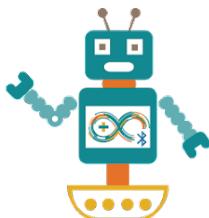


Podemos usar os programas anteriores de control do LED dende o ordenada, agora usando unha app, lembra que para subir os programas a EchidnaBlack é aconsellable non conectar o módulo Bluetooth ata que suba o programa.

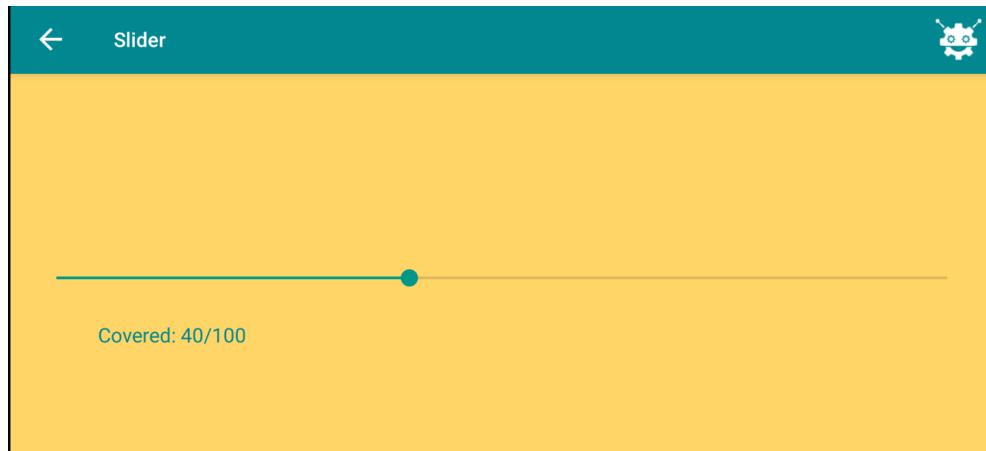
Subimos o programa “Led_Via_Serie”, conectamos o Bluetooth, no SmartPhone imos o menú [Dispositivos..Bluetooth...] [Vincular novo dispositivo] esperamos a que aparezca “EchidnaBlack” o seleccionamos , poñemos o pin “1234” e aceptamos.



Queda instalar a APP "[Arduino Bluetooth Control](#)" de "broxcodes". Seleccionando o dispositivo "EchidnaBlack" xa podemos controlar o LED (cando a APP se conecte o módulo Bluetooth o led de este quedara acendido). Eliximos Arrow keys, A frecha cara arriba acende o LED e a frecha cara baixo o a paga.



Agora probamos co programa "Dim_Led_Via_Serie" e na APP eliximos [Buttons & Slider], xiramos a pantalla e aparece unha barra escorrente, que podemos cambiar para axustar a luminosidade do led.

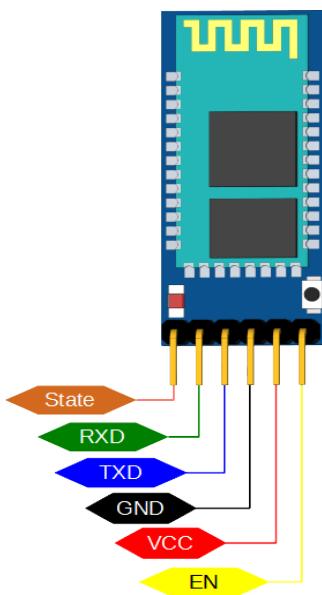


Podemos atopar multitud de APP que son compatibles, teremos que comprobar sempre a velocidade de comunicación



Se o adaptador que temos é un HC-05, este módulo pode ter dous roles escravo ou mestre, o rol que temos no HC-06 é sempre escravo.

O HC-05 ten seis conexións, “State” estado de comunicación, “RXD” recepción de datos, “TXD” transmisión de datos, “GND” masa ou negativo de alimentación e “VCC” Positivo de alimentación e “EN” habilitación ou KEY, nós usaremos ós as catro conexións centrais.



A programación cambia con respecto ó HC-06.

O primeiro e axustar a velocidade “BaudRate” no programa “SoftwareSerial” a 38400 bps. Antes de acender a montaxe premeremos o botón do HC-05 e non o soltaremos ata comprobar que o Led do HC-05 tilila amodo, é indicativo de que entramos en modo programación.

Unha vez que subimos Sketch ca nova velocidade a EchidnaBlack abrimos unha terminal axustamos a velocidade e eliximos Ambos NL e CR.

- Comenzamos co comando de atención “AT” o que responderá “OK”.
- Cambiarlle o nome “AT+NAME=EchidnaBlack”, responderá “OK”
- Clave de conexión AT+PSWD=”1234”, responderá “OK”
- Rol como escravo AT+ROLE=0, responderá “OK”
- Velocidade AT+UART=115200,0,0
- Reiniciar para establecer os cambios AT+RESET
 - [Manual de comandos para HC-03/05](#)

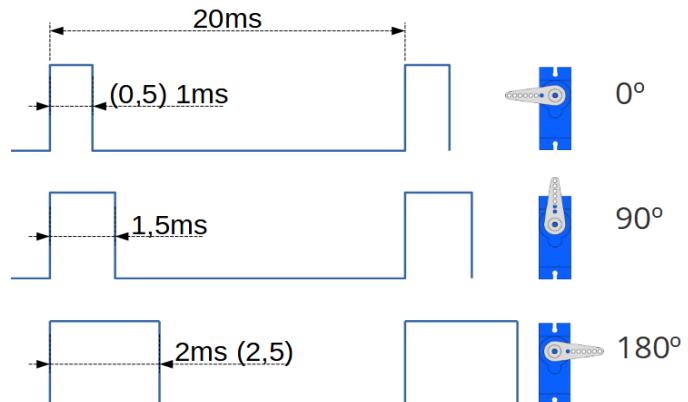


Usando elementos externos.

Servomotor

O “Servo” ten a capacidade de manter unha posición que enviamos mediante o sinal de control.

O rango de xiro típico é de 180°, pero tamén atoparemos servos de rotación continua.

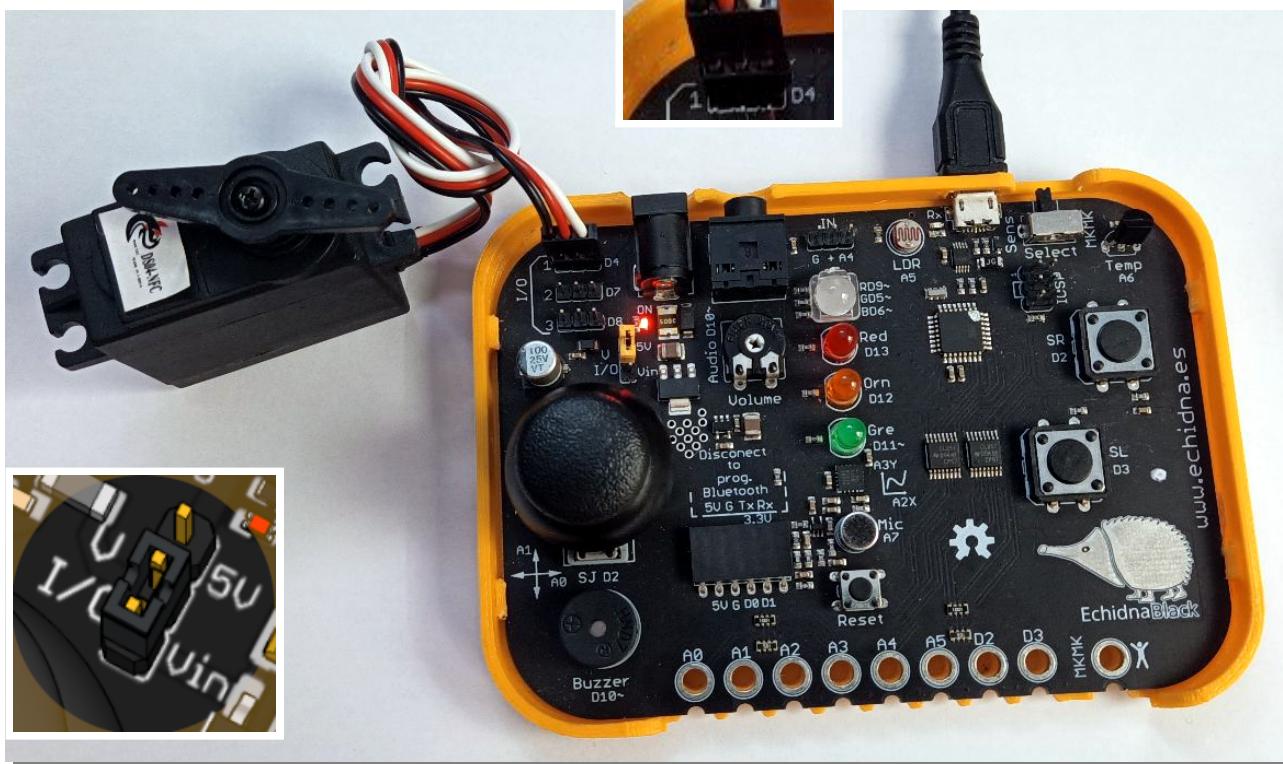


O cable de conexión está formado por tres (3) fíos:

Alimentación (+V)

Negativo (GND)

Sinal de control



- 💡 O Temos que ser conscientes do consumo do(s) servo(s) que imos manexar, si consume máis de 500mA, usar unha alimentación externa e cambiar a ponte a Vin (Ver páxina Selector de Alimentación).

O IDE de Arduino ten exemplos de manexo de servos, con pequenos cambios podemos adaptalos as nosas 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
*/
#include <Servo.h>

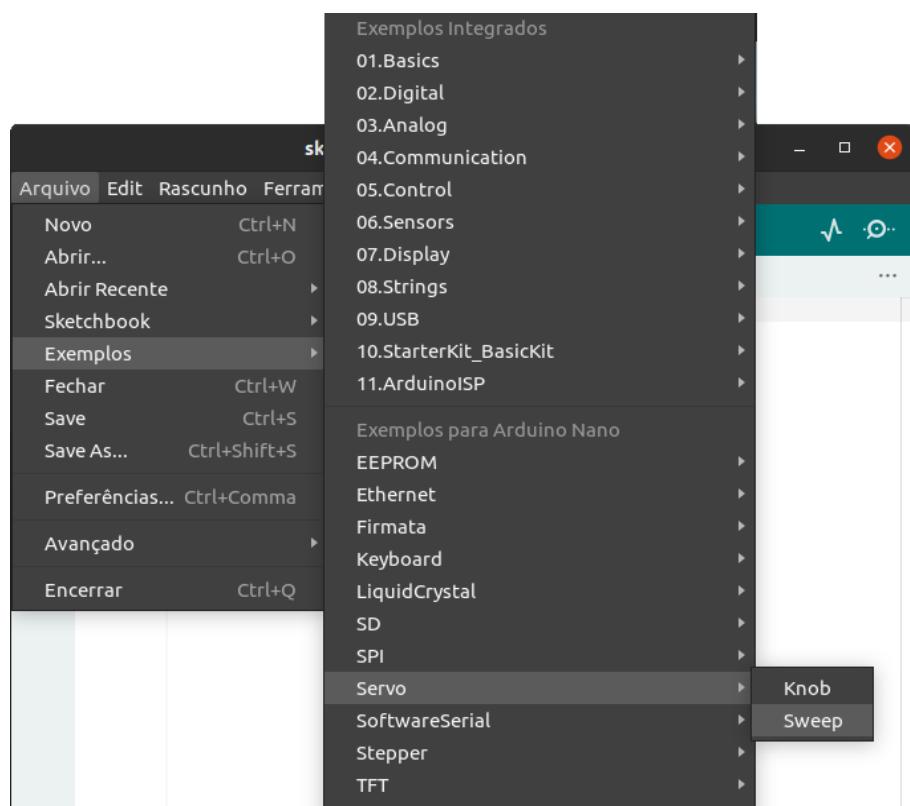
//librería de control do
servo
Servo myservo; //creamos o obxecto myservo para controlar o servo
#define Joy_X A0 //pin conectado o eixo X do Joystick
int val; //variable para almacenar o valor de lectura do Joystick

```
void setup() {  
    myservo.attach(4); //conectamos o pin 4 o obxecto myservo  
}
```

```
void loop() {  
    val = analogRead(Joy_X); //leemos o potenciómetro do Joystick (X)  
    val = map(val, 0, 1023, 0, 179); //escala os valores de entrada  
    myservo.write(val); //envía a posición do Servo  
    delay(15); //unha espera para que o servo chegue a posición  
}
```

💡 Analise:

Aquí atopamos “**#include <Servo.h>**” que inclúe no programa a librería servo, que facilita o control do servo. As librerías son programas escritos por outras persoas que nos axudan a engadir novas funcionalidades os nosos programas e facilitan a conexión con sensores e actuadores. Mais adiante volveremos a falar de librerías .



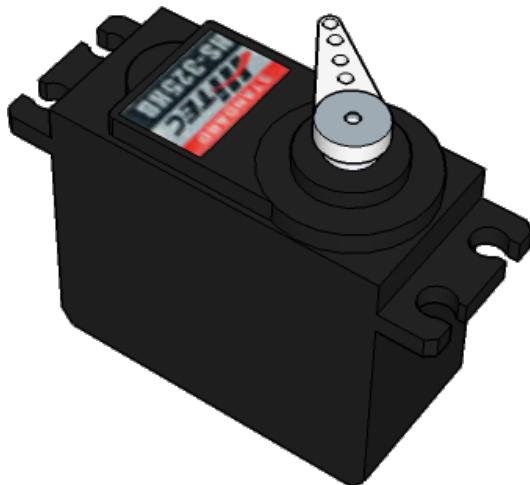


Na instrución “**Servo** myservo;” declaramos myservo como un obxecto ou variable, co método “myservo.attach(4);” indicámoslle en que pin temos conectado o servo neste caso o pin 4.

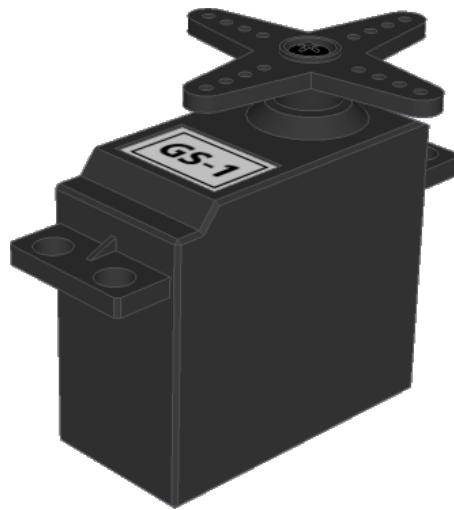
Realizamos a lectura do eixo “X” Joystick, que nos devolta un valor de 0 a 1023, mediante a instrucción “val = map(val, 0, 1023, 0, 179);” escalamos os valores de entrada os valores en grados que pode manexar o Servo.

O seguinte método que usamos é “myservo.write(val);” donde lle pasamos o valor “val” en grados que ten que xirar.

Para controlar un servo de rotación continua, temos que ter en conta que quedará parado cando lle enviamos un valor de 90° , xirará en sentido horario cando supere o valor de 90° chegando a máxima velocidade o rondar os 179° , se o valor é inferior a 90° xirará en sentido antihorario e cando ronde os 0° collerá a máxima velocidade.



$0 - 179^\circ$



Xiro continuo

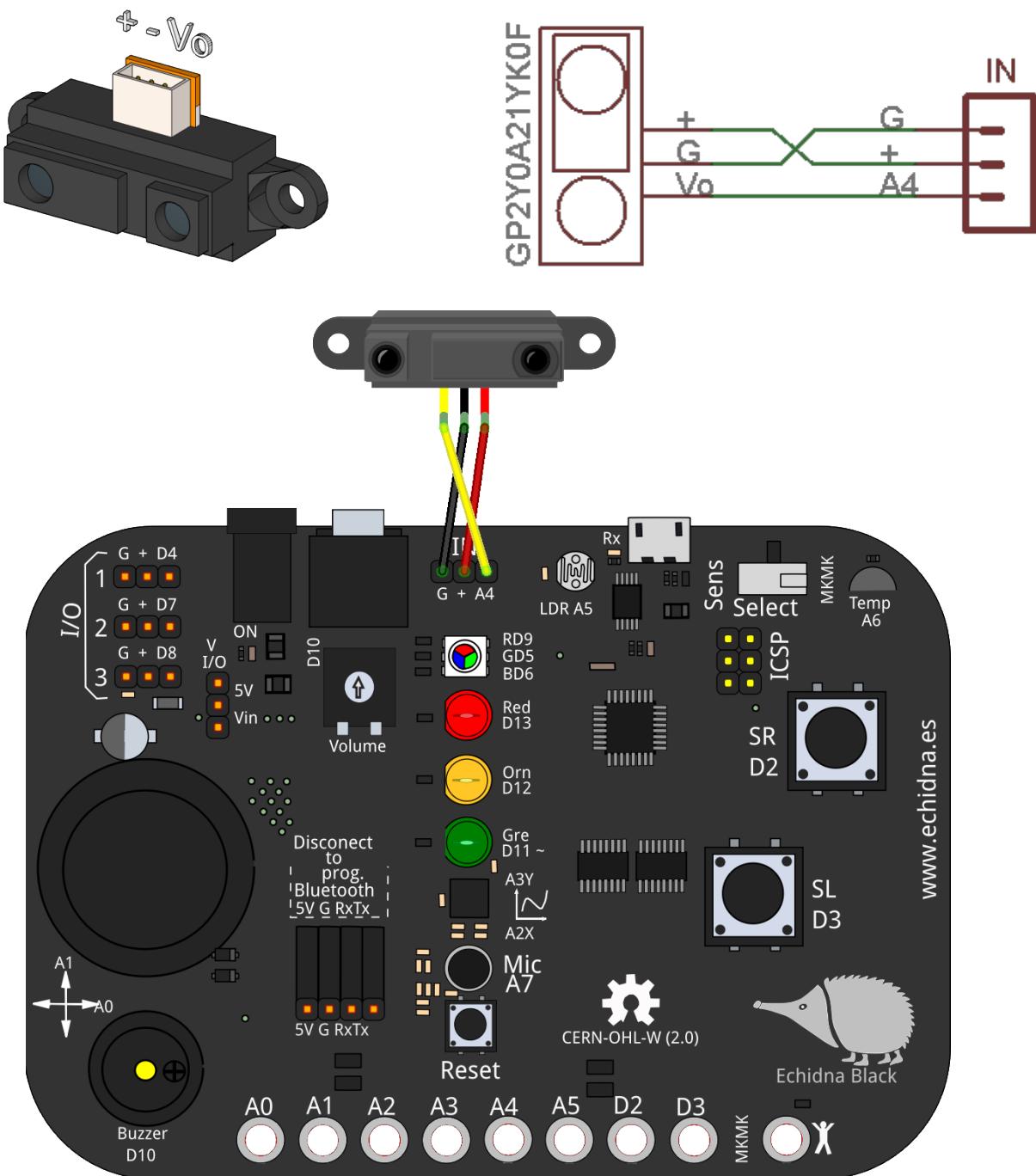


Medida de distancia.

Sensor de proximidad de infravermellos Sharp 0A41SK

É un sensor de distancia que proporciona unha tensión segundo o ángulo de rebote dos infravermellos nunha superficie.

