

Arduino + Pure Data

Conexión de la Plataforma Open Hardware
Arduino con Pure Data



Ver. 1.0

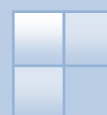


+



José Manuel Ruiz Gutiérrez

Serie: Herramientas Gráficas para la programación de
Arduino



Índice

0. Una primera reflexión.
1. ¿Qué es Pure Data?
2. Gobierno de **Arduino** directamente a través del puerto USB
 - 2.1.1. Gobierno de dos LEDs
 - 2.1.2. Lectura del valor de un sensor.
 - 2.1.3. Lectura de tres valores analógicos.
 - 2.1.4. Lectura de señales analógicas y digitales a la vez
 - 2.1.5. Control del nivel de iluminación de un LED, salida PWM
3. Librería **Pduino**
4. Trabajando con la librería **Pduino**
 - 4.1. METODO DE TRABAJO 0
 - 4.1.1.1. Activación de una salida en modo intermitente “BLINK”.
 - 4.1.1.2. Activación de una salida y lectura de un canal analógico
 - 4.1.1.3. Lectura de dos canales de entrada digital a la vez.
 - 4.1.1.4. Leer y escribir un mismo valor digital
 - 4.1.1.5. Gobierno de dos servos conectados a las salidas PIN9 y PIN10
 - 4.1.1.6. Salida PWM con visualización gráfica.
 - 4.1.1.7. Encendido gradual de dos diodos leds.
 - 4.2. METODO DE TRABAJO 1
 - 4.2.1.1. Ejemplos de trabajo con el modelo “base_m1”
 - 4.2.1.2. Lectura de una entrada digital
 - 4.2.1.3. Lee y escribe señal digital
 - 4.2.1.4. Alarma
 - 4.2.1.5. Vúmetro
 - 4.2.1.6. Alarma tritonal
 - 4.2.1.7. Lectura canal analógico.
 - 4.2.1.8. Termostato básico
 - 4.2.1.9. Contador de impulsos
 - 4.2.1.10. Contador de impulsos con display
 - 4.2.1.11. Control gráfico de dos salidas
 - 4.2.1.12. Generador de tono básico
 - 4.2.1.13. Generador de tono básico con visualización de señal.
 - 4.3. METODO DE TRABAJO 2
 - 4.3.1.1. Blink

- 4.3.1.2. Lectura de un canal analógico y visualización en display
- 4.3.1.3. Rampas salidas PWM
- 4.3.1.4. 8 Salidas aleatorias
- 4.3.1.5. Órgano básico
- 4.3.1.6. Control de una salida mediante una tecla en modo biestable

5. Librerías Gráficas para **Arduino**

- 5.1. Otras Librerías Gráficas interesantes que incluye Pure Data

6. Bibliografía:



Este trabajo está bajo licencia [Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by-nc-nd/3.0/))

0. Una primera reflexión.

Este manual que tienes en tus manos es fruto de mi experiencia docente con alumnos de Grado de Diseño de Producto de la *Escuela de Arte Superior Antonio López (Tomelloso Ciudad Real –España–)* con los que me empené y comprometí en acercarlos al uso de las herramientas **Open Hardware** en el campo del *Diseño e Interacción*.

Mi experiencia con la Plataforma **Arduino** en los últimos años me ha llevado al convencimiento de que esta herramienta puede ser muy útil para los artistas y creativos. Su facilidad de uso y la enorme cantidad de información que se ha difundido por la red de Internet facilitan su uso y con el la incorporación de una poderosa herramienta de diseño y prototipado.

Mi mayor ilusión es que este material sea útil para todos aquellos que lo utilicen. Mi esperanza es poder contribuir con esta pequeña aportación al crecimiento y expansión de este magnífico movimiento mundial que es el **Open Hardware** y **Open Source**. Espero sugerencias y críticas a mi trabajo para poder rectificar los errores y validar los aciertos.

Esta vez me he decidido por **Pure Data**, en adelante **PD**, por ser un Software gratuito y con un amplio respaldo académico por parte de las más prestigiosas instituciones educativas del mundo. Para mí el hecho de que sea utilizado por estudiantes, profesores y artistas es una garantía de su valor.

Ojala y profesores, investigadores, alumnos y entusiastas del arte, el diseño, la música, la electrónica y la informática, aprovechen este trabajo para su formación y afán de conocimiento. Deseo también que cada vez sean más los trabajos que se pongan gratuitamente al servicio de la comunidad. Son muchos los que lo han hecho hasta ahora y espero que sean más los que se sumen a esta idea de “compartir el conocimiento”.

José Manuel Ruiz Gutiérrez j.m.r.gutierrez@gmail.com

Blog de referencia: <http://josemanuelruizgutierrez.blogspot.com>

24 de Agosto 2013



Este trabajo está bajo licencia [Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by-nc-nd/3.0/)

1. ¿Qué es Pure Data?

PD es un lenguaje de programación gráfico desarrollado por [Miller Puckette](#) durante los años 90 para la creación de música por ordenador interactiva y obras multimedia. Aunque Puckette es el principal autor del software, Pd es un proyecto de código abierto y tiene una gran base de desarrolladores trabajando en nuevas extensiones al programa. Está publicado bajo una licencia similar a la licencia BSD. Interesados en DSP. Personas que estudian el procesamiento de señales digitales tienen en PD un laboratorio para experimentar el tiempo real, y programar señales digitales. (Wikipedia)

Con esta herramienta se pueden realizar una amplia gama de aplicaciones relacionadas con la interacción hombre computador en el ámbito de la multimedia y especialmente en el tratamiento del sonido y de las imágenes.

He seleccionado este software por varios motivos entre los que quiero destacar que es de uso libre y que en torno a él existe una amplísima comunidad de usuarios y desarrolladores que mantienen viva su utilización en los más diversos ámbitos.

En relación con nuestros objetivos es muy importante su utilización para comunicarse con la plataforma **Arduino** tanto en su versión básica (**Arduino UNO**) como en la extendida (**Arduino MEGA**).

En este manual no voy a explicar conceptos básicos de uso de PD. Se supone que el lector sabrá manejar al menos de manera básica el entorno. Lo que pretendo es facilitar la conexión de **Arduino** a PD a través de ejemplos básicos que el lector podrá resolver simplemente disponiendo de una tarjeta **Arduino Uno** o similar, del ID **Arduino v 1.05**, el software PD y las librerías **Firmata** y **Pduino** para la comunicación vía puerto USB. Al final del manual pongo una bibliografía interesante para ampliar el tema.

2. Gobierno de Arduino directamente a través del puerto USB

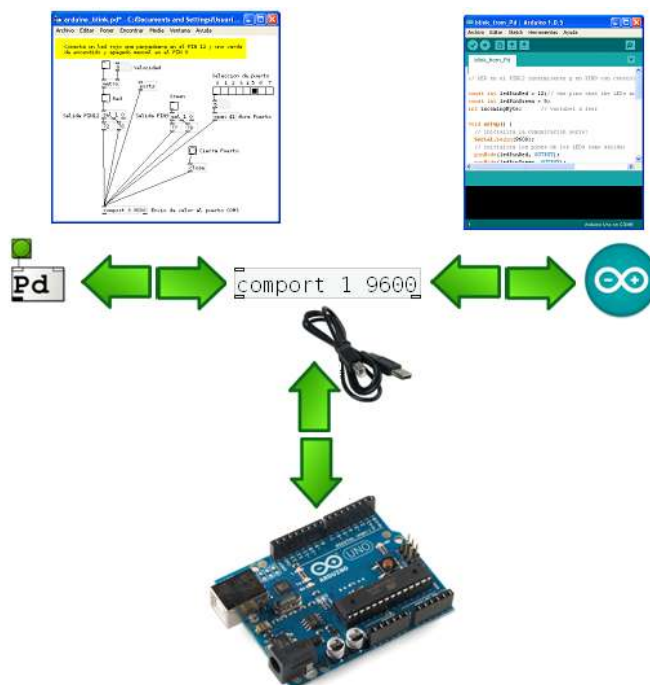
PD permite el intercambio de datos a través del puerto serie, por eso es factible comunicar **Arduino** con esta poderosa herramienta y realizar aplicaciones en las que se ponen en juego el envío de señales en ambos sentidos de una forma muy sencilla.

No es mi objetivo estudiar con detalle los protocolos de manejo de puerto serie por parte de PD, esa tarea la dejo para próximos trabajos.

En la comunicación de **Arduino PD** a través del puerto existen dos aplicaciones a desarrollar bien diferenciadas, a saber:

- Aplicación propia de **Arduino** que deberá residir en la tarjeta y estará escrita con la sintaxis del **IDE Arduino**.
- Aplicación **PD** que se realizara en un **patch** de este y a través de la cual se enviarán y recibirán datos del puerto que irán o vendrán de la tarjea **Arduino**.

Esta forma de comunicarse entre ambas herramientas quizá es muy rudimentaria pero no deja de ser interesante y fácil de manejar. Pasamos a explicar algunos caso prácticos con el fin de poder aprender a manejar el puerto y sus comandos más básicos.



Dara lo mismo luego pues ya le enviaremos el puerto por el que se comunicará.

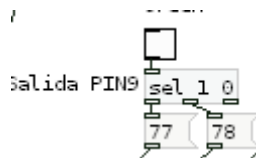
En esta aplicación pretendemos que el LED rojo conectado en el PIN12 parpadee y que el LED verde conectado en el PIN9 sea gobernado mediante un objeto **Toggle**. Para esta forma de gobierno se trata de enviar un mensaje por el puerto serie que sea leído e interpretado por **Arduino**, casi siempre se envían códigos ASCII.

Códigos de gobierno:



Para el LED Rojo PIN12


- H (72) Activa el LED
- L (76) Desactiva el LED

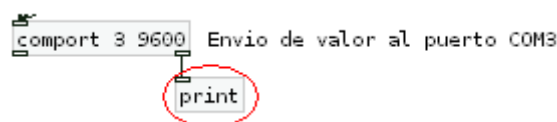


Para el LED Verde PIN9

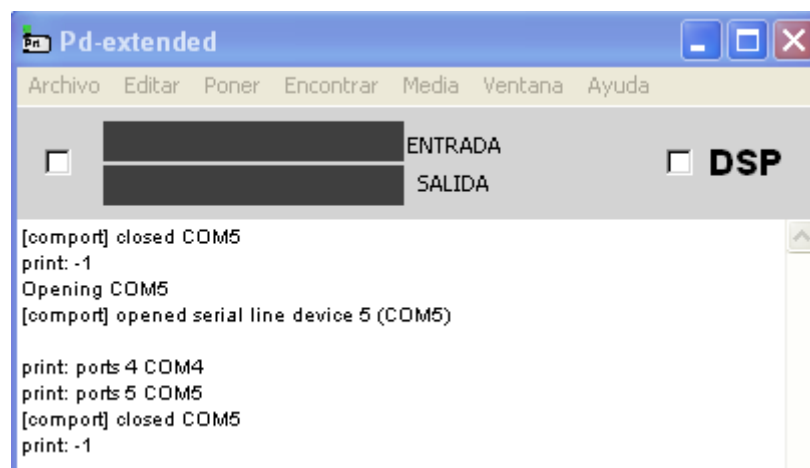
- M (77) Activa el LED
- N (78) Desactiva el LED

La activación intermitente se consigue con un bloque **metro** y la otra con un sencillo **Toggle**.

Con el mensaje **ports** leemos los puertos conectados  en el terminal de PD siempre que pongamos en el **uoutlet** derecho del bloque de puerto una orden **print**



Este es el aspecto que presenta la ventana de trazado de **PD**



Programa de Arduino: *blink_from_pd.ino*

Este es el programa que debemos escribir en el **IDE de Arduino** y cargarlo en la tarjeta. Vemos que sencillamente lo que hace es definir los PIN12 y PIN9 como salidas, abrir el puerto y ponerse a leer, en función del código ASCII que le llega realiza una función u otra.

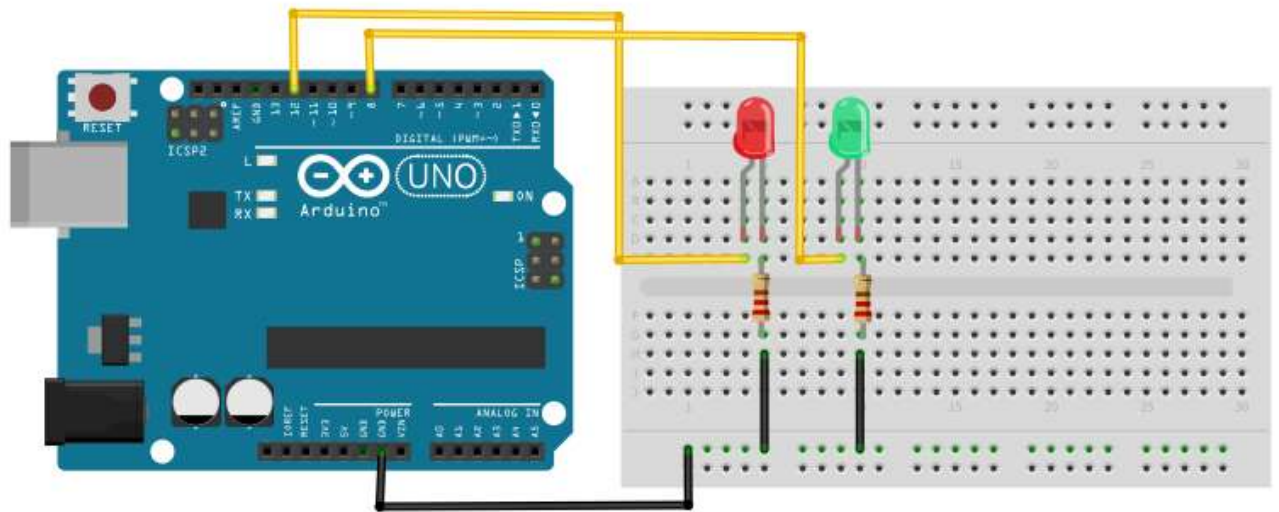
// LED en el PIN12 intermitente y en PIN9 con control manual.

```
const int ledPinRed = 12; // the pins that the LEDs are attached to
const int ledPinGreen = 9;
int incomingByte; // variable a leer

void setup() {
  // inicializa la comunicación serie:
  Serial.begin(9600);
  // inicializa los pines de los LEDs como salida:
  pinMode(ledPinRed, OUTPUT);
  pinMode(ledPinGreen, OUTPUT);
}

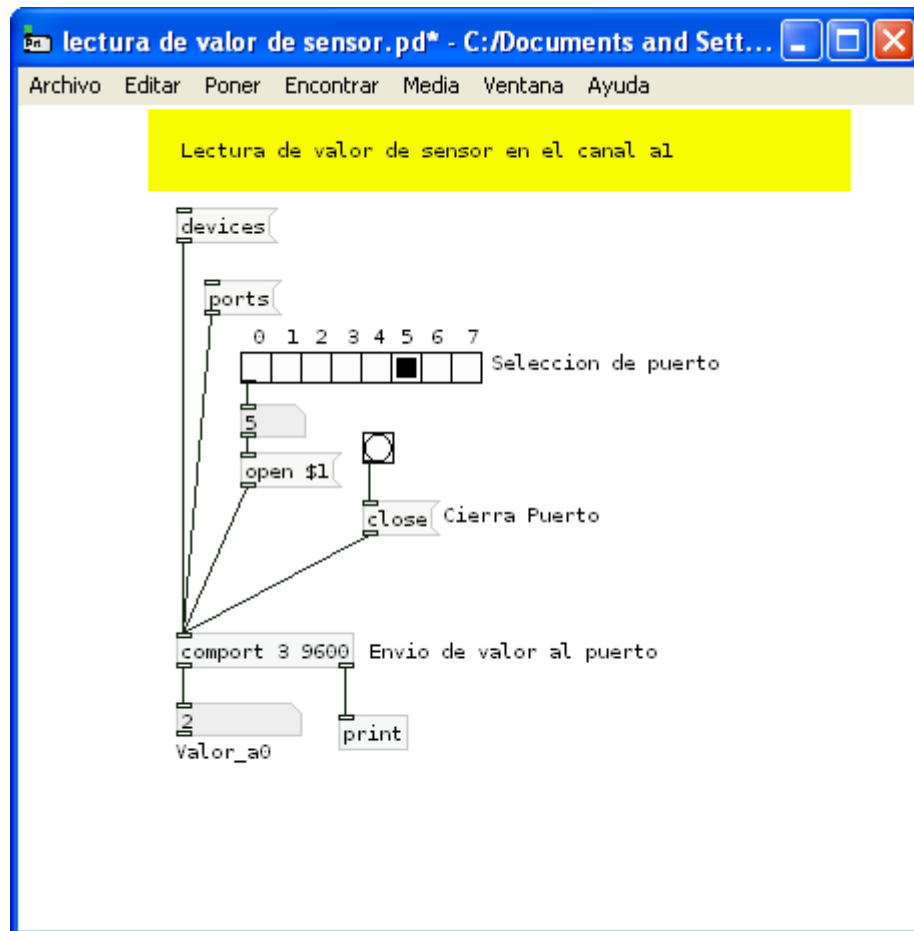
void loop() {
  // mira si está abierto el puerto serie:
  if (Serial.available() > 0) {
    // lee en el buffer:
    incomingByte = Serial.read();
    // si se lee la letra mayúscula H (ASCII 72), cambia de estado al LED rojo:
    if (incomingByte == 'H') {
      digitalWrite(ledPinRed, HIGH);
    }
    // Si la letra leída es la mayúscula L (ASCII 76) desactiva el LED rojo:
    if (incomingByte == 'L') {
      digitalWrite(ledPinRed, LOW);
    }
    // si la letra mayúscula es M (ASCII 77), activa el LED verde:
    if (incomingByte == 'M') {
      digitalWrite(ledPinGreen, HIGH);
    }
    // si la letra mayúscula es N (ASCII 78), desactiva el LED verde LED:
    if (incomingByte == 'N') {
      digitalWrite(ledPinGreen, LOW);
    }
  }
}
```