Ten un rango de detección de 10 a 80 cm, consume uns 30mA a 5Vcc





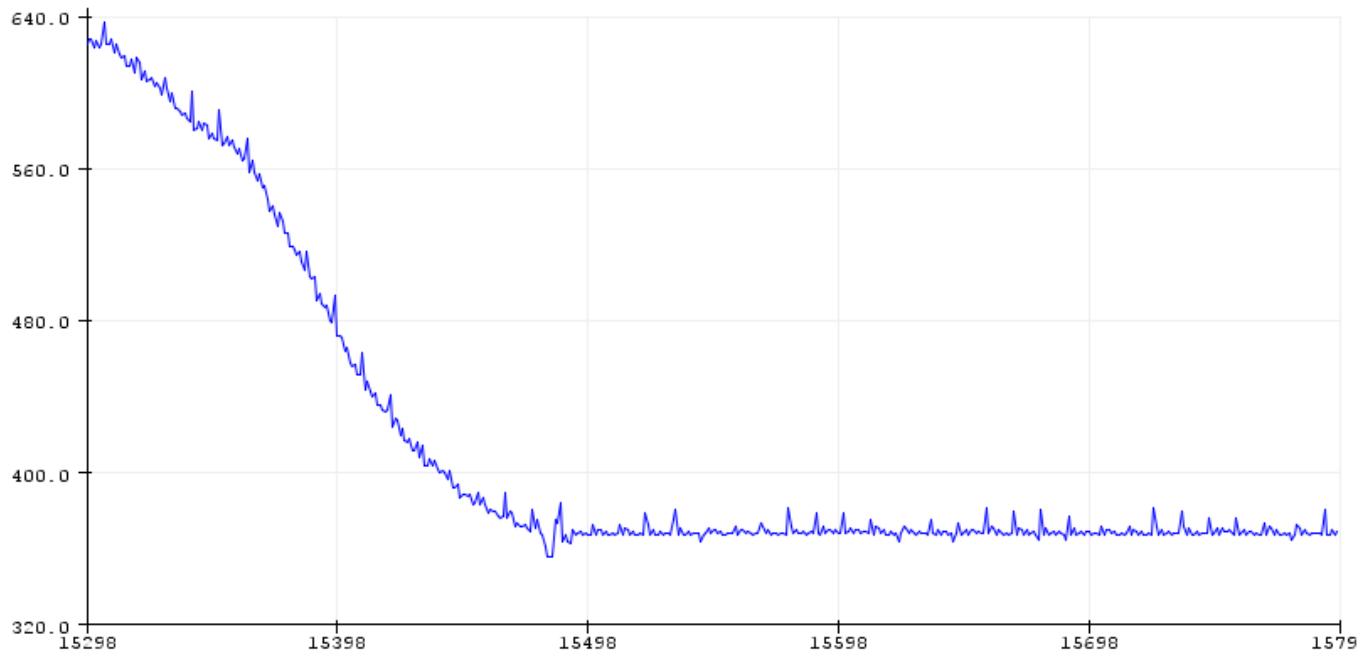
Podemos medir simplemente o valor da tensión das saída do sensor.

```
/* OA41SK_simple
 * Medida directa dos valores da saída do sensor
 */
#define Sensor A4 //pin conectado o sensor
int ValSensor = 0; //Variable de lectura

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

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

Usando o Plotter serie comprobamos que temos bastante ruído.

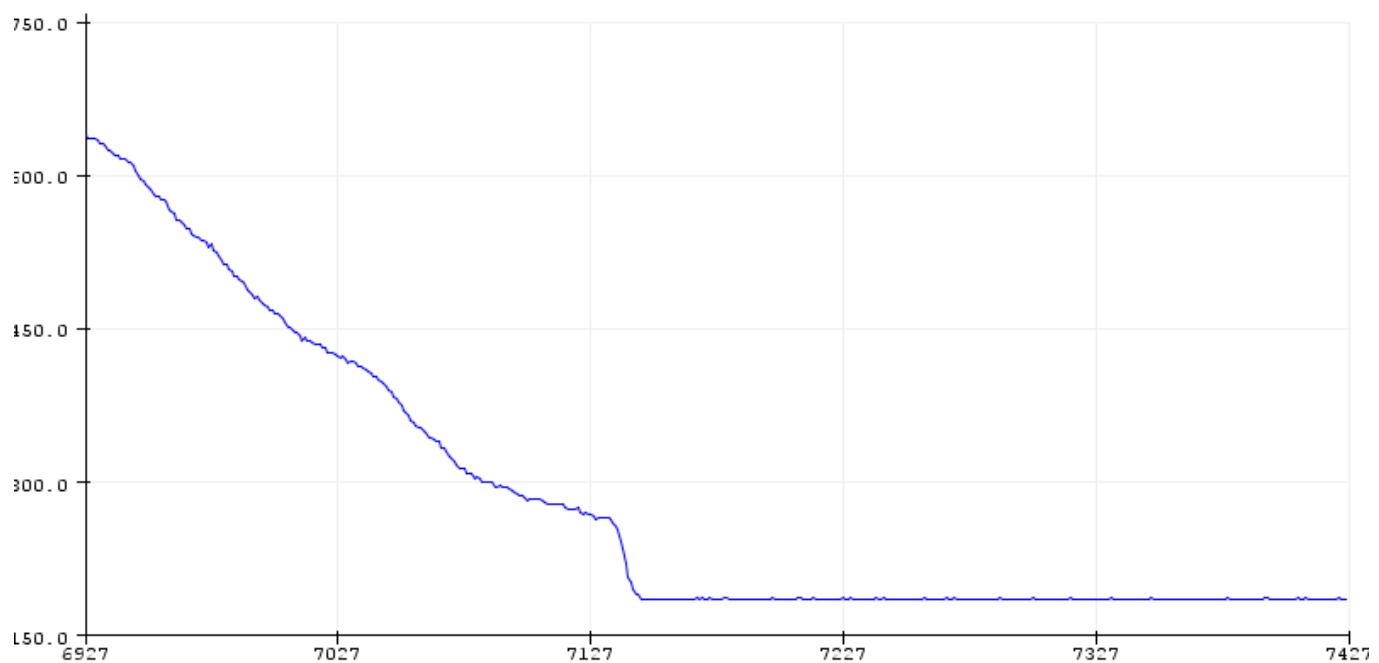




Para tentar evitar isto podemos realizar unha media de medidas:

```
/* 0A41SK_media
   Media dos valores da saída do sensor
*/
#define Sensor A4 //pin conectado o sensor
int ValSensor = 0; //Variable de lectura
int n = 20;           //numero de lecturas
long Media = 0;     //valor a acumular medidas e media
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 do sensor
        Media = Media + ValSensor;    //acumula medidas
        delay(1);                   //pausa para estabilizar
    }
    Media = Media/n;           //calcula media
    Serial.println(Media);    //envía a media vía serie
}
```

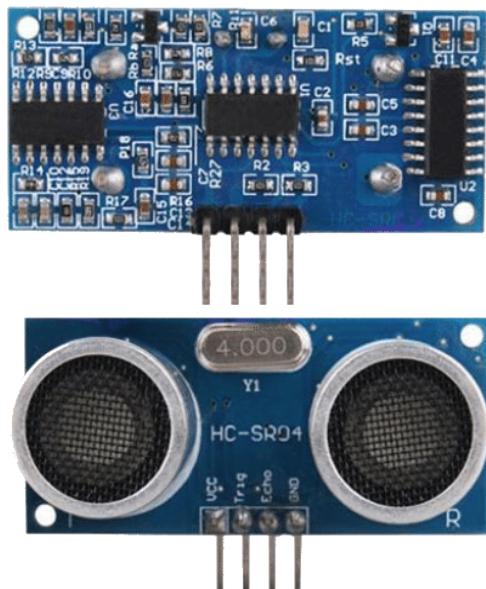
Podemos observar que a medida xa é más limpia.





Sensor distancia con ultrasonidos HC-SR04

O sensor HC-SR04 usa o sonar para determinar a distancia a un obxecto como fan os morcegos ou os golfinos. Ofrece unha excelente detección no rango de 2 cm a 400 cm. O seu funcionamento non se ve afectado pola luz solar ou o material negro como o sensor Sharp, aínda que os materiais acusticamente absorbentes como un pano poden ser difíciles de detectar.



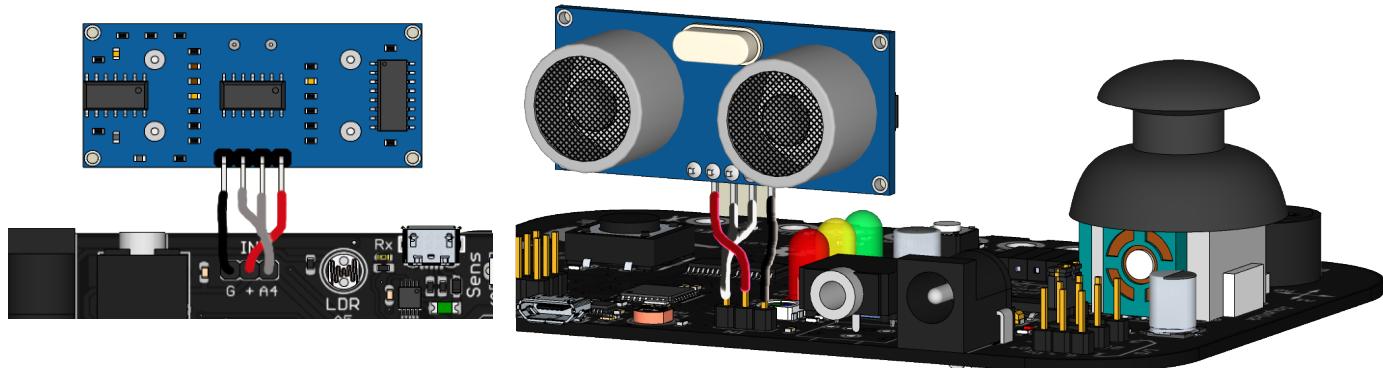
Características:

- Alimentación: +5V DC
- Corriente de repouso: <2mA
- Corriente de trabajo: 15 mA
- Ángulo efectivo: <15°
- Distancia de alcance: 2 cm – 400 cm
- Resolución : 0,3 cm
- Ángulo de medición: 30 graos
- Ancho de pulso “Trigger”: 10uS

Ten catro pines de conexión:

| | | |
|------|---|--|
| VCC | = | +5VDC. |
| Trig | = | “Trigger” Disparo do sensor. |
| Echo | = | “Echo” Pulso de saída, ancho proporcional a distancia. |
| GND | = | GND Masa (-) ou negativo de alimentación. |

Podemos conectar os dous pines Trig e Echo entre si, e así poder conectalos a EchidnaBlack





```
/* UltraSon
 * Medidor de distancias mediante ultrasóns HC-SR04
 * conectado a entrada/saída IN "A4", como só ten un pin,
 * unimos os pines "Trigger e ECHO" do HC-SR04 e conectalos o IN
 */
#define IN A4 //pin de conexión
const int Vson = 34300.0; // Velocidade son en cm/s
void setup() {
    Serial.begin(9600); // Inicia a conexión serie
}

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

    //Para obter a distancia en cm, convite o tempo a segundos xa que está en
    microsegundos
    //por iso multiplicamos por 0.000001 (podemos axustar este valor por as
    tolerancias do sensor)
    int distancia = tempo * 0.000001 * Vson / 2.0;

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

💡 Analise:

O procedemento de lectura é moi sinxelo: Configuramos o pin (A4) como saída e aplicamos un “1” durante 10ms.



Seguidamente configuramos o pin (A4) como entrada, e medimos o ancho do pulso que devolta o HC-SR04, que multiplicado pola velocidade do son e dividido entre 2 temos a distancia na que rebota o sinal. Se pasamos o tempo a segundos, tendo a velocidade do son en cm/s, xa teremos a distancia en cm.



Para os seguintes exemplos usaremos algunha librería, e simplificar os programas.

Instalar librerías

O IDE Arduino ven con bastantes librerías preinstaladas, na páxina de arduino.cc/reference/en/libraries podes ver información de cada unha de elas

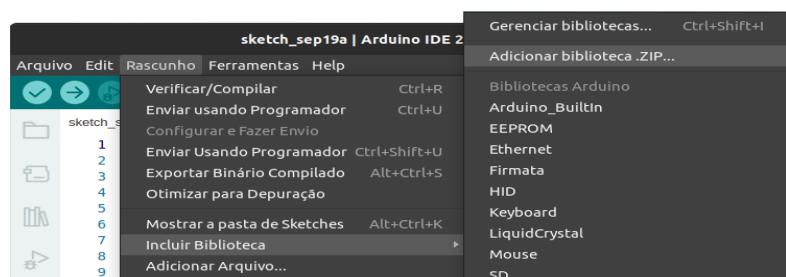
Temos varias formas para incluír librerías no IDE Arduino:

Utilizar a icona  ou no menú [Ferramentas] [Xestionar bibliotecas] permítenos instalar novas e mantelas actualizadas dos repositorios.

- Para incluíla no noso programa usamos o menú [Sketch] [Incluír libraría]
 - As librerías non estándar de Arduino temos que instalalas nos, para facelo temos vías formas

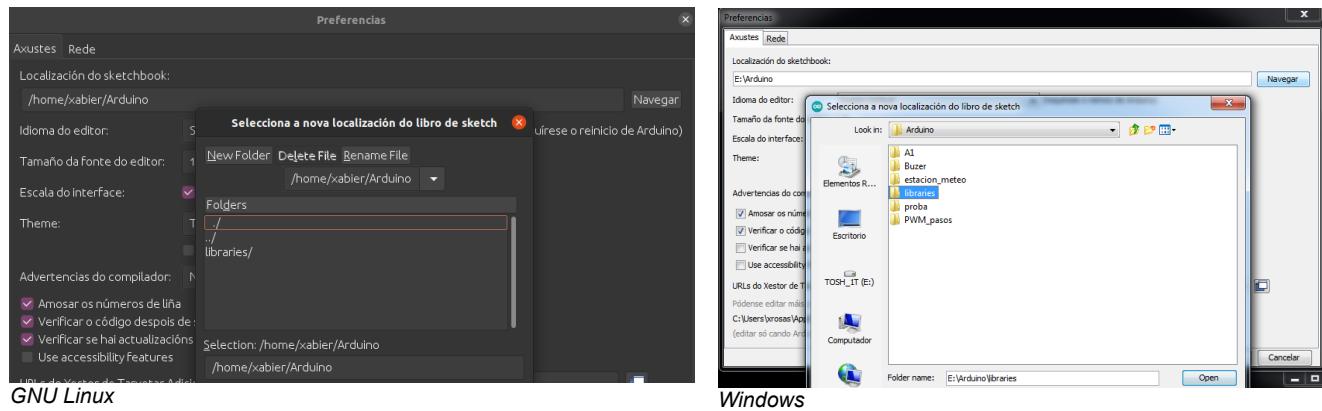
Podemos incluíla

dende un arquivo “.zip”





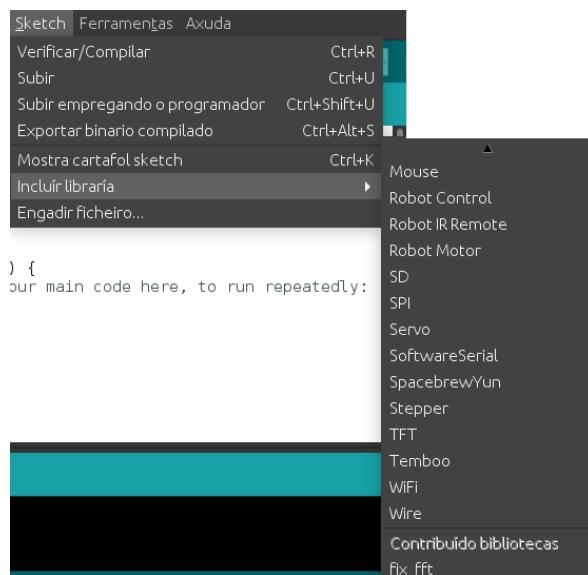
Outra forma para versións anteriores a 2.0.0 é totalmente manual, descargamos a libreira e a descomprimimos dentro do cartafol de librías do IDE Arduino, a localización de ese cartafol podemos atopala nas preferencias de Arduino



Unha vez descomprimida a libreira temos que reiniciar o IDE Arduino para que aparezca a libreira no menú.

Lembra que non todas as librías son eficaces, non sempre se depuran manteñen no tempo, non son fáciles de usar, o normal é que teñan varios exemplos de utilización.

Non debemos de ter instaladas librías duplicadas, xa que o compilar o programa o IDE collerá a primeira que apareza no listado de preferences.txt, e pode ser que non é a que queremos.



Se temos dubidas podemos pasarnos polos foros de Arduino para buscar información

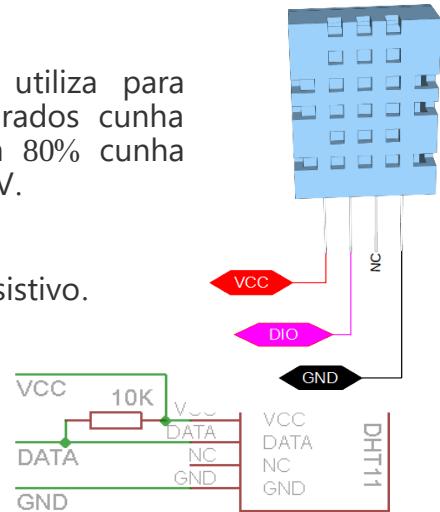


Medida do % de humidade e temperatura co DHT-11.

O DHT11 é un sensor de baixo prezo, que se utiliza para medir a temperatura (nun rango de 0 a 50 graos centígrados cunha precisión de $\pm 2^{\circ}\text{C}$) e a humidade (nun rango de 20% a 80% cunha precisión de $\pm 5\%$) alimentase con unha tensión de 3 a 5V.

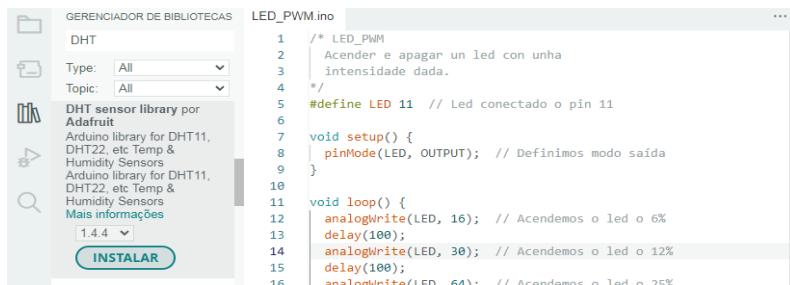
Para a medición da temperatura, ten un termistor NTC resistivo.

Para conectalo a Echidna Black precisamos conectar unha resistencia “Pull-Up” a saída de datos. (Podemos usar usar a instrucción “`pinMode` (`Pin_Sen`, `INPUT_PULLUP`);” para activar a resistencia Pull-up interna).



Usaremos a librería DHT de Adafruit, a instalaremos como vimos no punto anterior :

Lembra instalar todo xa que esta librería de pende da Adafruit_sensor.h.