Esquema de montaje:



2.1. Lectura del valor de un sensor.

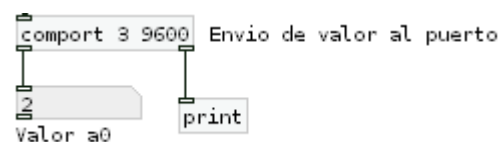
Con este ejemplo vamos a leer el valor del puerto A1 de **Arduino** y lo vamos a mostrar en una caja numérica de PD.




Fichero: *m2- lectura de valor de sensor.pd*

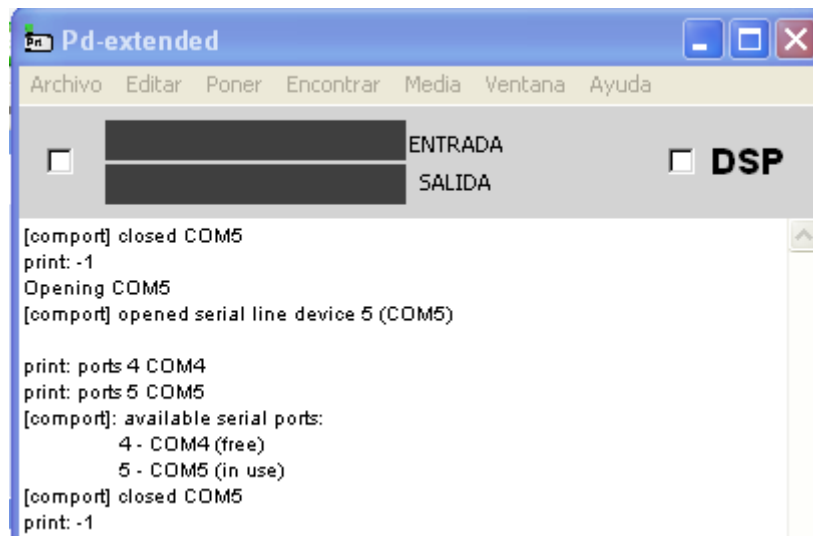
Programa PD:

El programa es muy parecido al anterior. En este caso, una vez abierto el puerto de comunicación vemos que al instante aparece el valor leído en la salida del bloque **comport** (outlet izquierdo) al que vemos conectado un bloque **Numero**



He añadido un mensaje *devices*  para ver como se muestran los elementos que están conectados a cada uno de los puertos que hay y que como ya hemos visto antes se pueden visualizar enviando el mensaje *ports*.

En la figura siguiente se muestra la ventana de dialogo de **PD** en la que vemos los mensajes generados en cada orden dada.



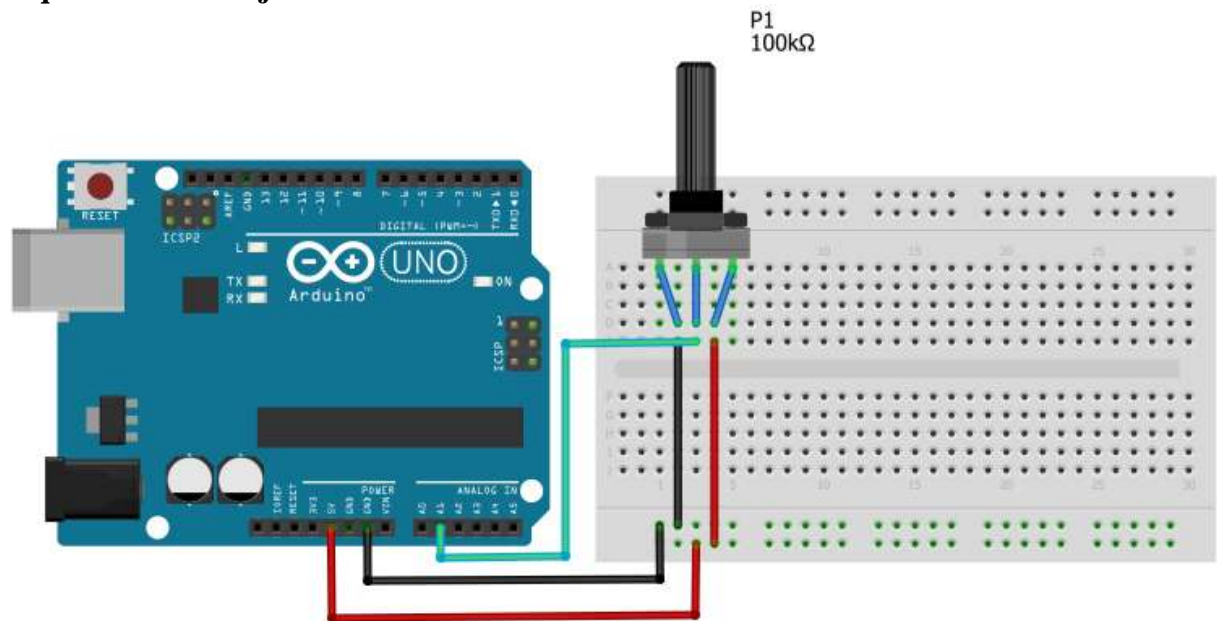
Programa de Arduino: *lectura_de_sensor_en_canal_a1-ino*

El programa a escribir en el **IDE Arduino** es muy sencillo. Se configura la variable de velocidad de transmisión y lo que se hace en la parte **loop** d el programa es leer el canal dividirlo por 4 y mandarlo al puerto

// Lectura de valor de un sensor y envío al puerto

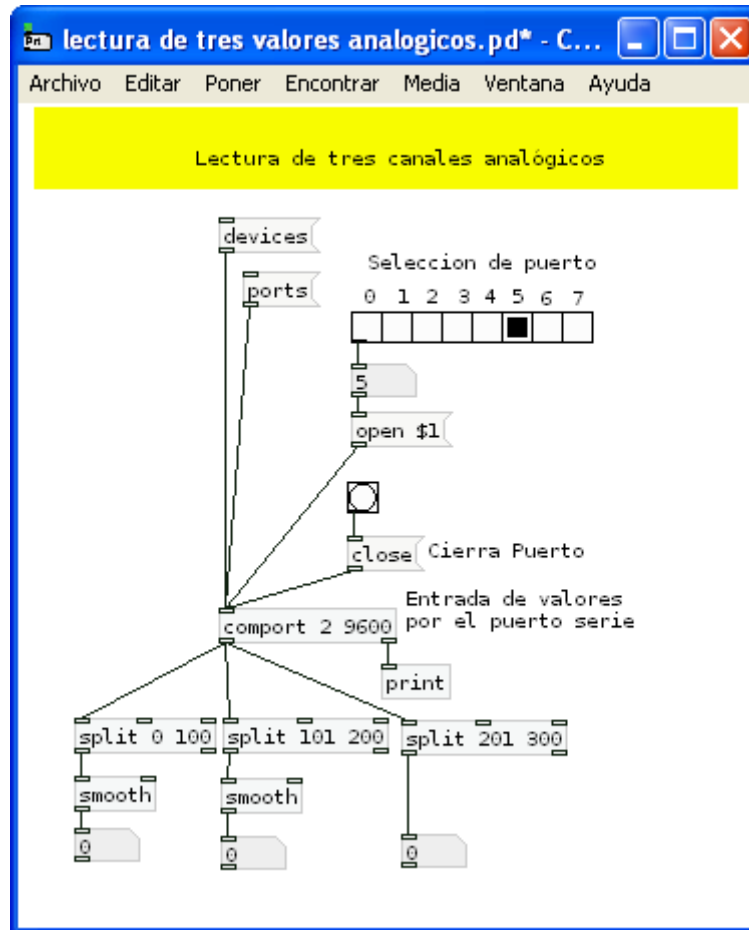
```
void setup() {
  Serial.begin(9600);
}
void loop() {
  int sensorValue = analogRead(A1);
  int serialValue = sensorValue / 4;
  Serial.write(serialValue);
}
```

Esquema de montaje



Lectura de tres valores analógicos.

Con este ejemplo leeremos tres sensores de **Arduino**.

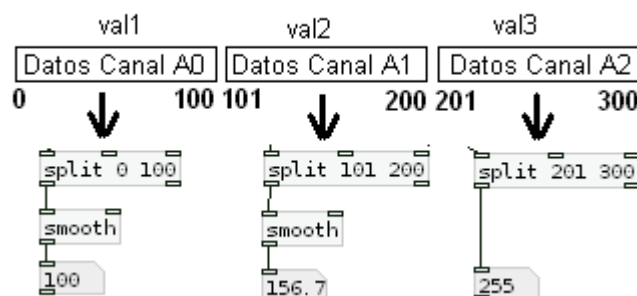


Fichero: *m3-lectura de tres valores analogicos.pd*

Programa para PD:

Arduino enviará al puerto serie, por el que se comunica con **PD**, los valores que lee en sus entradas analógicas. **PD** recibirá estos valores y mediante la función Split realizará la separación de caracteres de las tramas leídas.

ESTRUTURA DE MAPA DE DATOS QUE ENVIA ARDUINO



La función **smooth** extrae los valores de cada bloque de valores.

Este programa se ha recogido del artículo:

Multiple Analog Values from **Arduino** to **PD**

http://colinzyskowski.com/?page_id=503

Programa para Arduino:

Fichero **Arduino**: *lectura_de_tres_valores_analogicos.ino*

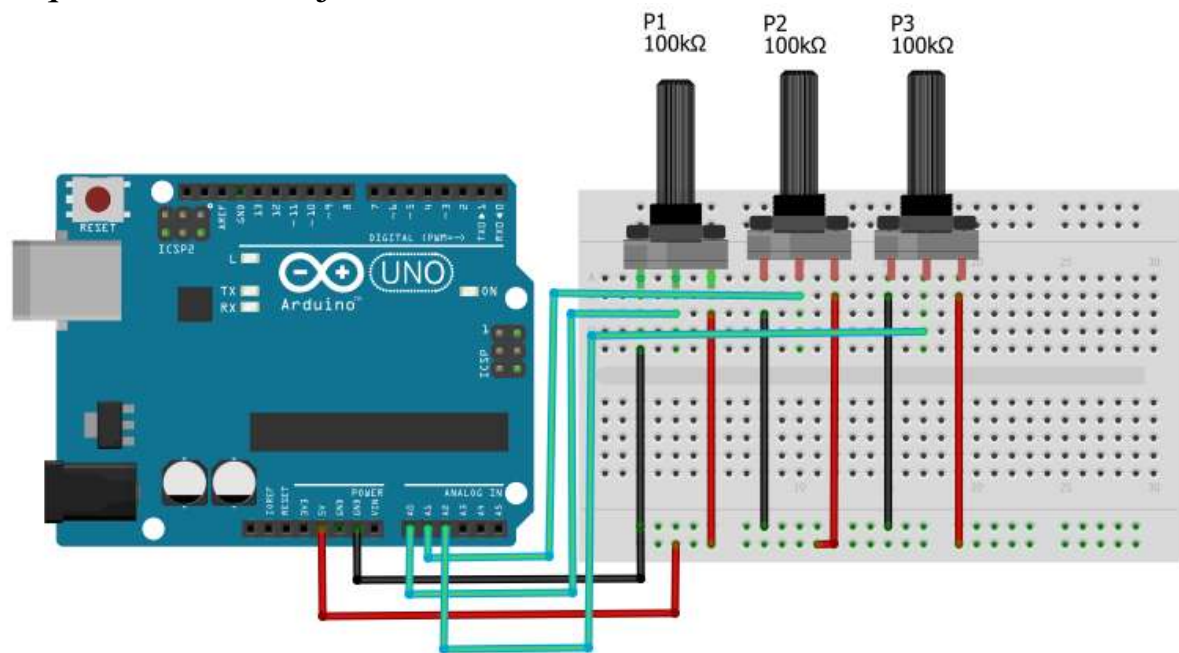
El programa define las áreas de depósito de los datos mediante la función **map** que no dejan de ser **array** de datos de valor comprendido entre **0 y 1023** de **100** datos cada uno y los envía al puerto

// Escribe en el puerto 3 valores de señales analógicas
//Codigo para cargar en Arduino