Neste exemplo imos usar o pin I/O 1 (D4)

```
/*
 * DHT_HT
 * Lectura da %de humidade e temperatura co sensor DHT11
 * Baseado no programa escrito por ladyada (Limor Fried) de Adafruit
 * Precisamos as librerias DHT-sensor-library e Adafruit_Sensor
 */

#include "DHT.h"

#define Sensor DHT11 // tipo de sensor, tamén podemos usar o DHT22 que é mellor
#define Pin_Sen 4 // pin onde conectamos o sensor
DHT dht(Pin_Sen, Sensor); //Sensor conectado en D4 e o modelo DHT11
int h; //variable para almacenar % da humidade
int t; //variable para a temperatura
void setup() {
    Serial.begin(9600); // inicia as comunicacóns
    Serial.println("Test DHT11!");
}
```



```
dht.begin(); // iniciamos o sensor as medidas
pinMode (Pin_Sen, INPUT_PULLUP); //Activamos a resistencia interna Pull up)
}

void loop() {
delay(2000); //Esperamos dous segundos entre medidas, é un sensor moi lento
int h = dht.readHumidity(); // Leemos a %de humidade
int t = dht.readTemperature(); // Leemos a temperatura

// Check if any reads failed and exit early (to try again).
// Chequeamos si algunha medida falla e tentalo outra vez
if (isnan(h) || isnan(t)) {
Serial.println("Faio de lectura do DHT!");
return;
}
int hic = dht.computeHeatIndex(t, h, false); // Devolta o valor en °C
// (Fahreheit = false)
Serial.print(F("Humididade: "));
Serial.print(h);
Serial.print("% Temperatura: ");
Serial.print(t);
Serial.println(F("°C "));
}
```

⌚ Analise:

O sensor utilizado non é o mellor pero é un dos mais accesible e o atoparemos en moitas montaxes con Arduino, o DHT22 ten maior rango e maior resolución. Este é un claro exemplo do que aforramos usando unha boa librería.

Solo iniciamos o sensor e lle pedimos as medidas.

```
Mensagem (Ctrl + Enter para enviar mensagem para 'Arduino Nano' em '/dev/ttyUSB0' Nova linha 9600 baud
Test DHT11!
Humidade: 54% Temperatura: 27°C
Humidade: 58% Temperatura: 27°C
Humidade: 61% Temperatura: 27°C
Humidade: 63% Temperatura: 27°C
```



Sensor capacutivo de humidade do chan.

Para medir a humidade do chan existen múltiples técnicas (Resistivo, reflectometría no dominio da frecuencia e sensores de neutróns...) neste caso imos usar un sensor capacitivo, canto maior sexa o contido de auga no chan maior será a capacitancia, e menor será a tensión de saída do sensor. Este sensor esta baseado no famoso temporizador NE555.

```
/* Humid_Capacitivo
 * Lectura da humidade do chan
 * Memoriza dous puntos un seco é outro húmido
 */
```

```
#define Sensor A4 // Entrada analólica para conectar o sensor
#define SR 2 // Entrada Pulsador SR para memorizar valores (calibrar)
#define L_Red 13 // LED vermello (Seco)
#define L_Org 12 // LED laranxa (Normal)
#define L_Gre 11 // LED verde (Húmido)
#define RGB_B 6 // LED azul (bomba funcionando)
#define Rele 4 // Saída I/O 4 relé para a bomba de rego
#define Buz 10 // Buzer (Zoador)
```

```
int ValSensor; // Variable para Valor de lectura do sensor
int VSeco = 510; // Valor inicial para seco
int VHumi = 300; // Valor inicial para húmido
double Media; // Valor medio das mostras do sensor
int n = 512; // Numero de lecturas para o valor medio
void setup() {
    Serial.begin(9600); // Inicializa as comunicacóns
    pinMode (Sensor, INPUT); // Configura as entradas/sáidas
    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 a media aritmética das medidas de humidade
```





```
media();
Serial.println(Media);
if (Media >= VSeco) { // O valor indica humidade moi baixa
    Vermello(); // Acende o LED vermello
    Rele_On(); // Activa o relé e o LED azul
}
if (Media > VHumi and Media < VSeco) { // Valor de humidade correcto
    Laranxa(); // Acende o LED laranxa
}
if (Media <= VHumi) { // Valor de humidade alta
    Verde(); // Acende o LED Verde
    Rele_Off(); // Apaga o relé e o LED azul
}
// Lectura do pulsador SR para entrar no menú de memorizar valores
if (digitalRead (SR) == 1) {
    tone (Buz, 4000, 200); //Beep
    memoriza(); //Chama a función memoriza
}
}

// Función para actualizar os valores de seco e húmido
void memoriza() {
    Rele_Off(); // Apaga o relé e o LED azul mentres se calibra
    // Memoriza o valor da terra seca
    Serial.println ("Pincha o sensor en terra seca");
    Serial.println ("Cando este listo pulsa SR");
    delay(500);
    while (digitalRead (SR) == 0) { //Mentres non se pulse SR
        digitalWrite (L_Red, !digitalRead(L_Red)); //Parpadeo do LED vermello
        delay(50);
    }
    tone (Buz, 2300, 200); //Beep, pulsouse SR
    media(); // Calcula a media de n medidas
    VSeco = Media; // Asigna o valor actual como terra seca
    digitalWrite (L_Red, 0);
    visu(); //Mostra os valores memorizados ata o momento

    // Memoriza o valor da terra húmida
    Serial.println();
    Serial.println ("Pincha o sensor en terra humida");
```



```

Serial.println ("Cando este listo pulsa SR");
delay(500);
while (digitalRead (SR) == 0) { //Mentres non se pulse SR
    digitalWrite (L_Gre, !digitalRead(L_Gre)); //Parpadeo do LED verde
    delay(50);
}
tone (Buz, 2300, 200); // Beep
media();
VHum = Media; // Asigna o valor actual como terra húmida
while (digitalRead (SR) == 1 ) {} // Espera que solte o pulsador SR
digitalWrite (L_Gre, 0); // Apaga o LED Verde
// Si os valores memorizados están mal, volvemos a lectura dos valores
if (VHum >= VSeco) {
    tone (Buz, 200, 1000); //Beep erro
    Serial.println ("Erro");
    Serial.println ("Volve a calibrar o sensor!");
    memoriza(); //volvemos a función memoriza (non fálelo moito)
}
// calculo da media da medida do sensor
void media() {
    for (int i = 0; i < n; i++) { //Bucle de n medidas
        ValSensor = analogRead(Sensor); //lectura do sensor
        Media = Media + ValSensor; //acumula as medidas
        delay(1);
    }
    Media = (int) (Media / n); // Fai a media das n medidas e elimina os decimais
}

//Acende o Relé e o LED azul
void Rele_On() {
    digitalWrite(Rele, 1);
    digitalWrite(RGB_B, 1);
}
//Apaga o Relé e o LED azul
void Rele_Off() {
    digitalWrite(Rele, 0);
    digitalWrite(RGB_B, 0);
}

```



```
//Acende o LED vermello apaga os outros
void Vermello() {
    digitalWrite (L_Red, 1);
    digitalWrite (L_Org, 0);
    digitalWrite (L_Gre, 0);
}

//Acende o LED laranxa apaga os outros
void Laranxa() {
    digitalWrite (L_Red, 0);
    digitalWrite (L_Org, 1);
    digitalWrite (L_Gre, 0);
}

//Acende o LED verde apaga os outros
void Verde() {
    digitalWrite (L_Red, 0);
    digitalWrite (L_Org, 0);
    digitalWrite (L_Gre, 1);
}

//Mostra os valores memorizados ata o momento.
void visu() {
    Serial.print("VSeco = ");
    Serial.print(VSeco);
    Serial.print(" VHumi = ");
    Serial.print(VHumi);
    Serial.println();
}
```

⌚ Analise:

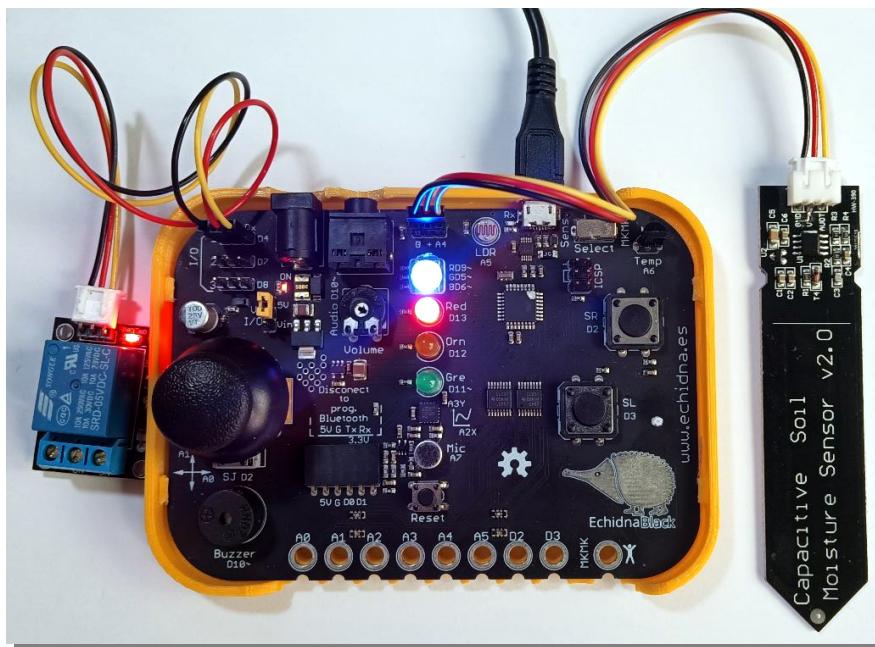
No programa “sketch” a parte das definición e axuste de roles de entrada/saída, atopamos con tres comparacións para determinar sis acendemos a bomba de rego. Cando a media aritmética das lecturas do sensor de humidade capacutivo e maior que o Valor Seco fará que se active a bomba sinalizando mediante o LED azul, e permanecerá activo ata que se detecte que o valor do sensor e inferior o límliar húmido, se o valor de medida está entre os dous valores acendemos o LED laranxa.

Un exame cílico no “loop” comproba a pulsación de SR, (beep), este nos levará a función de memorizar (calibrar) novos valores de chan seco e húmido. Fai un cambio de estado rápido do Led vermello, para indicar que temos que por o sensor en terreo seco, esta é unha forma moi simplificada de cambiar de estado



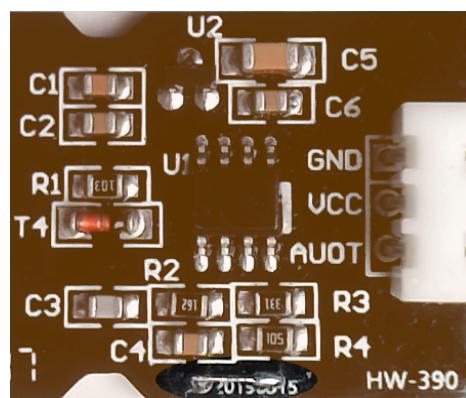
unha saída, consiste en por a saída no estado negado “`!digitalRead(L_Red)`” do que tiña previamente e esperar un pequeno tempo. O premer “Sr” e antes de saír desta función, realizase unha comprobación de valores si o valor húmedo e maior ou igual que o valor seco indicaría que non temos medidas correctas, e nos devolta a calibración con beep de erro.

As funciós de activación dos LEDes, Relé... están fora do bucle “loop” para maior claridade.



Este sketch presenta un inconveniente, si apagamos perdemos os valores de calibración, para solucionar isto podemos gardar os valores na memoria EEPROM do microcontrolador, así que imos adicar un rato a aprender un pouco a usar esa memoria non volátil, e faremos unha Version nova do programa anterior, que sexa quen de gardar os datos límites de calibración do sensor capacitivo.

Nota: algúns sensores teñen un erro de fabricación, falta a conexión da resistencia R4 a GND, facendo que a resposta sexa moi lenta, unha simple ponte soluciona o erro.





EEPROM

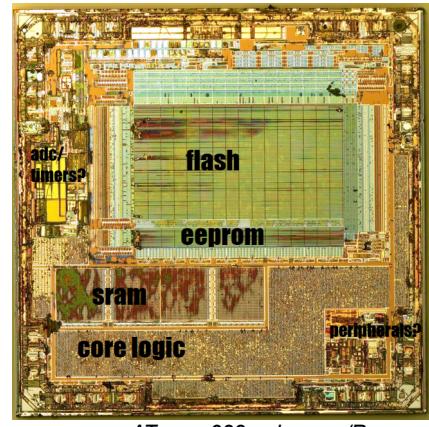
As siglas viñen de “Electrically Erasable Programmable Read-Only Memory” é unha memoria que se pode escribir e borrar electricamente, non se perden os datos cando a apagamos, retendo os datos de forma permanente.

O Atmega328P dispón nesta memoria de 1024 bytes (1 byte son 8 bits), non é unha memoria rápida, tarda uns 3,3ms en escribir cada cela, e 0,3ms en ler. Pode ser escrito/borrado unhas 100 000 veces, non a podemos usar como memoria RAM.

Por defecto a memoria aparece co valor 255 (FF), indicando que está borrada ou non usada.

Usamos a librería EEPROM, que xa ven instalada no IDE Arduino.

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



As funcións más básicas son as “Read” e “Write” que poden ler e escribir un byte (0 a 255) no enderezo indicado (0 a 1023)

```
EEPROM.read(enderezo); // Le no enderezo un byte  
EEPROM.write(enderezo, dato); // Escribe no enderezo o dato (byte)
```

Se queremos escribir cousas más grandes que un byte temos:

```
EEPROM.put(enderezo, dato); // Escribe no enderezo o dato (byte, int, float...)
```

Ollo o usar `EEPROM.put()`, teremos en conta o tamaño dos datos en bytes, que almacenamos, para calcular o enderezo para o seguinte valor

```
EEPROM.get(enderezo, dato); // Le no enderezo un de calquera tipo, ollo  
temos que almacenalo co tipo de dato correcto.
```

Actualizar un dato, só escribe o dato si é diferente o almacenado, isto é moi útil para non gastar ciclos de escritura de EEPROM:

```
EEPROM.update(enderezo, dato); // Só almacena datos tipo byte.
```

Si precisamos comprobar a lonxitude dunha variable podemos usar:

```
lonxitude = sizeof(variable); // Para o caso dunha variable “int” devolverá 2.
```



Sensor capacitivo de humidade con Límites en EEPROM.

```

/* Humid_Capacitivo_EEPROM
Lectura da humidade da terra
Memoriza dous puntos un seco é outro húmido
*/
//*****  

#include <EEPROM.h> // Librería para manexar a memoria EEPROM
//*****  

#define Sensor A4 // Entrada analólica para conectar o sensor
#define SR 2 // Entrada Pulsador SR para memorizar valores (calibrar)
#define L_Red 13 // LED vermello (Seco)
#define L_Org 12 // LED laranxa (Normal)
#define L_Gre 11 // LED verde (Húmido)
#define RGB_B 6 // LED azul (bomba funcionando)
#define Rele 4 // Saída I/O 4 relé para a bomba
#define Buz 10 // Buzer (Zoador)  

int ValSensor; // Variable para Valor de lectura do sensor
int VSeco = 510; // Valor inicial para seco
int VHumi = 300; // Valor inicial para húmido
double Media; // Valor medio das mostras do sensor
int n = 512; // Numero de lecturas para o valor medio
//*****  

byte Memor; // Si é 255 indicará que non se gravou na EEPROM
int EMemor = 0; // Enderezo co dato de comprobación
int EVSeco = 2; // Enderezo para gardar/ler o valor para terra seca
int EVHumi = 4; // Enderezo para gardar/ler o valor para terra húmida
//*****  

void setup() {
    Serial.begin(9600); // Inicializa as comunicacións
    pinMode (Sensor, INPUT); // Configura as entradas/saídas
    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) { // Comproba si valores actualizados na EEPROM
    EEPROM.get(EVSeco, VSeco);      // Pasa os valores as variables
    EEPROM.get(EVHumi, VHumi);
    Serial.print("Valores almacenados"); // Envía os valores almacenados vía
serie
visu();
}

//*****
}

void loop() {
//realiza a media aritmética das medidas de humidade
media();
Serial.println(Media);
if (Media >= VSeco) { // O valor indica humidade moi baixa
    Vermello();           // Acende o LED vermello
    Rele_On();            // Activa o relé e o LED azul
}
if (Media > VHumi and Media < VSeco) { // Valor de humidade correcto
    Laranxa();           // Acende o LED laranxa
}
if (Media <= VHumi) { // Valor de humidade alta
    Verde();              // Acende o LED Verde
    Rele_Off();            // Apaga o relé e o LED azul
}
// Lectura do pulsador SR para entrar no menú de memorizar valores
if (digitalRead (SR) == 1) {
    tone (Buz, 4000, 200); //Beep
    memoriza(); //Chama a función memoriza
}
}

// Función para actualizar os valores de seco e húmido
void memoriza() {
    Rele_Off(); //apaga o rele mentres se calibra
    // Memoriza o valor da terra seca
    Serial.println ("Pincha o sensor en terra seca");
}
```



```

Serial.println ("Cando este listo pulsa SR");
delay(500);
while (digitalRead (SR) == 0) { //Mentres non se pulse SR
    digitalWrite (L_Red, !digitalRead(L_Red)); //Parpadeo do LED vermello
    delay(50);
}
tone (Buz, 2300, 200); //Beep, pulsouse SR
media(); // Calcula a media de n medidas
VSeco = Media; // Asigna o valor actual como terra seca
digitalWrite (L_Red, 0);
visu(); //Mostra os valores memorizados ata o momento

// Memoriza o valor da terra húmida
Serial.println();
Serial.println ("Pincha o sensor en terra humida");
Serial.println ("Cando este listo pulsa SR");
delay(500);
while (digitalRead (SR) == 0) { //Mentres non se pulse SR
    digitalWrite (L_Gre, !digitalRead(L_Gre)); //Parpadeo do LED verde
    delay(50);
}
tone (Buz, 2300, 200); // Beep
media();
VHumi = Media; // Asigna o valor actual como terra húmida
while (digitalRead (SR) == 1 ) {} // Espera que solte o pulsador SR
digitalWrite (L_Gre, 0); // Apaga o LED Verde

// Si os valores memorizados están mal, volvemos a lectura dos valores
if (VHumi >= VSeco) {
    tone (Buz, 200, 1000); //Beep erro
    Serial.println ("Erro");
    Serial.println ("Volve a calibrar o sensor!");
    memoriza(); //volvemos a función memoriza (non fálelo moito)
}
//*****
EEPROM.get(EMemor, Memor); // Recuperamos o valor das veces que se graba
Memor = Memor - 1; // Decrementamos o valor
EEPROM.update(EMemor, Memor); // Actualizamos o valor na memoria EEPROM
EEPROM.put(EVSeco, VSeco); // Actualizamos o valor Seco na memoria EEPROM

```



```
EEPROM.put(EVHumi, VHumi); // Actualizamos o valor Húmido na memoria EEPROM
// Pasamos os valores... vía serie.
Serial.print(" Datos almacenados ");
Serial.print(255 - Memor);
Serial.println(" veces");
visu();
//****************************************************************************
}
// calculo da media da medida do sensor
void media() {
    for (int i = 0; i < n; i++) { //Bucle de n medidas
        ValSensor = analogRead(Sensor); //lectura do sensor
        Media = Media + ValSensor; //acumula as medidas
        delay(1);
    }
    Media = (int)(Media / n); // Fai a media das n medidas e elimina os decimais
}

//Acende o Relé e o LED azul
void Rele_On() {
    digitalWrite(Rele, 1);
    digitalWrite(RGB_B, 1);
}
//Apaga o Relé e o LED azul
void Rele_Off() {
    digitalWrite(Rele, 0);
    digitalWrite(RGB_B, 0);
}
//Acende o LED vermello apaga os outros
void Vermello() {
    digitalWrite (L_Red, 1);
    digitalWrite (L_Org, 0);
    digitalWrite (L_Gre, 0);
}
//Acende o LED laranxa apaga os outros
void Laranxa() {
    digitalWrite (L_Red, 0);
    digitalWrite (L_Org, 1);
    digitalWrite (L_Gre, 0);
}
```



```
//Acende o LED verde apaga os outros
void Verde() {
    digitalWrite (L_Red, 0);
    digitalWrite (L_Org, 0);
    digitalWrite (L_Gre, 1);
}

//Mostra os valores memorizados ata o momento.
void visu() {
    Serial.println();
    Serial.print("VSeco = ");
    Serial.print(VSeco);
    Serial.print(" VHumi = ");
    Serial.print(VHumi);
    Serial.println();
}
```

💡 Analise:

Para que sexa más fácil ver os cambios no programa, estos están escritos entre asteriscos “//*****...”.