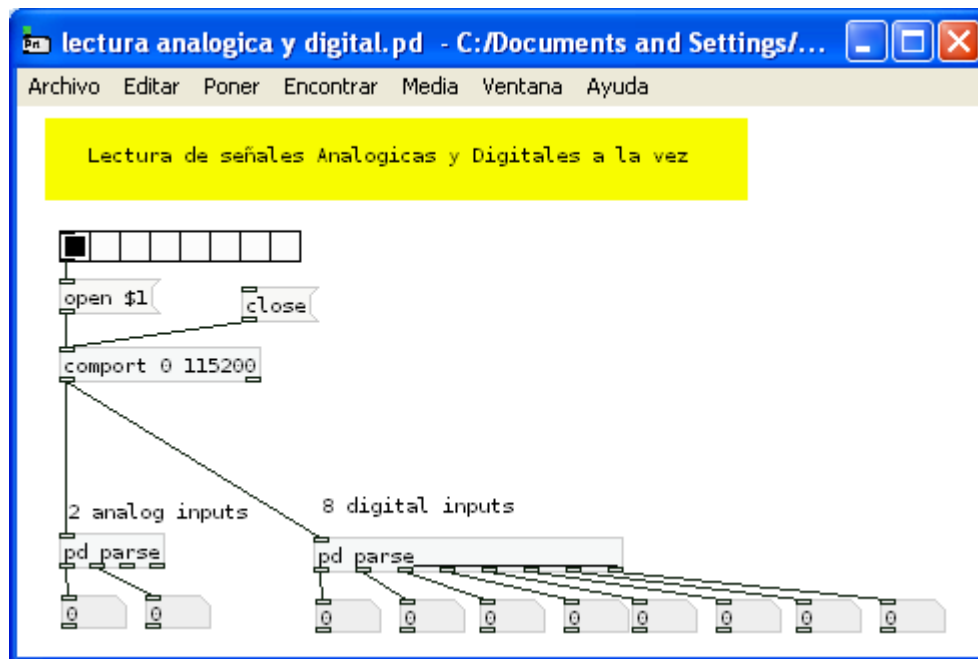
```
int val1;
int val2;
int val3;
void setup(){
  Serial.begin(9600);
}
void loop(){
  val1 = analogRead(0);
  val2 = analogRead(1);
  val3 = analogRead(2);
  //etc...
  val1 = map(val1, 0, 1023, 0, 100);
  val2 = map(val2, 0, 1023, 101, 200);
  val3 = map(val3, 0, 1023, 201,300);
  //podrianos hacer lo mismo para mas sensores
  Serial.write(val1);
  Serial.write(val2);
  Serial.write(val3);
  //NO USAR Serial.println porque los valores no pueden ser leido por Pd
  correctamente
  delay(100);
}

//fin del codigo de arduino
```

Esquema de montaje:



2.2. Lectura de señales analógicas y digitales a la vez



Fichero: *m4-lectura analógica y digital.pd*

Programa para PD:

Leeremos en esta ocasión a través del puerto datos de los canales digitales y datos de los analógicos y después, mediante dos patch de **PD pd parse** los decodificamos y los mostramos. Los bloques no los explicaré por escaparse a mis pretensiones en este manual. El ejemplo lo pongo tal y como lo he recogido del foro de **PD**

Programa para Arduino:

Programa **Arduino**: *lectura_anlogica_y_digital.ino*

/*

Lectura de canales Analogicos y Digitales desde PD

*/

```
float val1;
float valDig;
int potPin1 = 0;
int potPin2 = 1;
int digPin2 = 2;
int digPin3 = 3;
int digPin4 = 4;
int digPin5 = 5;
```

```
int digPin6 = 6;
int digPin7 = 7;
int digPin8 = 8;
int digPin9 = 9;
void setup(){
  Serial.begin(115200);
  pinMode(potPin1,INPUT);
  pinMode(potPin2,INPUT);
  pinMode(digPin2, INPUT);
  pinMode(digPin3, INPUT);
  pinMode(digPin4, INPUT);
  pinMode(digPin5, INPUT);
  pinMode(digPin6, INPUT);
  pinMode(digPin7, INPUT);
  pinMode(digPin8, INPUT);
  pinMode(digPin9, INPUT);
}

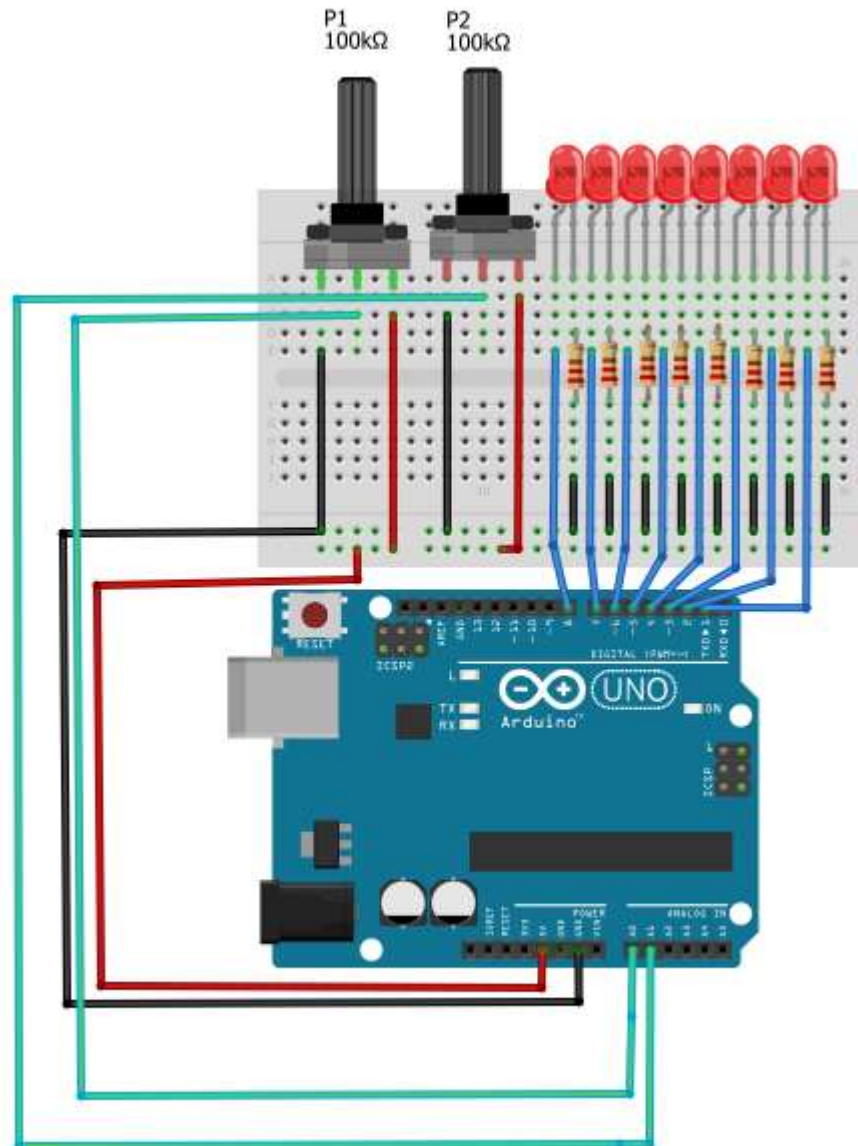
void loop(){

  // Analog Pins...
  for (int j=0;j<2;j++) {
    val1=analogRead(j);
    Serial.print(j);
    Serial.println(val1,DEC);
  }

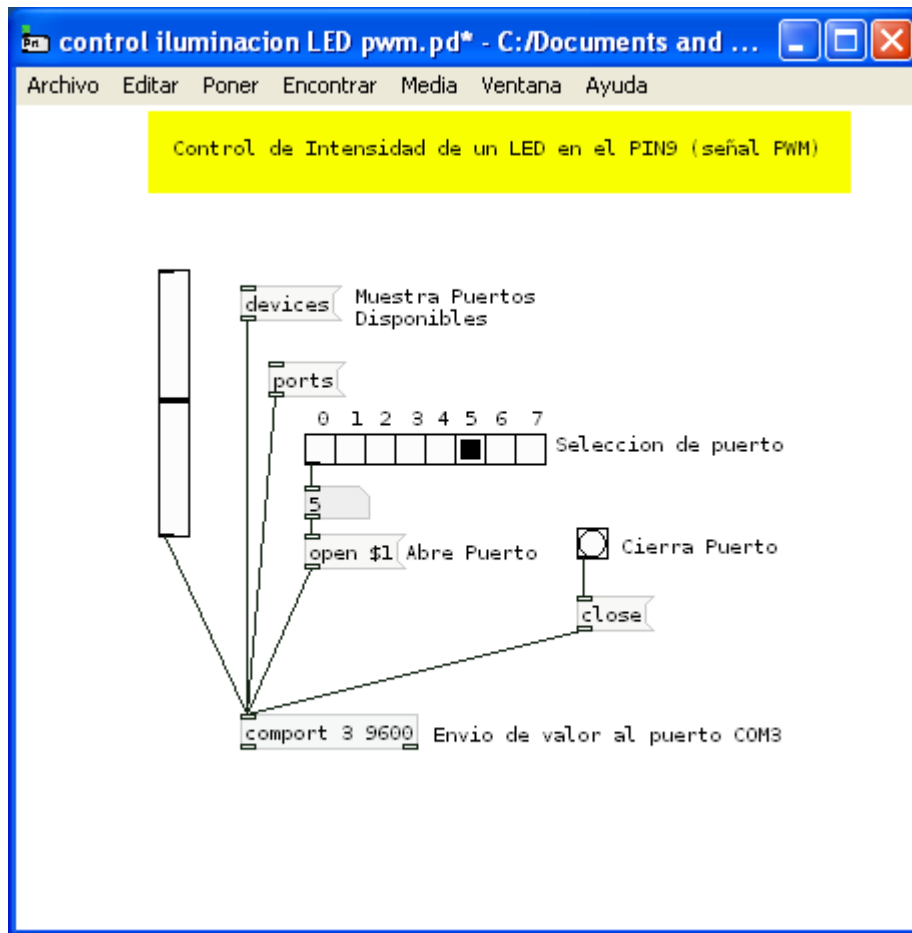
  // Digital Pins...
  for (int i=2;i<10;i++) {
    valDig=digitalRead(i);
    // "ID" for digital inputs... questionable...
    Serial.print(5,DEC);
    Serial.print(i,DEC);
    Serial.println(valDig,0);
  }

}
```

Esquema de montaje:



2.3. Control del nivel de iluminación de un LED, salida PWM



Fichero: *m5-control iluminación LED pwm.pd*

Programa para PD:

Este ejemplo consiste en enviar un valor analógico desde **PD** para que sea leído a través el puerto por **Arduino** y lo lleva a una de sus salidas **PWM**, concretamente al **PIN9**.

Básicamente se trata de abrir el puerto y enviar un valor directamente generado por un objeto **Vslider**

Programa para Arduino:

El programa para **Arduino** leerá el valor y lo asignara a una variable que será la que coloque en la salida del **PIN9**

Fichero: *Control_intensidad_led.ino*

// Control de la intensidad de un led desde PD

```
int incomingByte = 0; // define variable a la que asociara el valor leído
int led = 9;

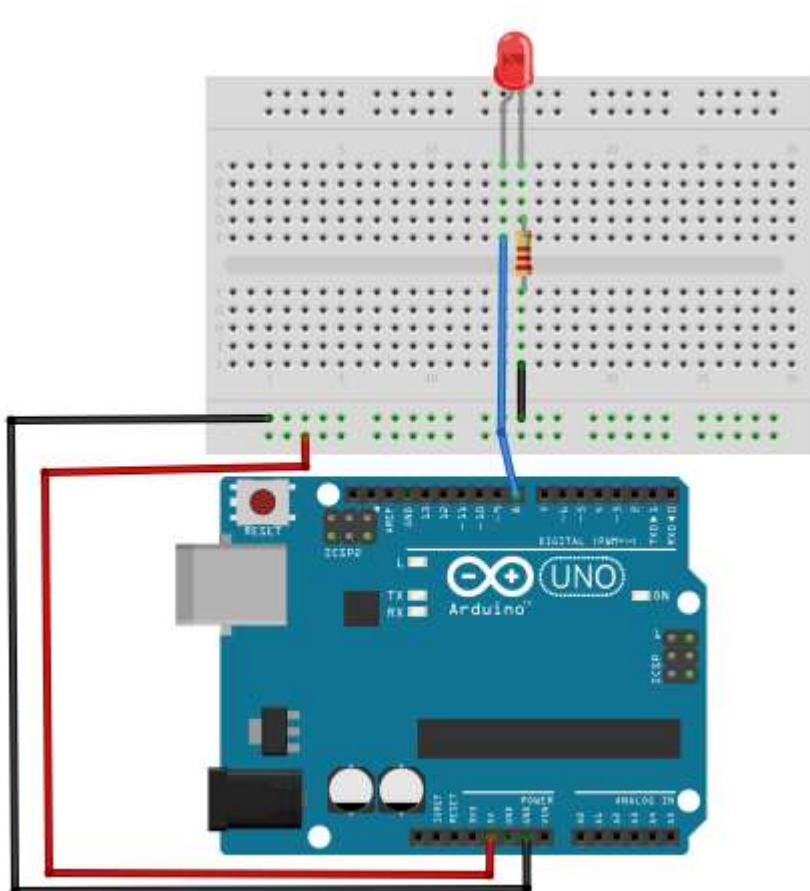
void setup() {
  Serial.begin(9600); // abre el puerto serie con una tasa de velocidad de 9600 bps
  pinMode(led, OUTPUT);
}

void loop() {

  // envia data solo cuando recibe dato:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

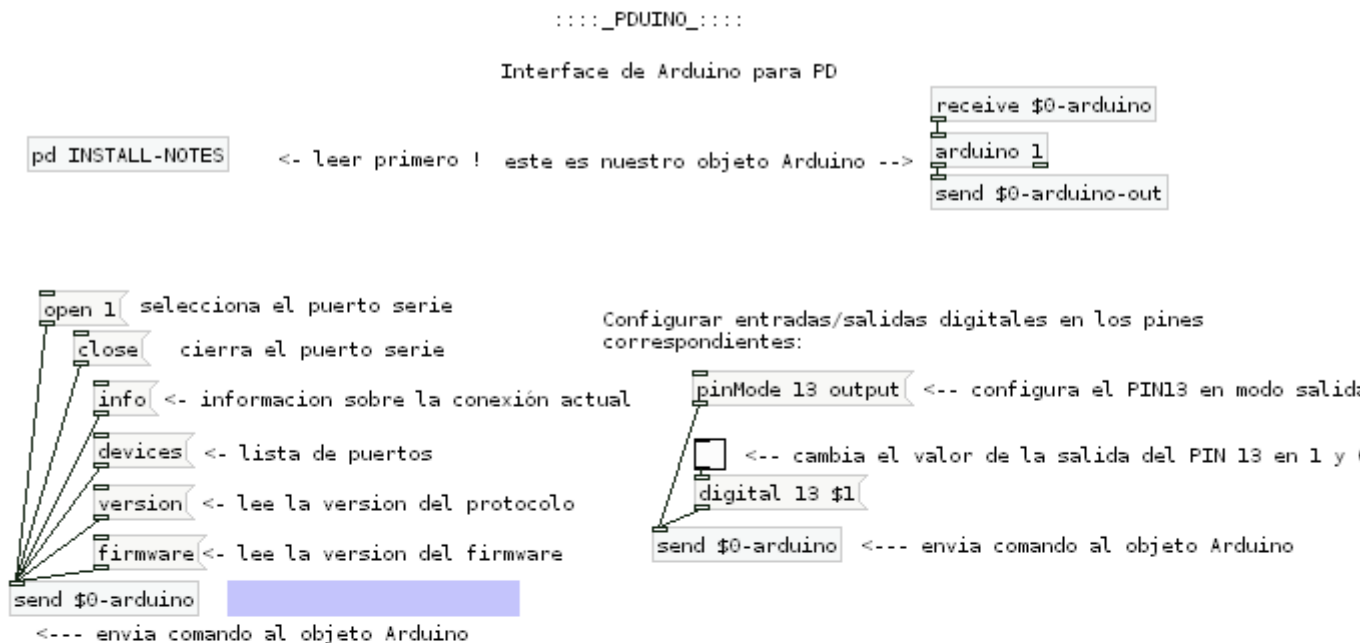
    // escribe el valor en la salida:
    analogWrite(led, incomingByte);
  }
}
```