O primeiro é comprobar si se fixeron cambios , para eso temos unha variable “Memor” que nos di cantas veces se gravaron os datos comenza en 255 indicando que non se gravaron datos, si se fixeron gravacións e precisamos recuperar os datos comprobamos que a variable é diferente de 255 “if (EEPROM.read(EMemor) != 255)”, si é diferente recuperamos os datos e os pasamos as variables con “EEPROM.get(EMemor, Memor);”...

Para gravar os datos de calibración, unha vez que se comproba que non temos erros, collemos o valor “Memor” e o decrecemos nunha unidade, e actualizase o valor na memoria EEPROM.update(EMemor, Memor);, agora pásanse os datos dos límites Vseco e VHumi a memoria con “EEPROM.put(EVSeco, Vseco);”, non podemos usar “EEPROM.update();” xa que estamos gravando variables “int” e “update“ só funciona con **byte**.

| | |
|--|---|
| Mensagem (Ctrl + Enter para enviar mensagem para 'Arduino Nano' em '/dev/ttyUSB0') Valores almacenados VSeco = 531 VHumi = 253 518.00 520.00 520.00 | Mensagem (Ctrl + Enter para enviar mensagem para 'Arduino Nano' em '/dev/ttyUSB0') Pincha o sensor en terra seca Cando este listo pulsa SR VSeco = 520 VHumi = 372 Pincha o sensor en terra humida Cando este listo pulsa SR Datos almacenados 4 veces VSeco = 520 VHumi = 366 519.00 Valores almacenados VSeco = 520 VHumi = 366 518.00 520.00 |
|--|---|



Receptor de Infravermellos.

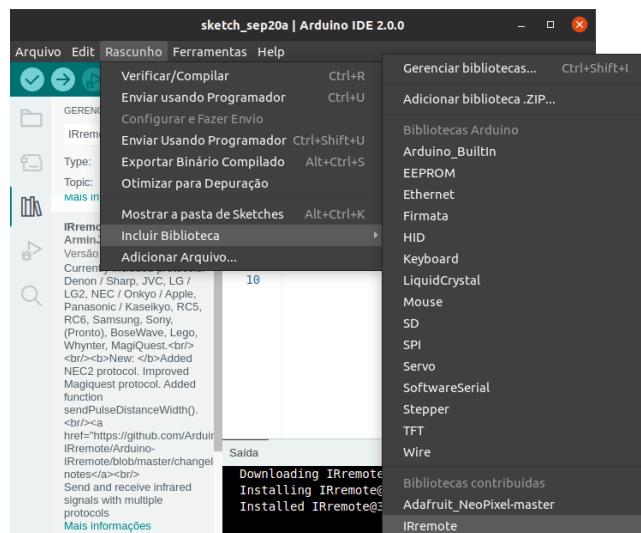
Os receptores de IR de Vishay (entre outros) utilizáñse para aplicacións de control remoto e comunicación de infravermellos, en repetidores/memorización de código de infravermellos, barreiras inmateriais de infravermellos e sensores de proximidade de infravermellos. Os receptores de infravermellos funcionan no rango de lonxitude de onda de 840 nm e 960 nm. Conta con filtro para minimizar os efectos da luz ambiental.

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

Os mandos de control remoto utilizan diversos protocolos de emisión, os más habituais son os de Philips RC5, RC6, NEC, Sony.

Podemos usar a librería “IRremote” que é unha das más usadas, e é compatible coa maioría dos protocolos dos mandos.

Buscamos a libreria “IRremote” e a instalamos.



No menú Sketch xa podemos atopar a librería para usala.

Actualmente é compatible cos seguintes protocolos:

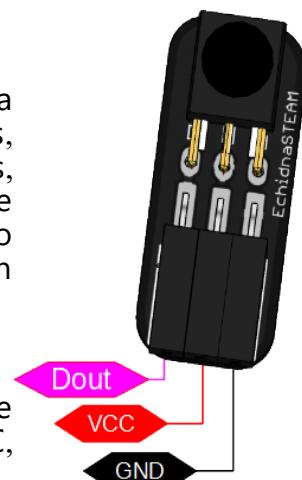
Denon/Sharp, JVC, LG/LG2, NEC/Onkyo/Apple, Panasonic/Kaseikyo, RC5, RC6, Samsung, Sony, BoseWave, Lego, Whynter, MagiQuest.

Autor: shirriff, z3t0, ArminJo.

Mantenedor: Armin Joachimsmeyer

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

Imos facer un pequeno programa que acenda/apague un led con calquera tecla do control remoto (compatible con moitos mandos).





```
/* IR_LED
 * Acender/apagar un LED o recibir
 * datos infravermellos dun control remoto
 */
#include <IRremote.h> //librería instalada

#define IR_PIN A4 //pin para recibir os datos
#define L_RED 13 //LED a controlar

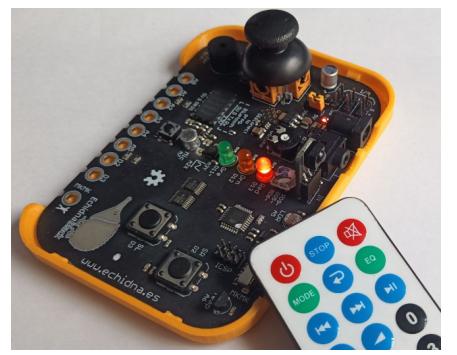
IRrecv Recep_ir(IR_PIN); // asignase a variable de recepción o pin
decode_results resultado; // variable co resultado da recepción
boolean estado = 0;

void setup()
{
    Recep_ir.enableIRIn(); // Habilitamos a recepción IR
    pinMode(L_RED, OUTPUT); // Modo salida
}

void loop() {
    if (Recep_ir.decode(&resultado)) {
        estado = !estado; // Comutamos o estado
        digitalWrite(L_RED, estado); // Ponemos o valor do estado no led
        Recep_ir.resume(); // Preparamos una nova recepción.
    }
    delay(300);
}
```

Analise:

Este exemplo é outra mostra do sinxelo que é usar unha librería, Asignase o pin do receptor e a variable de recepción, usamos unha variable binaria “estado” que a comutamos cada vez que recibimos un dato por infravermellos, na liña “estado = !estado;” facendo que si estado tiña un “0” agora terá un “1”, ese valor o pasamos o estado do LED.



Agora que xa temos un receptor podemos ampliar o programa para recoller os distintos códigos do mando, para poder utilizar EchidnaBlack e un emisor de infravermellos como un Control Remoto.



IR_Receptor

```
/* IR_Receptor

Receptor de infravermellos. Mostra o protocolo e os datos recibidos
Baseado no programa ReceiveDump de Armin Joachimsmeyer

*/
#include <IRremote.h> //librería
#define RECV_PIN A4    //pin asignado o receptor
IRrecv irrecv(RECV_PIN);
void setup()
{
    Serial.begin(9600); // iniciamos as comunicacións serie
    irrecv.enableIRIn(); // Inicia a recepción IR
    Serial.println("Receptor listo ");
}
void loop() {
    if (IrReceiver.decode()) { //se temos recepción entramos na estrutura switch
        digitalWrite (L_RED, HIGH); //Acende o Led cando recibe datos
        Serial.print("Protocolo ");
        switch (IrReceiver.decodedIRData.protocol) { //descodifica o protocolo
            case NEC: // no caso de recibir datos co 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("Non coñecido: ");
                break;
        }
        //Envia os datos vía serie
    }
}
```



```

Serial.print ("Enderezo=0x");
Serial.print ( IrReceiver.decodedIRData.address, HEX);
Serial.print (" Comando=0x");
Serial.println ( IrReceiver.decodedIRData.command, HEX);
delay(200); //espera 200ms
irrecv.resume(); //limpa o bufer para atender outra recepción
digitalWrite (L_RED, LOW); //apaga o Led o rematar a recepción

}

}

```

Analise:

Esperamos en “**IrReceiver.decode()**” a que cheguen datos por infravermellos, e entramos nunha estrutura switch..case. Do mesmo xeito que if, switch..case controla o fluxo do programa. Neste caso na instrucción switch compara o valor “**IrReceiver.decodedIRData.protocol**” co especificado en cada “**case**”, e executa o contido ata a instrución “**break**”.

Admite tamén un “**default**” que se usa para executar un bloque no caso que ningunha das condicións anteriores se cumpra.

Unha vez que se envía o tipo de protocolo, adxuntao enderezo e o comando en hexadecimal.

```

Mensagem (Ctrl + Enter para enviar mensagem para 'Arduino Nano' em '/dev/ttyUSB0'  Nova linha  9600 baud
Receptor listo
Protocolo NEC: Enderezo=0x0 Comando=0x44
Protocolo NEC: Enderezo=0x0 Comando=0x45
Protocolo NEC: Enderezo=0x0 Comando=0x46
Protocolo NEC: Enderezo=0x0 Comando=0x47
Protocolo NEC: Enderezo=0x0 Comando=0x44
Protocolo NEC: Enderezo=0x0 Comando=0x40
Protocolo NEC: Enderezo=0x0 Comando=0x43
Protocolo NEC: Enderezo=0x0 Comando=0x7
Protocolo NEC: Enderezo=0x0 Comando=0x15
Protocolo NEC: Enderezo=0x0 Comando=0x16
Protocolo NEC: Enderezo=0x0 Comando=0x19

```

Ln 1, Col 1 UTF-8 □ Arduino Nano em /dev/ttyUSB0 2 □

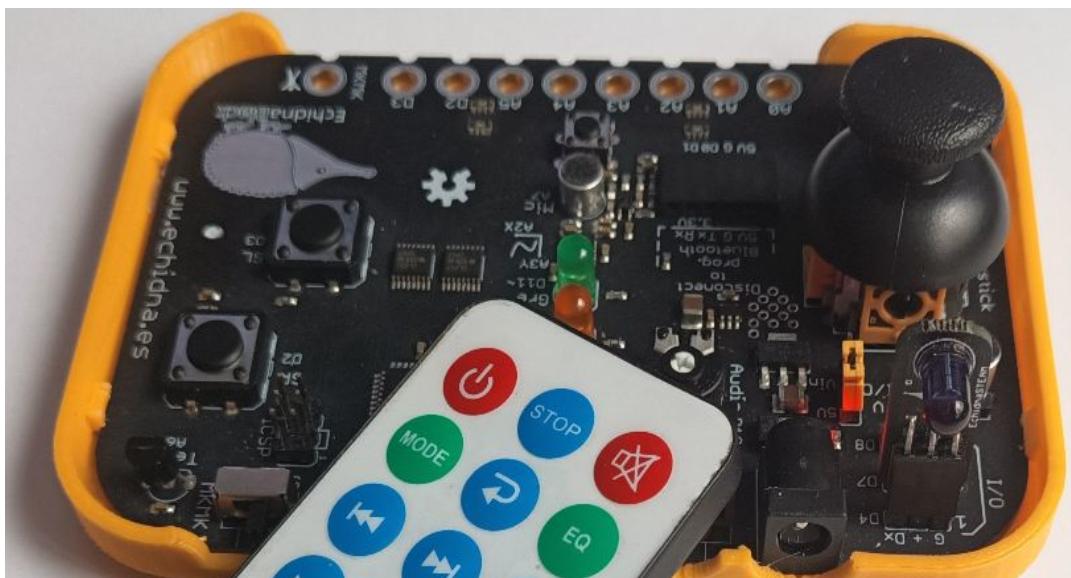


Emisor de Infravermellos.

Usando un led infravermello TSAL4400 de Vishay ou outro compatible, podemos realzar un control remoto (mando a distancia), para poder controlar o aparello que necesitemos, imos facer un pequeno exemplo para ter o control do volume.

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

Usando o programa anterior IR_Receptor, podemos comprobar o enderezo e o comando das teclas do control remoto que queremos emular.



Protocolo NEC: Enderezo=0x0 Comando=0x16
Protocolo NEC: Enderezo=0x0 Comando=0x19

```
/* IR_Emisor
```

Un mando a distancia para o equipo de audio.

O pulsador SR envía comandos de subir volume, SL baixara.

```
*/
```

```
#define IR_SEND_PIN 4 //pin do emisor de Ir
#include <IRremote.hpp>
#define L_Red 13 //LED para indicar que emite
#define SR 2 //Pulsador para enviar subir volume
#define SL 3 //Pulsador para baixar volume
int enderezo = 0x0; //enderezo do mando-receptor
byte subeVol = 0x19; //comando subir volume
byte vaixaVol = 0x16; //comando baixar volume
byte comando ;
```



```

void setup() {
    pinMode(L_Red, OUTPUT); // LED
    Serial.begin(9600);
    IrSender.begin(); //inicializa o envío de datos Ir
    Serial.print("Listo para enviar comandos vía Ir, no pin ");
    Serial.println(IR_SEND_PIN);
}

void loop() {
    if ((digitalRead(SR)) == 1) { //si SR está pulsado sube volume
        comando = subeVol;
        envia(); //chama a función enviar datos
    }
    if ((digitalRead(SL)) == 1) { //si SL está pulsado baixa volume
        comando = vaixaVol;
        envia();
    }
}
// función para presentar datos vía serie e envialos vía Ir
void envia() {
    Serial.println();
    Serial.print("Enviando: enderezo=0x");
    Serial.print(enderezo, HEX);
    Serial.print(" comando=0x");
    Serial.print(comando, HEX);
    Serial.println();
    IrSender.sendNEC(enderezo, comando, 0); //envía datos Ir
    delay(200);
}

```

💡 Analise:

Unha vez que temos os valores do enderezo 0x0, e identificados os comandos a emitir subeVol = 0x19 e vaixaVol = 0x16, xa podemos decidir cando enviamos cada comando, usamos dúas estruturas if, unha para cada pulsador, facendo que o comando teña o valor que nos interesa, usando a función “**IrSender.sendNEC(enderezo, comando, 0);**” enviamos os datos, non esquecer inicializar a emisión Ir “**IrSender.begin();**” na función “**void setup()**”.



Detectando campos magnéticos.

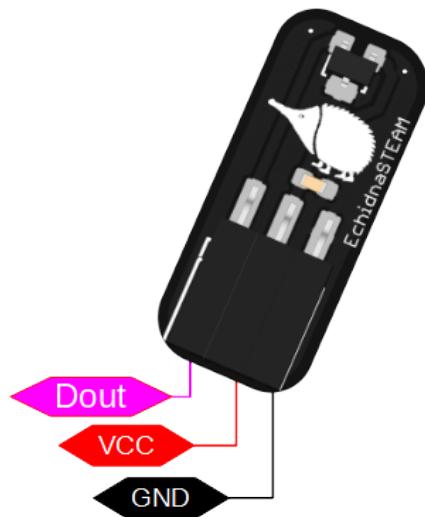
Nesta montaxe imos usar o sensor de Efecto Hall SS39ET da empresa Honeywell, estes sensores entregan unha tensión na saída que varia en función do campo magnético aplicado.

Valores característicos: 1,4mV/Gauss e 2,50V at 0 Gauss, ou sexa a saída terá un valor maior de 2,5V (512) cando detecte un campo magnético norte, e será menor cando o campo sexa sur.

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

```
/*HALL Proximidade
Detectando a proximidade de un campo magnético
Acendendo un par de LEDes como testemuñas
Acende o Led verde ca intensidade do campo detectado
*/
#define HALL A4 //Sensor Efecto HALL conectada a A4
#define L_Red 13 //Led vermello a D13
#define L_Orn 12 //Led laranxa a D12
#define L_Gre 11 //Led verde a D11
int limiarN = 550; //valor activación cara norte
int limiarS = 500; //valor activación cara norte
int ValorHALL;
int ValorMap;
void setup() {
  Serial.begin (9600);
  pinMode (HALL, INPUT); // Definimos como entrada
  pinMode (L_Red, OUTPUT); // Definimos modo saída
  pinMode (L_Orn, OUTPUT); // Definimos modo saída
  pinMode (L_Gre, OUTPUT); // Definimos modo saída
}
void loop() {
  ValorHALL = analogRead(HALL); // Leemos a entrada

  // comproba si o valor supera o limiarN
  if ( ValorHALL > limiarN) {
    digitalWrite (L_Red, HIGH); // Acende o LED vermello
  }
  //Se non supera o limiar norte a pagalo LED
  else {
```





```

digitalWrite (L_Red, LOW); // Apaga o LED vermello
}
// comproba si o valor é inferior a limiarS
if ( ValorHALL < limiarS) {
    digitalWrite (L_Orn, HIGH); // Acende o LED laranxa
}
//No caso contrario apaga o LED laranxa
else {
    digitalWrite (L_Orn, LOW); // Apaga o LED
}
Serial.println(ValorHALL); //Envía o valor vía serie
//Escala o valor de entrada o valor de iluminación do led verde
ValorMap = map (ValorHALL, 150, 900, 0, 64);
analogWrite (L_Gre, ValorMap); // acende o LED verde co valor escalado
// un pequeno retardo para estabilizar as medidas.
delay(10);
}

```

 Analise:

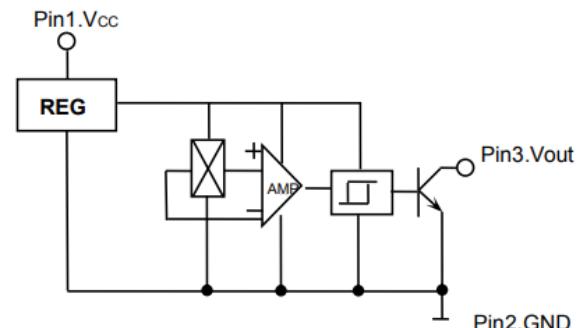
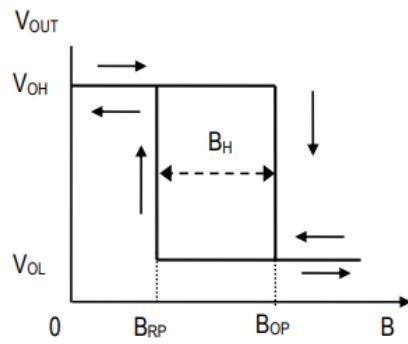
Acende O LED Vermello ando detecte un campo magnético norte e acende o LED laranxa cando detecte un campo magnético sur, os valores de activación podémolos axustar en limiarN e limiarS. O led verde acenderase con unha intensidade proporcional o valor da tensión de saída do sensor, en repouso un valor normal é 523.