Esquema de montaje:



Librería Pduino

La librería **Pduino** fue creada para conectar **Arduino** con PD a través del conocido Firmware denominado **Firmata** (<http://firmata.org/>). Recomiendo ver <http://arduino.cc/en/Reference/Firmata> en donde se explica con detenimiento esta librería. Actualmente se dispone de la versión “pduino 0.5.beta8” que se puede descargar en [pduino 0.5.beta8](#)



Con la tarjeta Arduino se pueden leer entrada y escribir en salidas digitales y analógicas, como se describe en los siguientes comandos que se encapsulan en los sub-parches:

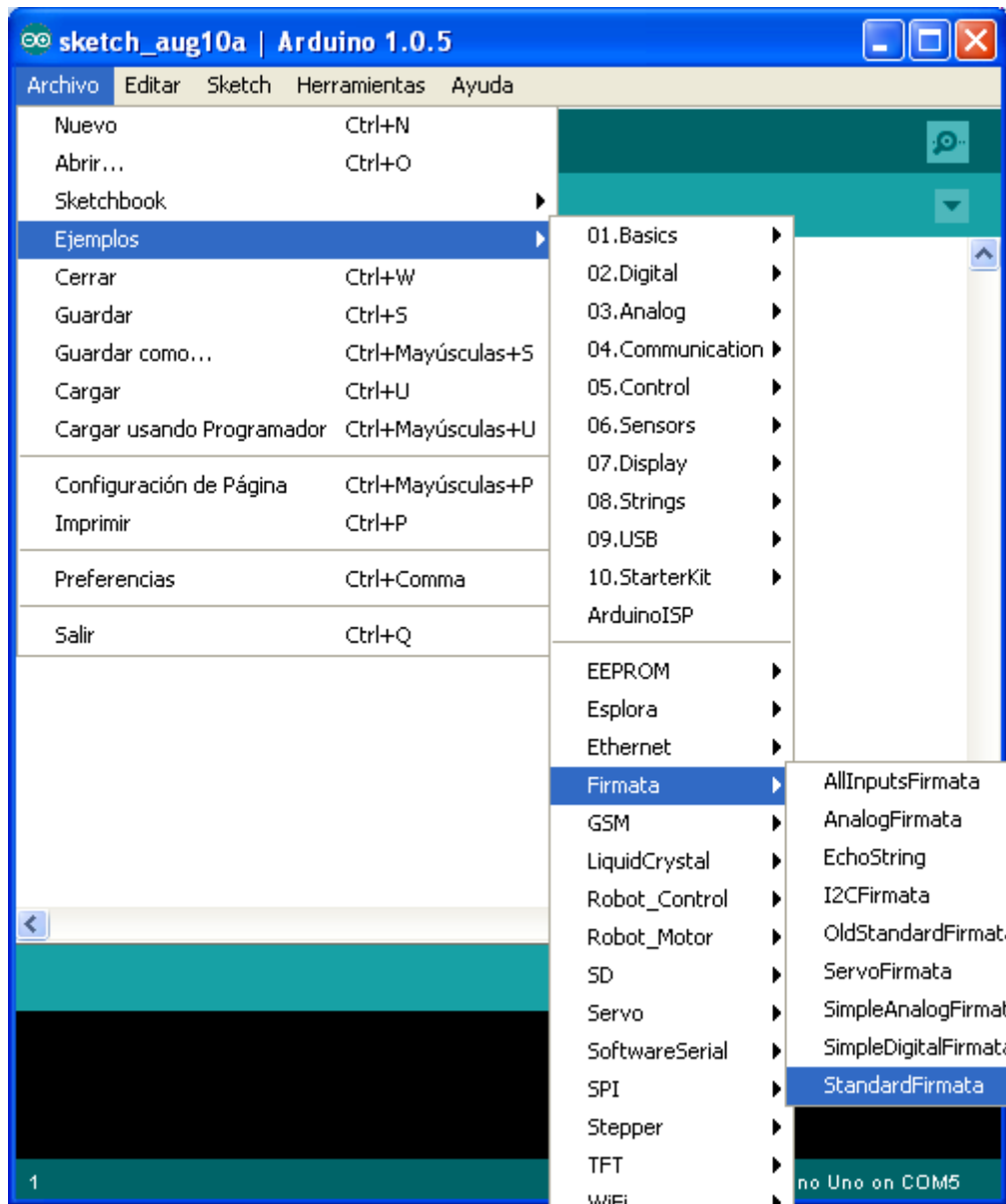
pd PINMODE	<- primero es preciso configurar el modo de trabajo del PIN (pinMode)
pd DIGITAL-INPUT	<- ejemplo de Entrada Digital
pd DIGITAL-OUTPUT	<- ejemplo de salida Digital
pd ANALOG-INPUT	<- ejemplo de Entrada Analógica
pd ANALOG-OUTPUT-PWM	<- ejemplo de Salida Analógica
pd SERVO	<- ejemplo de Control de Servo
pd GETTING-INFO	<- ejemplo para recoger estado de datos
pd SWITCHING-INPUTS	<- ejemplo para conmutar el valor de salidas on/off (optional)

2006, released under GNU GPL;
Gerda Strobl, Georg Holzmann

Traducido por J.M.Ruiz Agosto 2013

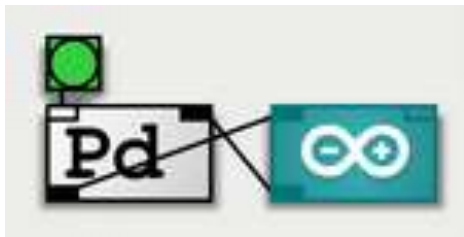
Este es el aspecto de la librería en formato pd y traducida al español que se puede ver en el fichero *arduino-help.pd* anexo a este manual.

IMPORTANTE: A continuación vamos a explicar los comandos y utilidades de la librería **Pduino** y para poder realizar los ejemplos que voy a explicar es preciso que se cargue, previamente, el firmware correspondiente en la tarjeta **Arduino** en el fin de poder realizar la comunicación entre **Arduino** y PD. El firmware que yo he utilizado es el mismo que viene en el IDE de **Arduino** que es el que he comentado anteriormente (Firmata). En la versión del IDE de **Arduino** 1.0.5 y posteriores se puede cargar yendo al menú Archivo-> Ejemplos-> y cargando el fichero *StandardFirmata* dentro de la carpeta *Firmata*



Este es el Firmware con el que realizaremos todos los trabajos relativos a la librería **Pduino**. Lo cargamos en el IDE y después lo descargamos sobre **Arduino**. Una vez realizada la descarga ya podemos comunicarnos con PD.