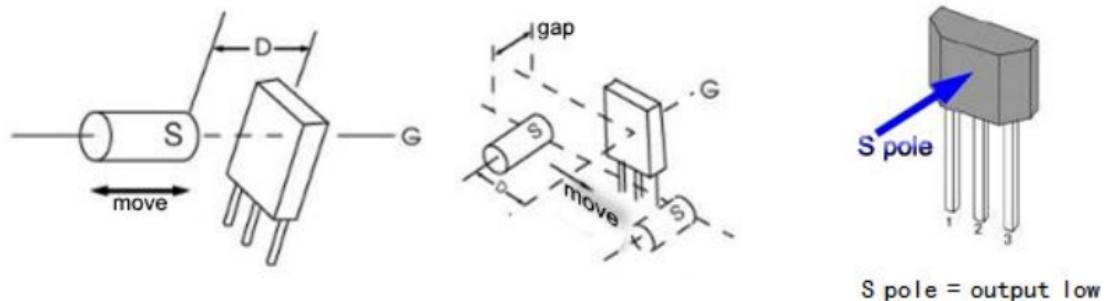


Si o que precisamos e detectar o paso dun imán por diante do sensor (velocímetro bicicleta, sensores de nivel...), podemolo cambiar por un [OH44E](#) ou [DRV5013-Q1](#), que teñen histérese é a saída ten dous estados ben definidos)

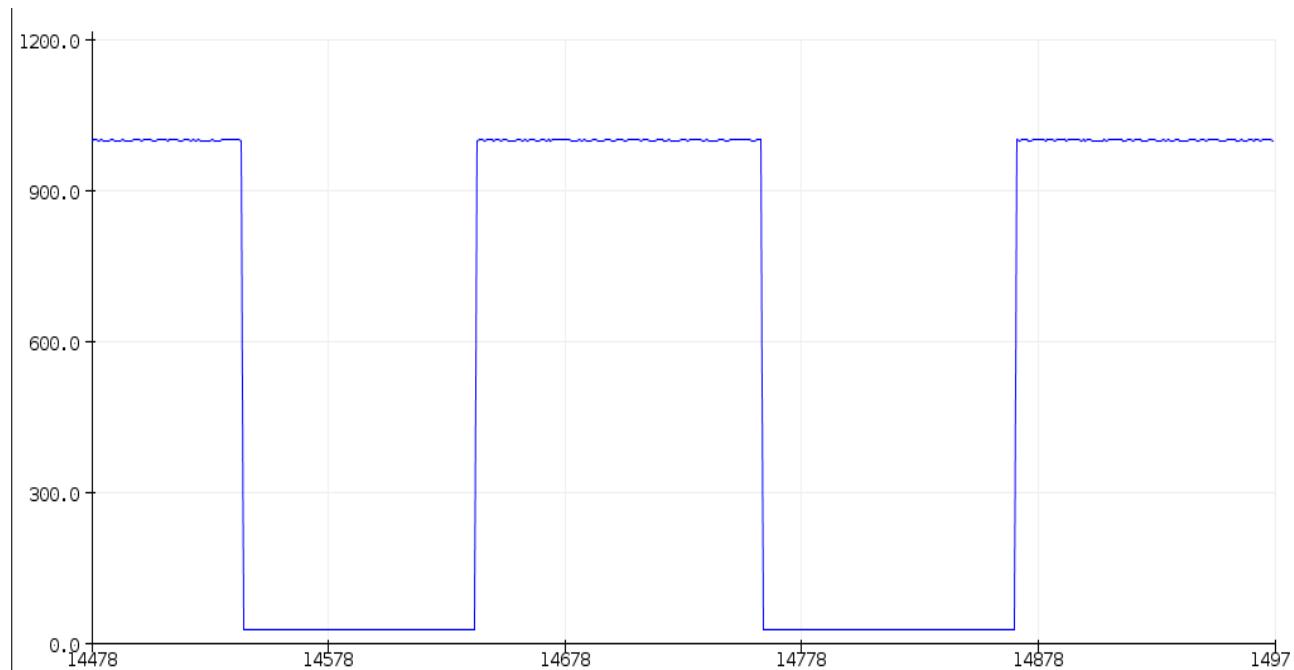
https://win.adrirobot.it/datasheet/speciali/pdf/OH44E-e_sensore-hall.pdf



Typical Working Mode



Nanjing Ouzhuo Technology co., Ltd

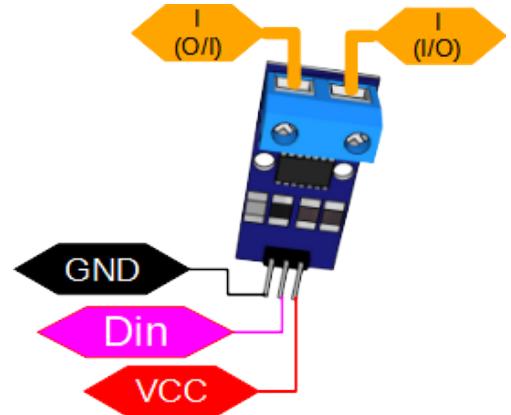




Medida de corrente con ACS712

Outra das aplicacións dos sensores de efecto Hall e a posibilidade de medir a corrente que pasa por un fío, xa que cando por un fío pasa unha corrente eléctrica esta xera un campo magnético, que pode ser detectado polo sensor ACS712, Presentase en tres versións:

| Modelo | Rango | Sensibilidade |
|-------------------|------------------|---------------|
| ACS712ELCTR-05B-T | $\pm 5\text{A}$ | 185mV/A |
| ACS712ELCTR-20A-T | $\pm 20\text{A}$ | 100mV/A |
| ACS712ELCTR-30A-T | $\pm 30\text{A}$ | 66mV/A |



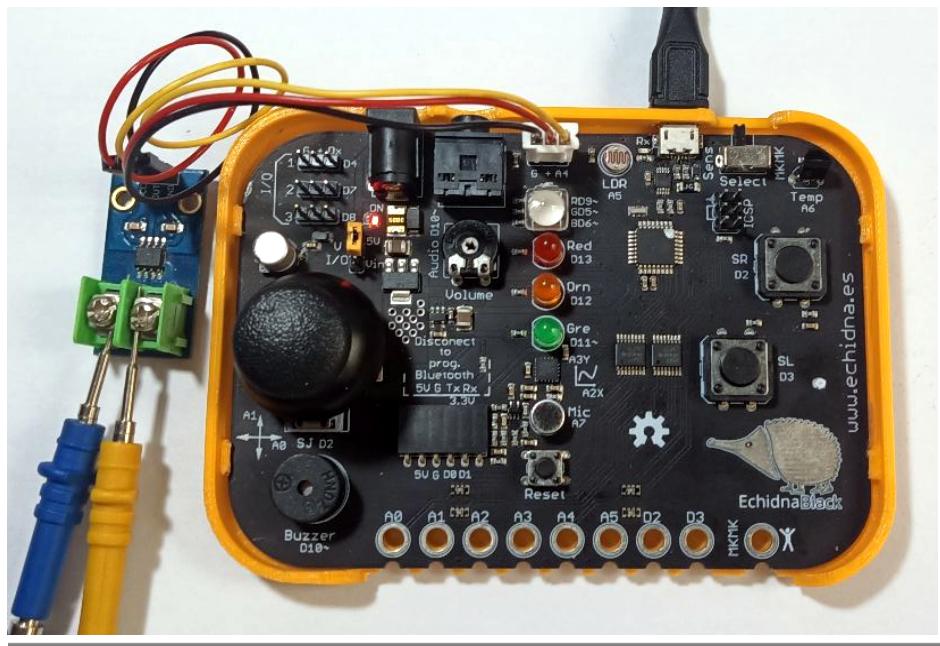
```
/*Med_Intensidade_ACS712_30A
 * Medida da intensidade usando ACS712-30A
float Sensibilidade = 0.066; //sensibilidade en
Volts/Ampères o sensor de 30A
float VSensor;      // Variable para a tensión do sensor
float I;            // Variable para o o calculo da Intensidade
float MediaI;       // Calculo da media das medidas
int n = 500;         // Nº de mostras para a media
void setup() {
  Serial.begin(9600);
}
void loop() {
  media();           //Chama a medida e calculo de media
  Serial.print("Tensión: ");
  Serial.print(VSensor);
  Serial.print("V Corrente: ");
  Serial.print(MediaI, 2);
  Serial.print(" A");
}
void media() {
  for (int i = 0; i < n; i++) { //Bucle de n medidas
    VSensor = analogRead(A4) * (4.996 / 1023.0); //lectura da tensión do sensor
    I = (VSensor - 2.5) / Sensibilidade; //Ecuación para obter a corrente
    MediaI = MediaI + I; //acumula as medidas
    delay(1);
  }
  MediaI = (MediaI / n); // Calcula a media das n medidas
}
```



Analise:

Temos que Axustar a “Sensibilidade” o ACS712 que teñamos. Medimos a tensión de saída tendo en conta que o ACS712 presenta unha tensión de “offset” de 2,50V xa que nos permite medir corrente en dous sentidos, ou corrente alterna, o cando a saída supere os 2,50V será corrente nun sentido, e si baixa dos 2,50V será no sentido contrario.

É importante ter unha referencia de tensión de precisión, a referencia de tensión normal de 5V que temos non é moi boa, no caso de esta placa mediuse a tensión en 4.996V.



Podemos ver tres medidas, a primeira de 1A a segunda de 0A e a terceira de -1A

```
Saida Monitor Serial x
Mensagem (Ctrl + Enter para enviar mensagem 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: -1.00 A
Tensión: 2.432V Corrente: -1.00 A
```



Visualizador de catro díxitos. TM1637

O TM1637 é un circuíto integrado que permite controlar seis díxitos e un teclado de dezaseis teclas, neste módulo ten catro díxitos que podemos controlar mediante dúas liñas “CLK” e “DIO” que é a liña de entrada e saída de datos, precisamos tamén a alimentación “5V” e GND.

Para facer fácil o control precisamos ter instalada a librería



TM1637Display.h, a instalaremos no Arduino IDE como sempre:

Poñendo no filtro “TM1637”





TM1637_Temperatura

```
/* TM1637_Temperatura
Usamos o Visualizador de catro díxitos controlado
por o IC TM1637 da empresa Titan Micro Electronics
Mediante os os pulsadores SR e SL,subimos ou baixamos o brillo
*/
#include <TM1637Display.h>

//Conexións do módulo de catro Dixitos
#define CLK 8 //Sinal de reloxo
#define DIO 7 //Sinal de datos

TM1637Display display(CLK, DIO); //pasamos os valores a función

#define SR 2 //Pulsador para subir o brillo
#define SL 3 //Pulsador baixar o brillo
#define LM35Pin A6 //Pin al que conectamos sensor temperatura

uint8_t brillo = 1; //Variable para almacenar o brillo
double temperatura; //Variable para almacenar a temperatura medida float
double enteira; //Parte enteira da medida de temperatura

const uint8_t celsius[] = { //Matriz para almacenar os caracteres especiais
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 a referencia analólica a 1.1V
(AtMega328)
display.clear(); //limpamos os datos do visualizador
delay(100);
}

void loop() {

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



```

if (digitalRead (SR) == 1 ) { //Comprobamos si SR esta pulsado
    brillo = brillo + 1; //Sube o brillo
    delay(200);
    if (brillo > 7) { //Evitamos superar o valor 7 do brillo
        brillo = 7;
    }
}
if (digitalRead (SL) == 1 ) { //Comprobamos si SL esta pulsado
    brillo = brillo - 1; //Baixa o brillo
    delay(200);
    if (brillo < 1) { //Evitamos baixar de 0 do brillo
        brillo = 0;
    }
}
display.setBrightness(brillo); //Enviamos o valor do brillo

double decimal = modf (temperatura, &enteira);

//Si a parte decimal supera o 0,5º aumentamos en unha unidade a parte enteira
if (decimal > 0.5) {
enteira++;
}

display.showNumberDec(enteira, false, 2, 0); //Envía a parte enteira
display.setSegments(celsius, 2, 2); //Envía os segmentos almacenados

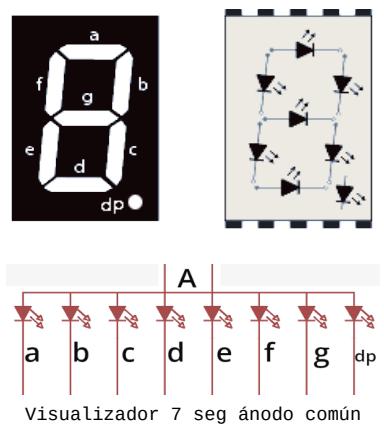
delay(500); // retardo de 500 milisegundos
}

```

💡 Analise:

Estes visualizadores reciben o nome de “visualizadores de sete segmentos”, están compostos por sete LEDes e algún punto decimal, xeralmente compartindo nun fío o ánodo ou cátodo común a todos los LEDes, no módulo que temos entre mans non temos que preocuparnos por estas conexións xa que o circuito integrado encargase de multiplexar todos los sinais, nós so temos que enviar os datos que queremos presentar.

Para axustar o brillo usamos dous parámetros : `display.setBrightness (brillo, activado);` brillo vai de 0 o máis





baixo a 7 mais alto, se o queremos apagar, usamos `display.setBrightness(brillo, false);` ou `display.setBrightness(brillo, 0);`, se non enviamos o segundo parámetro considerase “true” ou “1”.

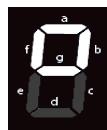
Para mostrar números usase `display.showNumberDec(número, ceros a esquerda, lonxitude, posición);` onde o primeiro parámetro é o número a mostrar, o resto dos parámetros son opcionais, o segundo é si queremos reencher con ceros a esquerda, o terceiro é o número de díxitos a mostrar e o último é a posición no visualizador contando dende a esquerda (0 a 3).

Se queremos mostrar os puntos que ten o visualizador, podemos usar a versión expandida da función anterior `display.showNumberDecEx (número , puntos , ceros a esquerda, lonxitude , posición)` exemplo:

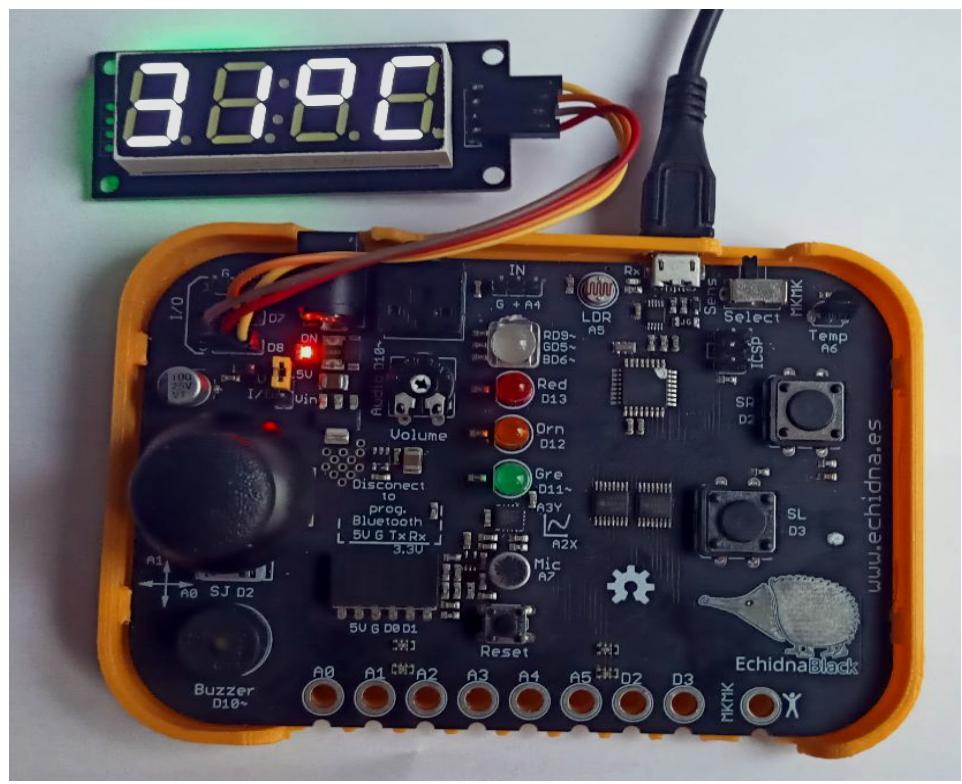
```
display.showNumberDecEx(2130, 0b01110000, false, 4, 0)
```

Para mostrar segmentos individuais usamos `display.setSegments(segmentos[], lonxitude, posición)`, o primeiro parámetro é a matriz que contén información dos segmentos a iluminar, o segundo e terceiro parámetros son os mesmos que no caso anterior.

A matriz que contén os dous caracteres “o” e “C”, para definir o que queremos visualizar temos varias formas de pasar os datos:



“o” En binario é 0b01100011 [l, g, f, e, d, c, b, a], en decimal é 99, en hexadecimal é 0x63, pero é más cómodo pasar só os segmentos que queremos iluminar “ SEG_A | SEG_B | SEG_G | SEG_F”





NeoPixel WS2812B

O Neopixel ou LED direcional consta de tres LEDes RGB e un microprocesador na mesma cápsula 5050, ten catro conexións Vcc, Gnd, Datos de entrada e Datos de saída o que nos permite conectarlos uns con outros, manexando todos mediante un único fío de datos a unha velocidade de 800Kbps a unha distancia máxima de 5 metros, sen ter que usar ningún acondicionador de sinal.

Podemos axustar a cor independentemente, chegando a 16 777 216 cores por pixel, Podemos a librería NeoPixel de Adafruit, ou a librería FastLED de Daniel Garcia.

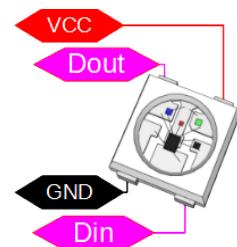
No seguinte exemplo só usamos un NeoPixel para mostrar o sinxelo que é controlalo

```
/* NeoPixel_I
   Cores o chou nun Neopixel
*/
#include <Adafruit_NeoPixel.h> //Libreria
#define PIN A4 //Pin conectado o NeoPixel
#define NUMPIXELS 1 //Número de NeoPixel
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

#define Retardo 100 //Tempo de actualización

void setup() {
  pixels.begin(); //Inicializa o NeoPixel
  pixels.clear(); //Limpa o NeoPixel
}

void loop() {
//Eliximos cores o chou
  int Red = random(64);
  int Gre = random(64);
  int Blu = random(64);
  pixels.setPixelColor(0, pixels.Color(Red, Gre, Blu)); //Nº NeoPixel e cor
  pixels.show(); //Envia os datos o NeoPixel
  delay(Retardo); //Pausa entre envíos
}
```





Vumetro con tira de NeoPixel WS2812B

Imos facer un vúmetro usando unha pequena tira de 10 NeoPixels.

```
* NeoPix_Vu_linea
Vúmetro dunha tira de 10 NeoPixels
*/
#include <Adafruit_NeoPixel.h> //librería
#define PIN A4 //Pin onde se conecta a tira de NeoPixels
#define NUM_LEDS 10 //Número de pixeles
#define Mic A7 //Entrada conectada o micrófono

int Son = 0; //Variable de lectura do son
word amp = 35; //factor de amplificación

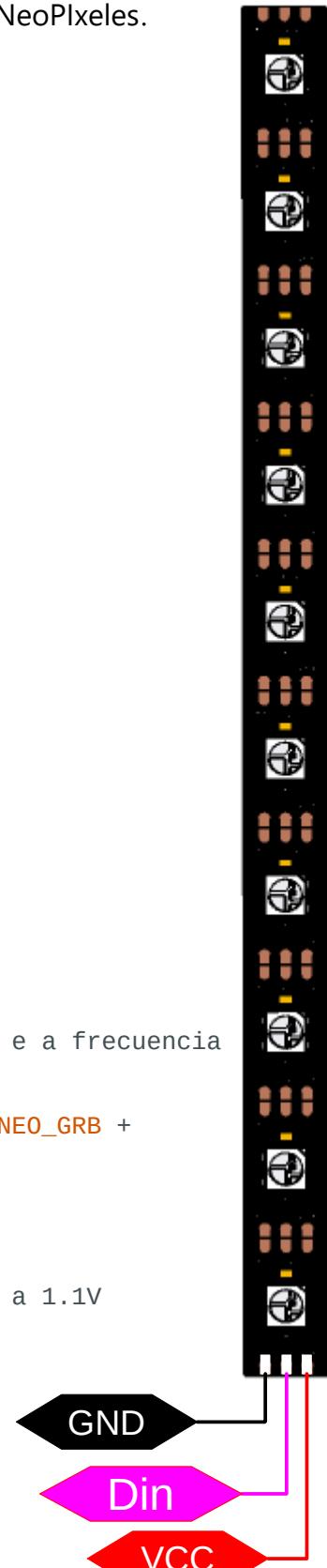
word mostras = 20; //numero de mostras (máx. 255)

double mSon = 0; //variable de mostras de son

int A = 0; //Variable de amplitud do son.
word retardo = 15; //numero de mostras (máx. 255)

word Red; //Variables para as cores
word Gre;
word Blu;
word Y; //variable exixo Y
//Crea o obxecto co nº de LEDes, o pin, a secuencia de cores e a frecuencia
de comunicación
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUM_LEDS, PIN, NEO_GRB +
NEO_KHZ800);

void setup() {
analogReference(INTERNAL); //Axustamos a referencia interna a 1.1V
pixels.begin(); //Inicia a comunicación con NeoPixels
pixels.clear(); //Borra todos pixeles
pixels.show(); //Envía os datos
}
void loop() {
//Media aritmética da entrada de son
for (int i = 0; i < mostras; i++) {
```





```

Son = analogRead(Mic) * amp;
if (Son > 1023) { //comproba que non superamos o máximo
    Son = 1023;
}
mSon = mSon + Son;
}
mSon = mSon / mostras; //Realiza a media aritmética

// mapea o valor do son o nº de leds
A = map(mSon, 0, 1023, 0, NUM_LEDS );
//Axuste da cor en función dos NeoPixelles accesos entre cero e o valor máximo
for (Y = 0; Y <= A ; Y++) {
    if (Y >= (NUM_LEDS * 4 / 5 )) {
        // 4/5 do total de NeoPixelles en vermello
        Red = 16;
        Gre = 0;
        Blu = 0;
    }
    if (Y > (NUM_LEDS / 2) and Y < (NUM_LEDS * 4 / 5)) {
        //NeoPixelles en laranxa.
        Red = 16;           /
        Gre = 8;
        Blu = 0;
    }
    if (Y < (NUM_LEDS / 2)) { //A metade de NeoPixelles en verde
        Red = 0;
        Gre = 8;
        Blu = 0;
    }
    pixels.setPixelColor(Y, pixels.Color(Red, Gre, Blu));
}
//Borra pixels superiores entre o valor máximo e o total de NeoPixelles
for (Y = A + 1; Y < NUM_LEDS; Y++) {
pixels.setPixelColor(Y, pixels.Color(0, 0, 0));
}
pixels.show(); //Envía os datos aos NeoPixelles
delay(retardo); //Pequeno retardo para deixar ver os Neopixelles
}

```



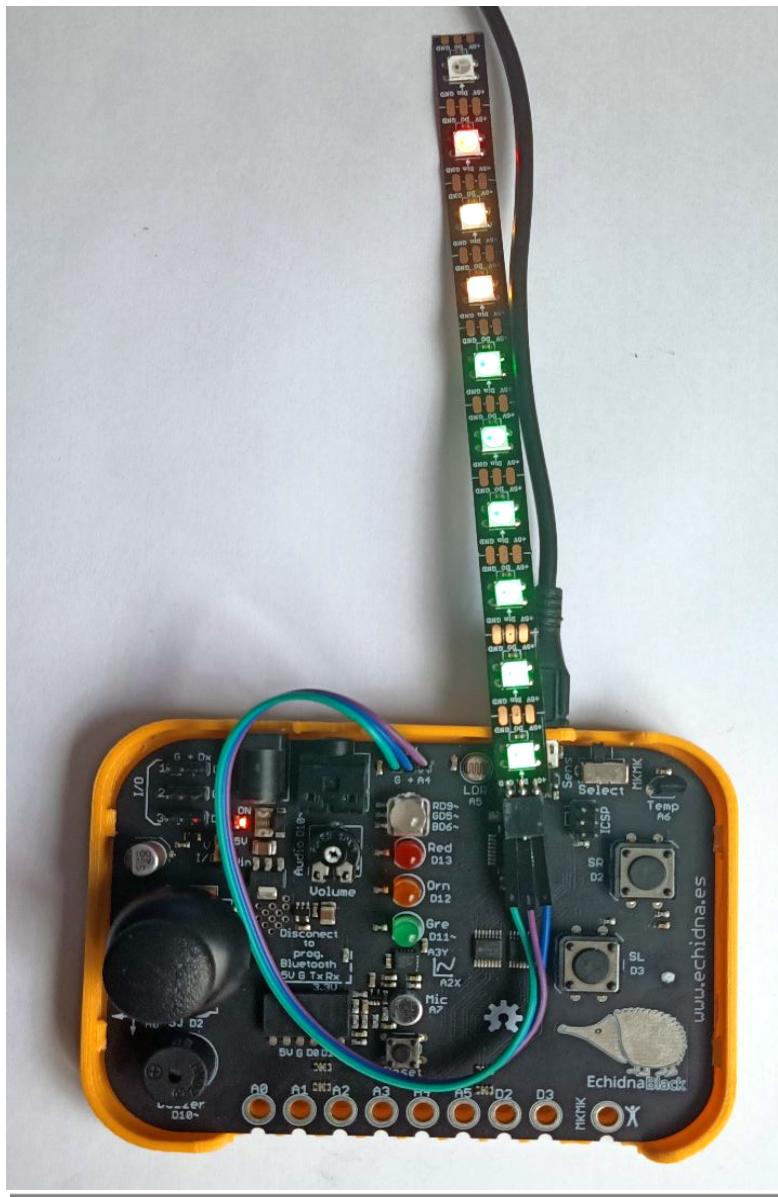
Analise:

A parte das definicións de pinos e variables, pasamos a usar a referencia interna de 1,1V. Realizamos unha comprobación para que o sinal de son multiplicado por amp non supere o máximo (1023). Tomamos mostras do sinal do micrófono para facer a media dos valores (canta máis mostras máis lento).

Mapea a escala máxima o numero total de NeoPixelles, comprobamos que NeoPixelles acendemos para elixir a cor que queremos en cada parte.

Acende os NeoPixelles entre cero e o máximo nese momento, apagando entre o máximo e o total de neoPixelles.

Finalmente temos unha pequena espera para poder ver cada ciclo de acendido dos NeoPixelles.

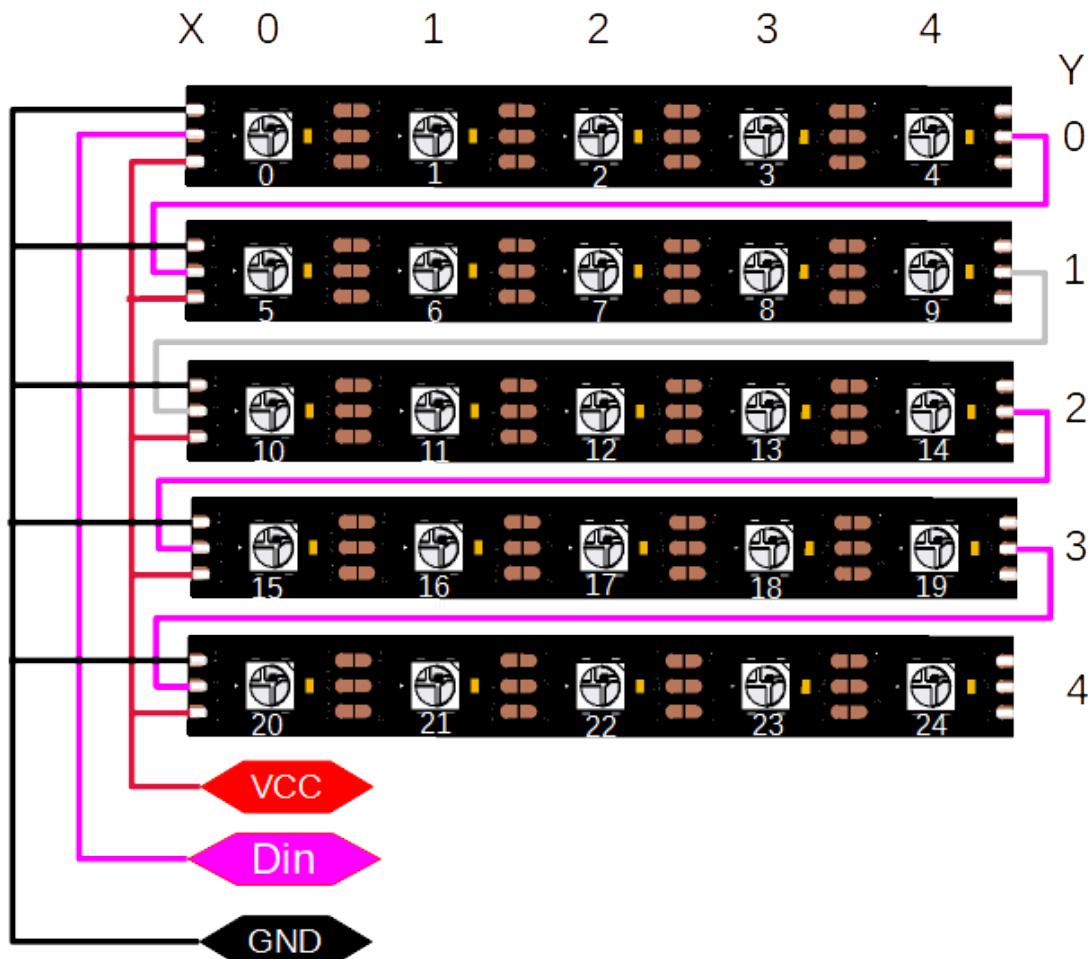




Pantalla 5 x 5 NeoPixel WS2812B

Xa podemos facer unha pequena pantalla usando cinco tiras de cinco NeoPixels ou unha PCB con NeoPixels pequenos 2x2mm.

Comenzamos conectando a entrada de datos no NeoPixel Superior esquerda, ese será a posición cero "0" desprazase a esquerda ata a posición catro "4", a seguinte tira terá as posiciones 5..9, a seguinte 10..14, ata a aposición 24, onde podemos conectar outras tiras ou outras pantallas.





Para manexar esta pantalla mediante coordenadas X, Y usamos a expresión: $X, Y = X+Y*5$ (caligari) si queremos iluminar o led na posición x3, y1 teremos que acender o NeoPixel = 3 + 5 = 8.

Como exemplo imos adaptar o programa anterior “NeoPix_Vu_linea” a pantalla, de cinco columnas e cinco filas.

```
/* NeoPix_Vu_linea_Pant
 * Visualizador da amplitude de audio de cinco bandas
 * Usando a pantalla de 5 x 5 Neopixels
 * "https://github.com/xdesig/5x5_NeoPixel_2020"
 */

#include <Adafruit_NeoPixel.h>
#define PIN_NEO A4

word Fil = 5;      //Filas
word Col = 5;      //columnas

int NUM_LEDS = Col * Fil;

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

#define MIC A7          // Entrada de Audio (Microfono)
#define SR 2            // Pulsador para subir a amplificacion
#define SL 3            // Pulsador para bajar a amplificacion

int Amp = 35;        // Valor amplificacion
int Son;              // Volume de audio
int A;                // Amplitude en leds

int Y;                // Variable de resultados (eixo Y) mapeados para mostrar na pantalla
int X;                // variable para recorer o eixo x
int Pix;              // Variable de pixel a mostrar (pix = x + y*5)
int Retardo = 10;     // Retardo en milisegundos entre ciclos

int Red;              // Variable da cor vermella
int Gre;              // Variable da cor verde
int Blu;              // Variable da cor azul
```





```

void setup() {
    analogReference(INTERNAL); //Ref. do conversor analóxico/dixital a 1.1Volts
    pinMode(SR, INPUT); // Definimos o rol de cada pin
    pinMode(SL, INPUT);
    pixels.begin(); // Inicializamos os NeoPixelles
}

void loop() {
    Red = random (32); //eliximos as cores de cada Ciclo "colores o chou"
    Gre = random (32);
    Blu = random (16);
    for (X = 0; X < Col; X++) { // Percorremos a pantalla no eixo X
        // Lemos a entrada de son e a multiplicamos por Amp
        Son = (analogRead(MIC) * Amp);
        if (Son>=1023){ // comprobamos que non superamos o valor máximo
            Son = 1023;
        }
        // Mapeamos os valores de amplitude (filas) e os invertemos
        // para iluminalos dende abaixo arriba
        A = map(Son, 0, 1023, Fil, 0);

        for (Y = Fil; Y >= A ; Y--) { // Acendemos os NeoPixelles Verticais
            Pix = X + Y * Col; // Pasamos as coordenadas X e Y a linea de pxeles
            // Acendemos cada pixel ca sua cor
            pixels.setPixelColor(Pix, pixels.Color(Red, Gre, Blu));
        }
        // Apagando os pxeles superiores o valor máximo medido de cada columna
        // Usamos "A-1" para non apagar os Neopixelles inferiores
        for (int Y = 0; Y < A-1; Y++) {
            Pix = X + Y * Col;
            pixels.setPixelColor(Pix, pixels.Color(0, 0, 0));
        }

        pixels.show(); // Enviamos os datos a pantalla de NeoPixelles
    }
    delay(Retardo); // un pequeno retardo entre ciclos
}

```



Analise:

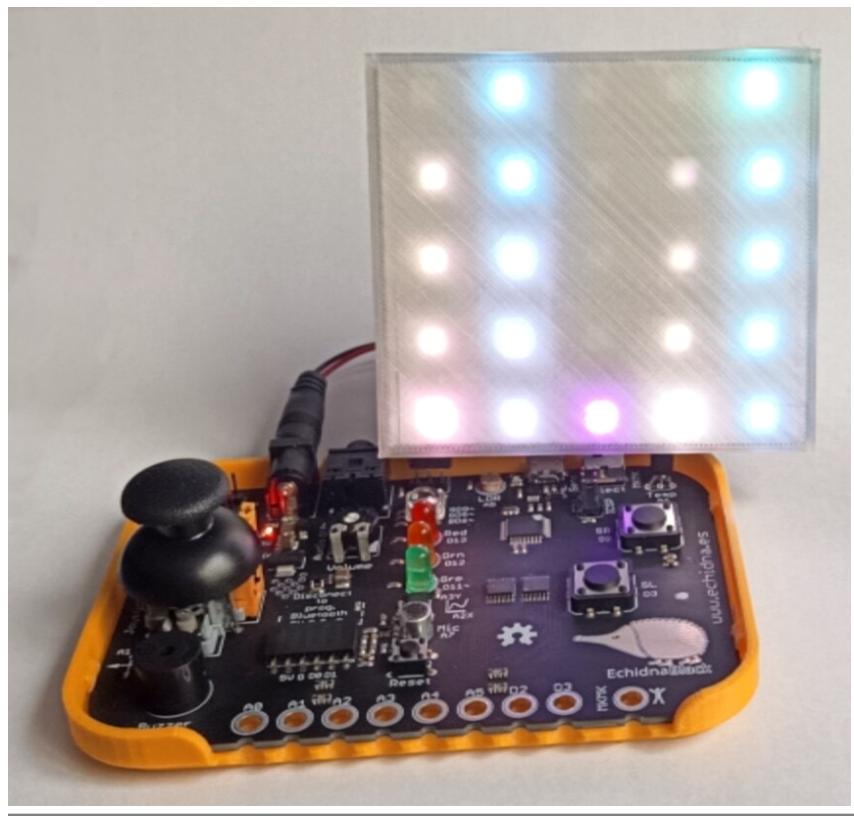
Para mostrar os datos na pantalla a percorremos de esquerda a dereita con “`for (X = 0; X < Col; X++)`”, facendo que o eixo X da pantalla sexa o eixo de tempo.

Seguidamente collemos o valor detectado polo micrófono, unha vez multiplicado por “Amp”.

Temos en conta que que os NeoPixelles de abaxo teñen posicións más altas que os de arriba por iso:

“`A = map(Son, 0, 1023, Fil, 0);`” o resultado está invertido, xa que queremos que se iluminen dende abaxo a arriba, con “`for (Y = Fil; Y >= A ; Y--)`” percorremos as filas para acender os NeoPixelles,

Para apagar os non usados “`for (int Y = 0; Y < A-1; Y++)`”, sempre seguido da expresión. “`Pix = X + Y * Col;`” por ultimo usamos un pequeno retardo para mostrar os NeoPixelles accesos durante más tempo (podemos baixalo se precisamos rapidez).





Analizador de espectro 5 x 5 NeoPixel WS2812B

Segundo coa a pantalla podemos facer un [analizador de espectro](#). Precisamos instalar a librería “fix_fft.h”.

```
/* Neopix_Espectro
 * Visualizador de espectro de son de cinco bandas
 * Usando a pantalla de 25 Neopixeles
 * Transformada rápida de Fourier (FFT). Autor:
Dimitrios P. Bouras, mantida
 * por Enrique Condes
 * XDeSIG 2021
*/
#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 FFT
char re[128], im[128]; // Matrices de mostras / resultados da FFT
byte i; // índice
int dat; // variable da amplitudade da 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 amplificación
int Y; // Variable de resultados (eixo Y) mapeados para mostrar na pantalla
int X; // variable para recorrer o eixo x
int A; // Amplitudade en NeoPixelles
int Pix; // Variable de pixel a mostrar (pix = x + y*5)
int Red; // Variable da cor vermella
int Gre; // Variable da cor verde
int Blu; // Variable da cor azul

void setup() {
analogReference(INTERNAL); // Ref. Analólica/dixital a 1.1Volts
pixels.begin(); // Inicializamos os NeoPixelles
}
```

GERENCIADOR DE BIBLIOTECAS

fft

Type: All

Topic: All

arduinoFFT por Enrique Condes
<enrique@shapeoko.com>
With this library you can calculate the frequency of a sampled signal.
A library for implementing floating point Fast Fourier Transform calculations on Arduino.

[Mais informações](#)

1.5.6

INSTALAR



```
void loop() {
    // Lectura de son e calculo da amplitud por frecuencia.
    for (i = 0; i < 128; i++) { // Tomamos 128 mostras da entrada analóxica
        (micro)
        int sample = amp * (analogRead(MIC));
        re[i] = sample / 4 - 128; //Escalamos para axustar a un carácter de -128 a
        127
        im[i] = 0; //Agora non temos valores imaxinarios e facemos todos iguais a
        cero
    }
    fix_fft(re, im, 7, 0); // Enviamos as mostras para a conversión FFT,
                           // devolvendo os resultados reais/imaxinarios nas mesmas matrices
    for (X = 0; X < 5; X++) { // Recorremos o eixo X da pantalla
        dat = sqrt(re[F[X]] * re[F[X]] + im[F[X]] * im[F[X]]); // A magnitud da
        // frecuencia é a raíz cadrada da suma dos cadrados das partes real e
        imaxinaria // dos vectores. Só recuperamos os valores das frecuencias elixidas
        F[x]
        // Mapeamos os valores de amplitud de cada frecuencia os píxeles verticais
        A = map(dat, 0, 25, 4, 0);

        for (Y = 5; Y >= A ; Y--) { // Acendemos os NeoPixelles Verticais
            Pix = X + Y * 5; // Calcula as coordenadas X e Y a liña de píxeles.
            if (Y == 0) { // Axustamos a cor a cada liña
                Red = 32;
                Gre = 0;
                Blu = 0;
            }
            if (Y == 1) {
                Red = 32;
                Gre = 16;
                Blu = 0;
            }
            if (Y == 2) {
                Red = 16;
                Gre = 8;
                Blu = 0;
            }
            if (Y == 3) {
                Red = 4;
                Gre = 16;
```



```

Blu = 0;
}
if (Y == 4) {
    Red = 0;
    Gre = 4;
    Blu = 2;
}
pixels.setPixelColor(Pix, pixels.color(Red, Gre, Blu));
}
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 os datos a pantalla de NeoPixel
}
}

```

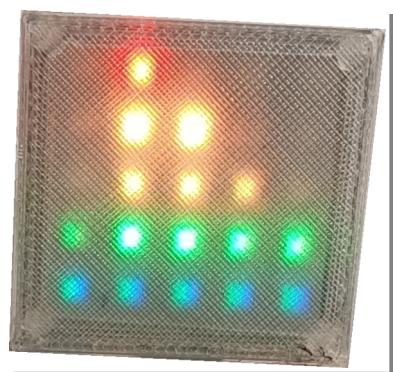
Analise:

O analizador de espectro de son, onde o eixo X está no dominio da frecuencia e o eixo Y representa a amplitude de cada frecuencia analizada. A librería “fix_fft.h” usase para calcular a [Transformada rápida de Fourier](#) e así descompoñer cada grupo de mostras na súas compoñentes espectrais (frecuencias) .

Tomamos 128 mostras do son que depositamos na matriz “re[128]”, unha vez realizado o cálculo terá os resultados reais e “im[128]” terá os resultados imaxinarios, para máis tarde realizar a raíz cadrada dos cadrados dos resultados reais mais as partes imaxinarias o cadrado, isto o facemos para cinco valores, xa que só temos cinco columnas na nosa pantalla. Podemos elixir que frecuencia corresponde a cada columna na matriz $F[5] = \{2, 3, 6, 9, 15\}$; cada nº o multiplicaremos por 73,8Hz.

No exemplo temos:

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



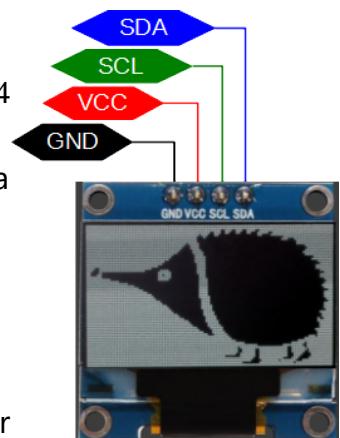


Pantalla Oled bus I²C

Usaremos a pantalla SSD1306 de 0,96" (2,43cm) de 128 x 64 píxeles.

É una pantalla de baixo custo, ten as vantaxes que presenta a tecnoloxía OLED e un bus de comunicacións con só dous pinos:

SDA e SCL.



Pero que é OLED? :

OLED é o acrónimo de Organic Light-Emitting Diode, díodo emisor de luz composto por materiais orgánicos electroluminiscentes.

Descubertos por Heeger, MacDarmid e Shirakawa, galardoados co premio Nobel de química no año 2000.

Producen unha gran luminosidade xa que son muy delgados, pódense fabricar en grandes formatos, no precisan iluminación externa, consumen menos que outras pantallas, ofrecen una amplio ángulo de visión, en torno os 170 grados, teñen unha longa vida de funcionamento entre 10000 a 40000 horas, (os azulis duran menos), teñen tempos de resposta menores, e un gran contraste, pola contra son sensibles a humidade que os degrada rapidamente.

Podemos consultar máis información en:

“Fundamentos de la Tecnología OLED, editado por P. CHAMORRO POSADA, J. MARTÍN GIL, P. MARTÍN RAMOS, L. M. NAVAS GRACIA en la Universidad de Valladolid.”

Conexións:

(Ollo algunas versións de pantalla cambian de sitio os pinos)

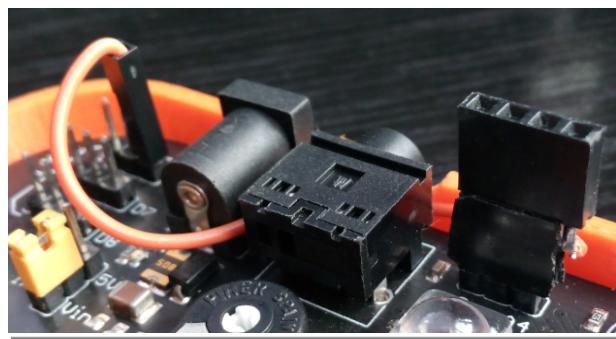
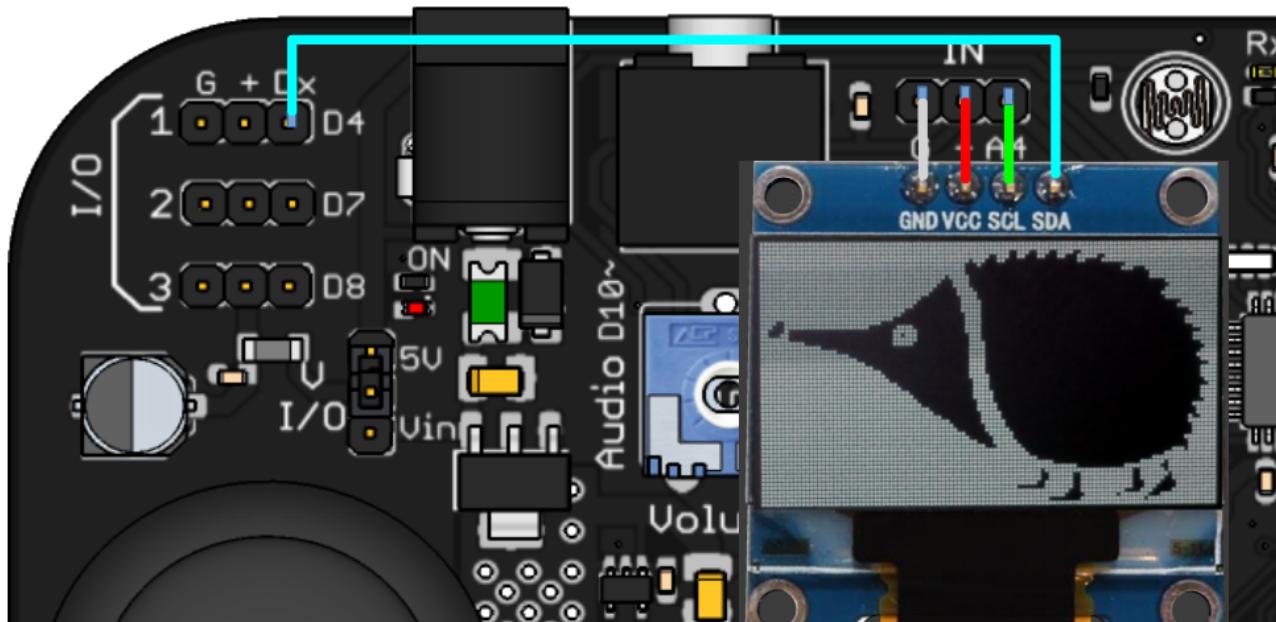
| Pantalla OLED | Echidna |
|--------------------|---------|
| GND | GND |
| VCC | VCC |
| SCL (System Clock) | A4 |
| SDA (System Data) | D4 |





No conector IN da EchidnaBlack só dispoñemos de GND, VCC e A4, obríganos a levar esa conexión cun pequeno cable a (I/O1) D4...

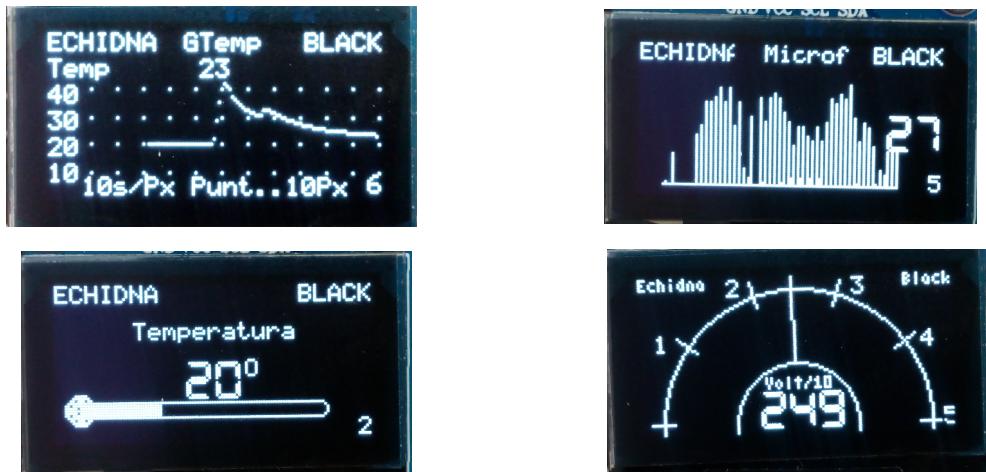
Temos a posibilidade de conectar a pantalla nos pinos D4, D7 ou D8, pero no conector IN a pantalla queda centrada. :-)





Pasemos a programación:

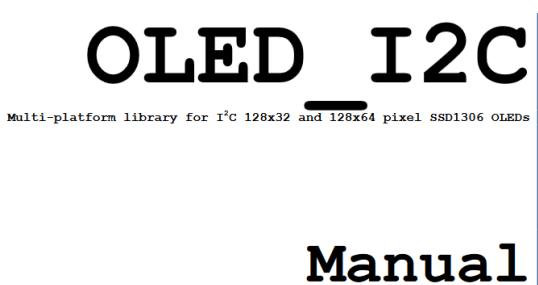
Precisamos unha librería que nos permita controlar a pantalla, cambiando os pinos de comunicacións. Podemos usar a librería “OLED_I2C.h” 2019 de Henning Karlsen con licencia CC BY-NC-SA 3.0. é una librería moi sinxela, non conta cas soluciones gráficas da librería de Adafruit “Adafruit_GFX.h”. cun un pouco de imaxinación podemos realizar algunas gráficas interesantes.



Librería:

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

Unha vez baixada a instalaremos con [Sketch] [incluir librería] [Engadir librería. ZIP...]





Oled_Cadro_JOY

```

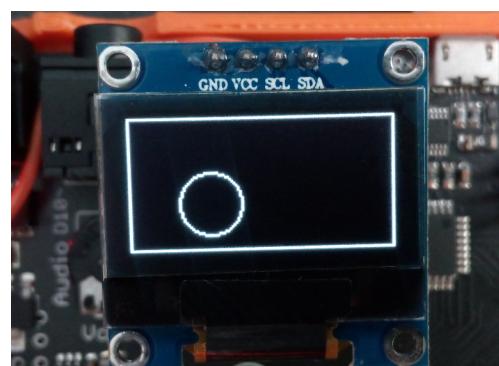
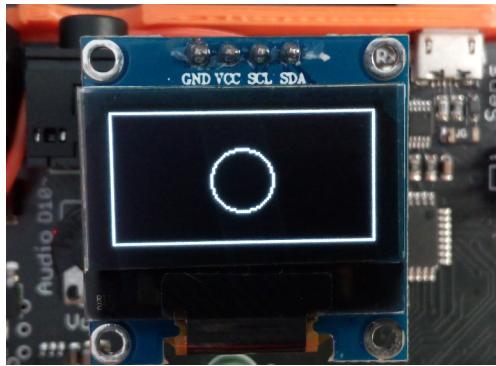
/* Oled_Cadro_JOY
 * Debuxa un cadro límite do joystick
 */

#include <OLED_I2C.h> //((C)Rinky-Dink Electronics
OLED myOLED(4, A4); //establece os pines de comunicación I2C OLED A4 e D4
//***** Establece as entradas da sinal a medir *****
#define EntradaX A0
#define EntradaY A1
//***** Establece os valores mínimos e máximos
const int VXmin = 0;
const int VXmax = 1023;
const int VYmin = 0;
const int VYmax = 1023;
const int Radio = 15; //Radio da circunferencia
int PosX = 64; //coordenada centro X
int PosY = 32; //coordenada centro Y
int MedidaX; //variable para a lectura do valor X
int MedidaY; //variable para a lectura do valor Y
int i; // Variable común
void setup()
{
    myOLED.begin(); //inicializa o visualizador OLED 128x64
    Serial.begin(115200); // Inicializa a comunicación serie
}
void loop () {
    //***** Lee a entrada analólica *****
    int MedidaX = analogRead(EntradaX);
    int MedidaY = analogRead(EntradaY);
    //***** Envía o valor vía serie *****
    Serial.print (MedidaX);
    Serial.print ("\t");
    Serial.println (MedidaY);
    //***** Escalamos os valores X e Y o tamaño da pantalla
    PosX = map( MedidaX, VXmin, VXmax, 0+Radio , 128-Radio );
    PosY = map( MedidaY, VYmin, VYmax, 64-Radio , 0+Radio );
    //***** Debuxa o rectángulo *****
}

```



```
myOLED.drawRect(0, 0, 127, 63); // coordenadas inicio x,y fin x,y  
//***** Debuxa o circulo na posición dada polo joystick *****  
myOLED.drawCircle( PosX, PosY, Radio);  
//***** Presenta toda a información que ten na memoria OLED **  
myOLED.update();  
delay(1);  
//***** Borra o visualizador *****  
myOLED.clrScr();  
}
```



💡 Analise:

No escalado (mapeado) das posicóns X e Y temos en conta o radio da circunferencia para que a circunferencia non saia do cadro.

Debuxar con esta librería é moi doado, aquí só usamos:

myOLED.drawRect (iniX, iniY, finX, finY) para o rectangulo
myOLED.drawCircle(PosX, PosY, Radio) para o círculo

O resto de comandos son:

OLED myOLED(4, A4) establece os pines de comunicación I2C OLED A4 e D4
myOLED.begin() inicializa o visualizador OLED
myOLED.update() copia o contido da memoria na pantalla.
myOLED.clrScr() borra a memoria



Oled_Medidor_Analog

```

/* oled\_Medidor\_Analog
* Medidor analáxico de agulla 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 os pines de comunicacóns I2C OLED
#define Entrada A5 // A entrada de sinal a medir
extern uint8_t TinyFont[]; // Fontes de textos e números
extern uint8_t SmallFont[];
extern uint8_t MediumNumbers[];
extern uint8_t BigNumbers[];
// Modificando os seguintes parámetros pódese cambiar de
// posición e de tamaño a representación analólica.
//Radio da semicircunferencia do cuadrante (valor clave).
const int Radio = 90; // por defecto 90.
const int Radiop = 65; //radio da semicircunferencia pequena onde comenza a
agulla para evitar o Nº.(65)
const int Xcentro = 62; //coordenada centro X do cuadrante. (62)
const int Ycentro = 100; //coordenada centro Y do cuadrante. (100)
const int AngMin = 45; //Angulo máximo para a liña da escala. (45)
const int AngMax = 135; //Angulo máximo para a liña da escala. (135)
const int Nmarcas = 4; //Nº de marcas que queremos por. (4)
//*****
int dotX; // variables para os puntos da escala
int dotY;
int marXini; // variables de inicio e final das marcas no cuadrante
int marYini;
int marXfin;
int marYfin;
int Xini; // variables de inicio e final da agulla
int Yini;
int Xfin;
int Yfin;
int i; // Variable común
const float Pi = 3.1415927; //Pi para calcular ángulos de grados en radiáns.
float Alfa; // ángulo no cuadrante expresado en radiáns
float AlfaMin = (AngMin - 180) * ( Pi / 180);
float AlfaMax = (AngMax - 180) * ( Pi / 180);
// calcula o paso entre os ángulos mínimo e máximo.

```



```
float Paso = (AlfaMax - AlfaMin) / Nmarcas;
float Marca [Nmarcas]; //establece a matriz para as marcas
//ollo por os valores como máximo de "Nmarcas"
// textos a representar no cuadrante
char const *valores [] = { "", "256", "512", "768", ""};
const int despr[] { 0, 10, 0, 10, 0}; // Desprazamento X para valores
void setup(){
    myOLED.begin(); //inicializa o visualizador OLED 128x64
    //myOLED.invert(true); //inverte o display
    Serial.begin(115200); // inicializa a comunicación serie
    Serial.println("Inicializando");
    myOLED.setFont(SmallFont); // fonte de letra pequena
    myOLED.print("Iniciando", CENTER, 05);
    myOLED.update();
    // calculamos os ángulos onde representar as marcas e os valores das mesmas
    Alfa = AlfaMin;
    for (i = 0; i < (Nmarcas); i++) {
        Marca[i] = Alfa;
        Alfa = Alfa + Paso; // incremento ou decrecemento para adaptalo os textos
    }
}
void loop () {
    int Medida = analogRead(Entrada); // lee a entrada analólica
    Serial.println (Medida); // envía o valor vía serie
    myOLED.setFont(BigNumbers);
    myOLED.printNumI(Medida, CENTER, 40); //visualiza o valor da entrada analólica
    Pint_text(); // pinta o texto no cuadrante
    DibuCirc(); // debuxa o círculo do cuadrante
    PintMarcas(); // debuxa as marcas do cuadrante
    PintValores(); // debuxa os valores
    //***** Pasa a media o ángulo alfa
    int MedidaM = map(Medida, 0, 1023, 45, 135); // medida a un máx. de 180°
    Alfa = (MedidaM - 180) * ( Pi / 180); // calcula o ángulo da medida
    //***** Debuxa a agulla no cuadrante
    Xini = Radiop * cos (Alfa) + Xcentro; // calcula os puntos de inicio
    Yini = Radiop * sin (Alfa) + Ycentro;
    Xfin = Radio * cos (Alfa) + Xcentro; // calcula os puntos de fin
    Yfin = Radio * sin (Alfa) + Ycentro;
    myOLED.drawLine(Xini, Yini, Xfin, Yfin); // Debuxa a liña da agulla
```



```

myOLED.update(); // presenta toda a información que ten a memoria o OLED
delay(1);
myOLED.clrScr(); // borra o contido do visualizador
}
//***** Pintando os puntos da escala *****
void DibuCirc() {
    for (float F = AlfaMin; F < AlfaMax; F = F + 0.04) {
        dotX = (Radio - 5) * cos (F) + Xcentro; // calcula os puntos X e Y
        dotY = (Radio - 5) * sin (F) + Ycentro;
        myOLED.setPixel(dotX, dotY);
    }
}
//***** Pintando os valores *****
void PintValores() {
    for (i = 0; i < Nmarcas; i++) {
        myOLED.setFont(SmallFont);
        marXfin = (Radio + despr[a[i]] ) * cos (Marca[i]) + Xcentro - 10;
        marYfin = (Radio ) * sin (Marca[i]) + Ycentro - 5;
        myOLED.print(valores[i], marXfin, marYfin);
    }
}
//***** Debuxa as marcas do cuadrante *****
void PintMarcas() {
    for (float F = AlfaMin; F < AlfaMax; F = F + Paso) {
        // Comenza a 10 puntos menos do radio para non empastar
        marXini = (Radio - 10) * cos (F) + Xcentro;
        marYini = (Radio - 10) * sin (F) + Ycentro;
        // Finaliza a 5 puntos por riba do circulo debuxado
        marXfin = Radio * cos (F) + Xcentro;
        marYfin = Radio * sin (F) + Ycentro;
        myOLED.drawLine(marXini, marYini, marXfin, marYfin); // Debuxa as marcas
    }
}
//***** Pinta os textos Echidna      STEAM *****
void Pint_text() {
    myOLED.setFont(SmallFont); // Fonte pequena
    myOLED.print("Echidna", LEFT, 55);
    myOLED.print("STEAM", RIGHT, 55);
}

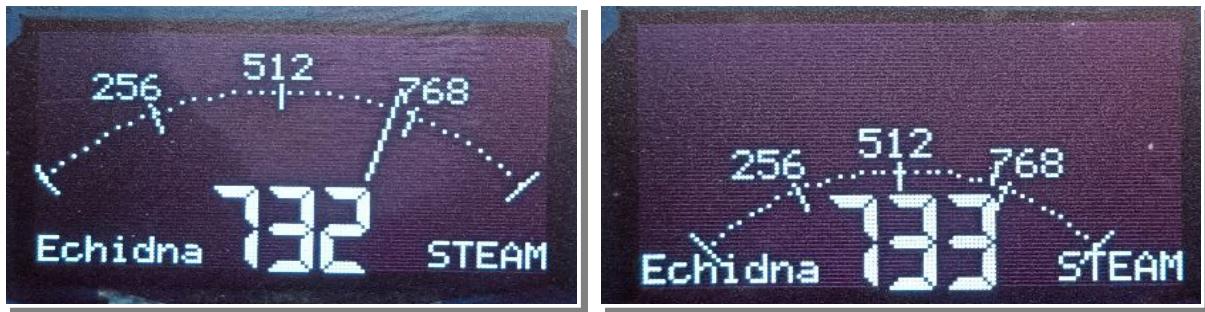
```



Analise:

Modificando os seguintes parámetros pódese cambiar de posición e de tamaño a representación analóxica: Radio, RadioP, Xcentro, Ycentro, Angmin, AngMax,Nmarcas

Por exemplo pasando o Radio = 70 deixando espacio superior para outras indicacións.



En valores [] = { "", "256", "512", "768", ""}; podemos por os nosos valores, para axustar a posición “X” de eses valores usamos a desprá[] { 0, 10, 0, 10, 0};

Para debuxar o cuadrante e o inicio/fin da agulla calculamos os ángulos en radiáns e calculamos o coseno do ángulo Alfa para a coordenada X e o seno para a coordenada Y.

Podemos descargar o programa [EchidnaOledTest](#) do repositorio, neste programa podemos ver moitas das solucións explicadas en este maual



Calculo resistencia serie “Rs”

Para calcular a resistencia serie “Rs”, teremos en conta:

Tensión que proporciona na saída (D_X).

Para este exemplo imos supoñer que o circuíto está conectado a unha das saídas do ATMega328P que está alimentado a 5Vcc, a tensión na saída para valor alto é 4,1V mínimo e 5V máximo.

Tipo de LED.

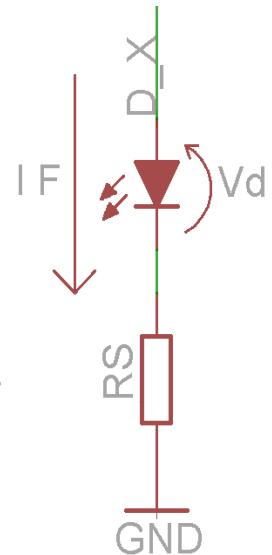
El LED será de 5mm de diámetro vermello, que ten as seguintes características. Corrente directa $I_F = 20mA$. Tensión directa $V_d = 1,9V$.

Cálculo de R_s .

A tensión que cae na resistencia será a tensión de saída do Atmega (D_X) que chamaremos V_i , menos a tensión necesaria para o funcionamento do LED V_d .

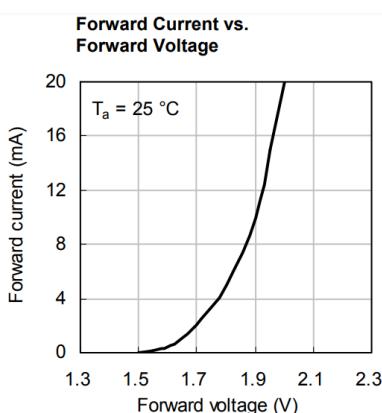
A corrente que circula pola resistencia e a mesma que a do LED I_F .

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



Para non forzar as saídas do microcontrolador e que os LEDes duren moito podemos baixar a corrente que pasa por el a uns 5mA, isto implica subir a resistencia a 620Ω onde o valor estándar é 680Ω .

Lembra que cada fabricante e tipo de LED ten uns valores diferentes de tensión V_F e corrente I_F de funcionamiento, teremos que consultar as follas características



| ELECTRICAL / OPTICAL CHARACTERISTICS at $T_A=25^\circ C$ | | | | | |
|---|-----------------------|---------------------|-------|------|----------------|
| Parameter | Symbol | Emitting Color | Value | | Unit |
| | | | Typ. | Max. | |
| Wavelength at Peak Emission $I_F = 10mA$ | λ_{peak} | High Efficiency Red | 627 | - | nm |
| Dominant Wavelength $I_F = 10mA$ | λ_{dom} [1] | High Efficiency Red | 617 | - | nm |
| Spectral Bandwidth at 50% $\Phi_{REL\ MAX}$ $I_F = 10mA$ | $\Delta\lambda$ | High Efficiency Red | 45 | - | nm |
| Capacitance | C | High Efficiency Red | 15 | - | pF |
| Forward Voltage $I_F = 10mA$ | V_F [2] | High Efficiency Red | 1.9 | 2.3 | V |
| Reverse Current ($V_R = 5V$) | I_R | High Efficiency Red | - | 10 | μA |
| Temperature Coefficient of λ_{peak} $I_F = 10mA, -10^\circ C \leq T \leq 85^\circ C$ | $TC_{\lambda_{peak}}$ | High Efficiency Red | 0.12 | - | $nm/^{\circ}C$ |
| Temperature Coefficient of λ_{dom} $I_F = 10mA, -10^\circ C \leq T \leq 85^\circ C$ | $TC_{\lambda_{dom}}$ | High Efficiency Red | 0.06 | - | $nm/^{\circ}C$ |
| Temperature Coefficient of V_F $I_F = 10mA, -10^\circ C \leq T \leq 85^\circ C$ | TC_V | High Efficiency Red | -1.9 | - | $mV/^{\circ}C$ |

Notes:

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



Desenladrillar.

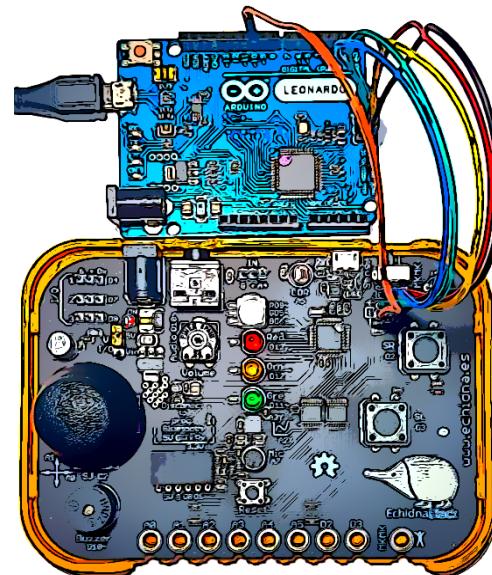
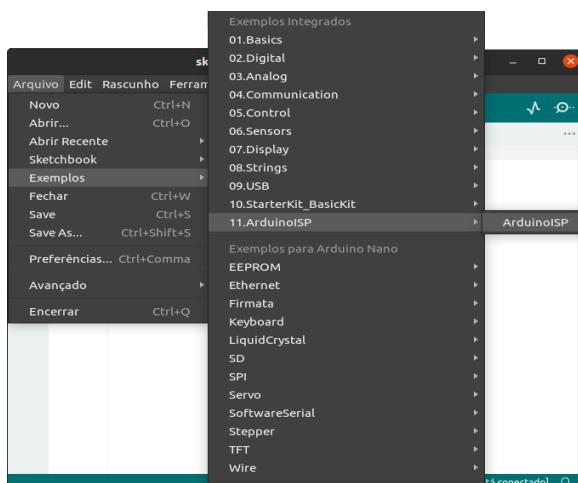
Para volver a vida a EchidnaBlack tras un “Briquedo” temos que gravar de novo o bootloader mediante un programador ICSP “In-Circuit Serial Programming”. Se non temos un programador, podemos usar un Arduino ou outra EchidnaBlack.

Imos comenzar con un Arduino Nano, Uno , Leonardo...

1. Conectamos o Arduino o ordenata, eliximos o tipo de Arduino.

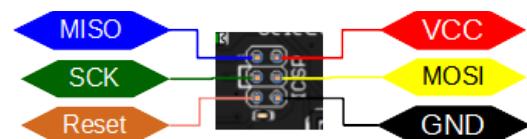
1. Subimos o Sketch (programa) que temos en:

[Ficheiro][Exemplos][ArduinoISP]



2. Realizamos as conexións entre o Arduino e EchidnaBlack:

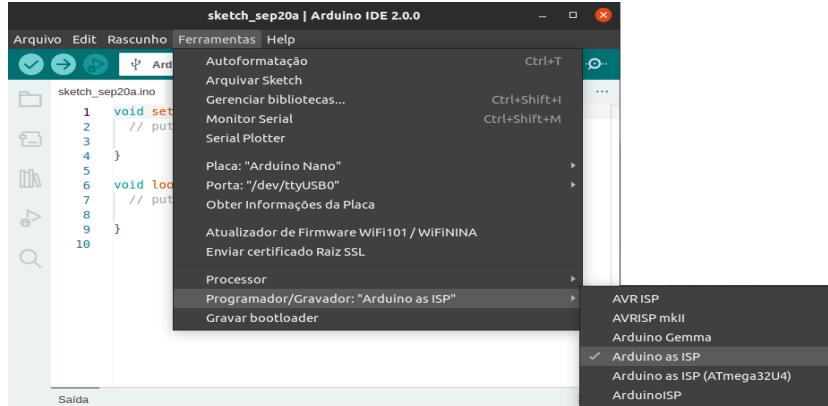
| Arduino | EchidnaBlack |
|---------|--------------|
| MISO | MISO |
| MOSI | MOSI |
| SCK | SCK |
| D10 | RESET |
| Vcc | Vcc |
| GND | GND |





1. 3. Seleccionamos como placa Arduino Nano Procesador ATmega328

4. Eliximos o programador “Arduino as ISP”



5. Xa podemos “Pinchar” en [Gravar bootloader]

Tras un minuto ou algo máis, depende do sistema operativo, xa teremos a EchidnaBlack co bootloader correcto.

Se non temos un Arduino, pero si outra EchidnaBlack, podemos convertela nun programador ISP, seguiremos os pasos anteriores con unhas pequenas modificacións das conexións e do programa.

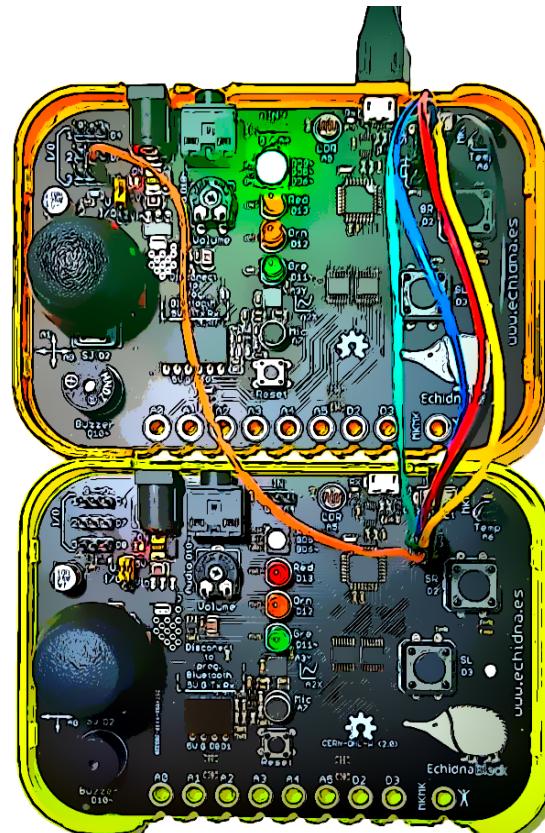
Conexións:

| Echidna Black Como programador | EchidnaBlack DESTINO |
|-----------------------------------|-------------------------|
| MISO | MISO |
| MOSI | MOSI |
| SCK | SCK |
| D10 | RESET |
| Vcc | Vcc |
| GND | GND |

Diagrama de conexións:

```

    graph LR
        subgraph "Echidna Black"
            MISO[MISO] --- ICSP[ICSP]
            SCK[SCK] --- ICSP
            Reset[Reset] --- ICSP
        end
        subgraph "EchidnaBlack DESTINO"
            VCC[VCC] --- VCC
            MOSI[MOSI] --- MOSI
            GND[GND] --- GND
        end
        MISO --> ICSP
        SCK --> ICSP
        Reset --> ICSP
        ICSP --- VCC
        ICSP --- MOSI
        ICSP --- GND
    
```





Cambios no “Sketch” ArduinolISP, nas liñas 73 a 76 :

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

The screenshot shows the Arduino IDE 2.0.0 interface with the title bar "ArduinoISP | Arduino IDE 2.0.0". The menu bar includes "Arquivo", "Edit", "Rascunho", "Ferramentas", and "Help". The toolbar has icons for save, undo, redo, and upload. The dropdown menu shows "Arduino Nano". The code editor displays the "ArduinolISP.ino" file. The code is as follows:

```
ArduinoISP.ino
b9 // The standard pin configuration.
70 #ifndef ARDUINO_HOODLOADER
71
72 #define RESET      8 // Use pin 10 to reset the target rather than SS
73 #define LED_HB      5
74 #define LED_ERR      9
75 #define LED_PMODE    6
76
77 // Uncomment following line to use the old Uno style wiring
78 // (using pin 11, 12 and 13 instead of the SPI header) on Leonardo, Due...
79
80 // #define USE_OLD_STYLE_WIRING
81
82 #ifdef USE_OLD_STYLE_WIRING
83
84     #define PIN_MOSI   11
85     #define PIN_MISO   12
86     #define PIN_SCK    13
87
88 #endif
89
90 // HOODLOADER2 means running sketches on the ATmega16U2 serial converter chip
91 // on Uno or Mega boards. We must use pins that are broken out.
92
```

The status bar at the bottom indicates "Ln 76, Col 22 UTF-8" and "Arduino Nano em /dev/ttyUSB0".

Eliximos a placa **Arduino Nano**, procesador **Atmega328P**, o porto serie correspondente, subimos ArduinolISP a EchidnaBlack (programador), unha vez subido xa podemos **Gravar bootloader** na EchidnaBlack (Destino).

The screenshot shows two windows of the Arduino IDE 2.0.0. The left window shows the "Ferramentas" (Tools) menu with options like Autoformatação, Arquivar Sketch, Gerenciar bibliotecas..., Monitor Serial, Serial Plotter, Placa: "Arduino Nano", Porta: "/dev/ttyUSB0", Obter Informações da Placa, Atualizador de Firmware WiFi101 / WiFiNINA, Enviar certificado Raiz SSL, Processor, Programador/Gravador: "Arduino as ISP", and Gravar bootloader. The right window shows the "ArduinolISP | Arduino IDE 2.0.0" interface with the "ArduinolISP.ino" file open. The code is identical to the one shown in the previous screenshot. Below the code editor is a terminal window titled "Saída" (Output) showing the bootloading process:

```
Copyright (c) 2007-2014 Joerg Wunsch

System wide configuration file is "/home/xabier/.arduino15/packages/arduino
User configuration file is "/home/xabier/.avrduudec"
User configuration file does not exist or is not a regular file, skipping

Using Port          : /dev/ttyUSB0
Using Program       :
Overriding Ba       : Burning bootloader...
```

The status bar at the bottom indicates "Ln 76, Col 22 UTF-8" and "Arduino Nano em /dev/ttyUSB0".

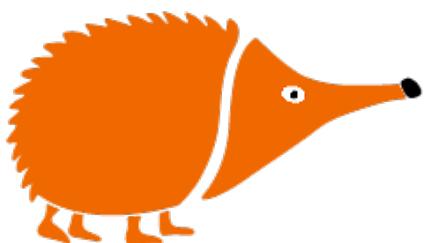


Atribución 4.0 Internacional (CC BY 4.0)

Eres é libre de:

Compartir — copiar e redistribuír o material en calquera medio ou formato

Adaptar — Remesturar, transformar e recrear sobre o material para calquera propósito.



EchidnaEducación

Esta licenza acéptase para obras culturais libres.

O licenciante non pode revogar estas liberdades mentres vostede compra os termos da licenza.

Nos seguintes termos:

Atribución — Debe dar o recoñecemento apropiado, fornecer un vínculo á licenza e indicar se se fixeron cambios. Pode facelo de calquera maneira razoábel pero non de maneira que poida sugerir que o licenciante o apoia a vostede ou o seu uso.

Sen restricións adicionais — Non pode aplicar termos legais ou medidas tecnolóxicas que legalmente impidan a outros facer algo que a licenza permite.

Notas:

Non ten que cumplir coa licenza para os elementos do material que estean no dominio público ou cando o seu uso se permita mediante unha excepción ou limitación aplicábel.

Non se ofrecen garantías. A licenza poida que non forneza todos os permisos necesarios para o uso pretendido. Por exemplo, outros dereitos como publicidade, privacidade, ou dereitos morais (intelectuais).

Redactado en Estrimia , setembro 2022 (001)



Xabier Rosas é un apaixonado da educación e a tecnoloxía, dedicado á promoción da programación e a robótica nas aulas a través do deseño de hardware e o uso software libre. Coa súa experiencia, Xabier crea recursos educativos que fan accesible o mundo da tecnoloxía a estudantes e docentes, fomentando a creatividade e a innovación.

Tres docentes decidimos crear nosa propia ferramenta para traballar con .robótica na aula usando hardware e software libre e contidos didácticos baixo licencia CERN_OHL-W e CC BY 4.0

Creamos Echidna Educación como unha asociación sen ánimo de lucro que ten como obxectivos: Promover o ensino da programación e robótica mediante o uso de ferramentas de código abierto. Facilitar o acceso á programación de dispositivos físicos a través da creación de electrónica de acceso libre "hardware open source" deseñado a partir das necesidades da aula.