Vamos a comentar cada uno de los comandos que reconoce esta librería. Antes de nada es preciso dejar claro:



NOTA IMPORTANTE: En todos los ejemplos y ejercicios que realicemos siempre deberá figurar en la carpeta en la que coloquemos nuestros ejercicios los ficheros *arduino.pd* y *arduino-help.pd* con el fin de que cuando sea invocado este objeto se pueda leer el fichero correspondiente. Si no lo hacemos así deberíamos indicar el patch en el que se encuentra estos dos ficheros que contiene la librería y la ayuda para su manejo.

Una vez entro del entorno la librería se invoca mediante la opción “Poner Objeto” y escribiendo su nombre, es decir **Arduino** y de esa modo aparecerá el bloque de la figura:

```
arduino 1
```

Si no está el fichero en la carpeta de uso aparecerá rodeado con un recuadro en línea discontinua roja.

En este manual, no obstante, he incluido ya fichero tipo pd con la librería cargada y en alguno de ellos con un esquema de la tarjeta **Arduino** que facilita el trabajo. Mas adelante lo explicaremos con detalle.

La librería (bloque arduino) se comunica con el resto de operadores de Pure Data mediante las funciones **receive** y **send** (se puede escribir también simplemente **r** y **s**) seguida de un texto que representa las variables asociadas a ese comando. En adelante veremos ejemplos concretos

```
receive $0-arduino
arduino 1
send $0-arduino-out
```

Comandos generales de manejo del puerto USB (COM) del que conectamos **Arduino**. En la siguiente figura se ven las más importantes:

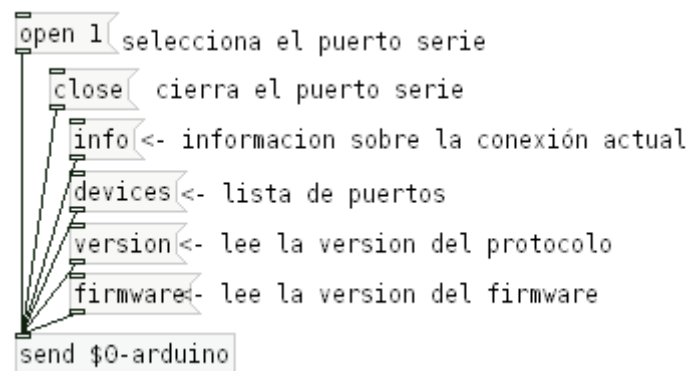
Mediante el bloque “**mensaje**” *open 1* abrimos el puerto numero 1.

El **mensaje** “*close*” cierra el puerto que estuviese previamente abierto.

El **mensaje** “*devices*” muestra la lista de elementos conectados

El **mensaje** “*version*” muestra la version del software **Pduino**

El **mensaje** “*firmware*” muestra la version del firmware cargado en **Arduino**



Todos ellos se envían al objeto **arduino** mediante el comando **send** en el que se especifica un valor genérico de variable **\$0** y el nombre del objeto **arduino**.

Con la tarjeta **Arduino** se pueden leer entrada y escribir en salidas digitales y analógicas, como se describe en los siguientes comandos que se encapsulan en los **sub-patches**:

SUB-PATCHES

Si abrimos el fichero “arduino-help.pd” que hemos mostrado anteriormente y hacemos clic con botón izquierdo del ratón estando el puntero sobre uno de ellos se abrirá el sub.patch correspondiente y podremos ver en él las funciones propias que se indican en su propio nombre. Es muy recomendable hacerlo y fijarse en los bloques de función que se colocan en cada caso. Explicare uno por uno estos sub-patch con el fin de aclarar la funcionalidad de la librería y de paso me ayudaré de sencillos ejemplos que a través de los comandos oportunos nos permitan comunicarnos e interactuar con arduino.

pd PINMODE	<- primero es preciso configurar el modo de trabajo del PIN (pinMode)
pd DIGITAL-INPUT	<- ejemplo de Entrada Digital
pd DIGITAL-OUTPUT	<- ejemplo de salida Digital
pd ANALOG-INPUT	<- ejemplo de Entrada Analógica
pd ANALOG-OUTPUT-PWM	<- ejemplo de Salida Analógica
pd SERVO	<- ejemplo de Control de Servo
pd GETTING-INFO	<- ejemplo para recoger estado de datos
pd SWITCHING-INPUTS	<- ejemplo para conmutar el valor de salidas on/off (optional)

pd PINMODE:

En este bloque se recogen las funciones necesarias para definir el modo de trabajo de cada uno de los pines de **Arduino**.

De manera básica para configurar un PIN de **Arduino** basta con enviarle al bloque **Arduino** mediante la función **mensaje** un mensaje con la siguiente sintaxis:

pinMode X z

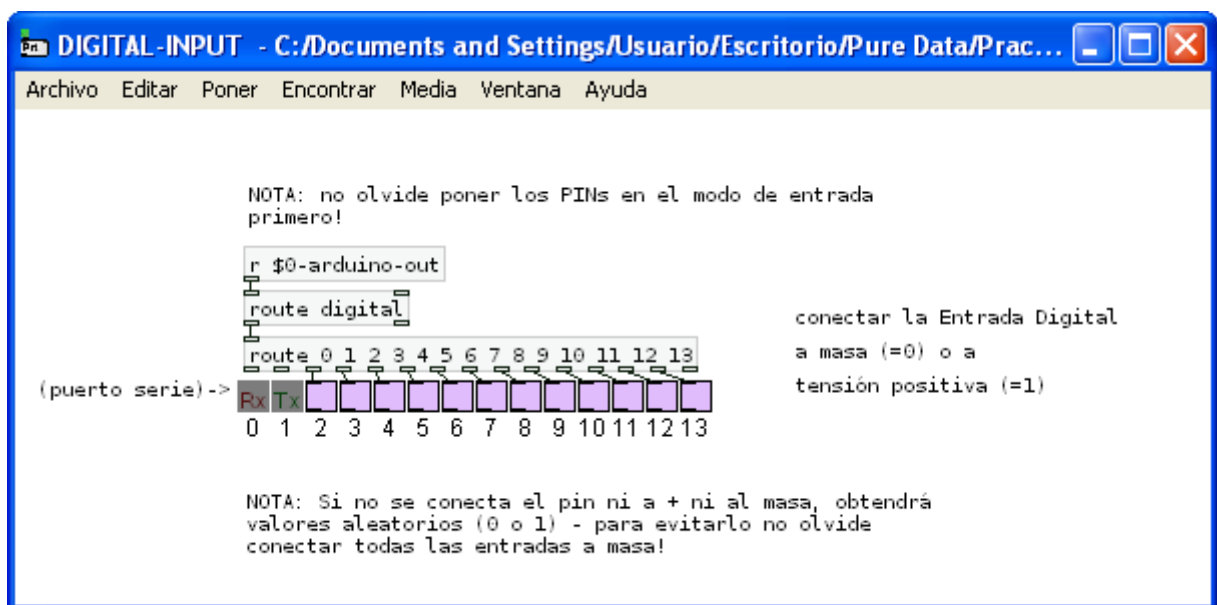
X= El numero de pin

z=valor 0 para *input*, 1 para *output*, 2 para *analog*, 3 para *pwm* y 4 para *servo*. Se puede escribir o bien el numero o bien el tipo

pinMode 10 0	==	pinMode 10 input
pinMode 10 1	==	pinMode 10 output
pinMode 10 2	==	pinMode 10 analog
pinMode 10 3	==	pinMode 10 pwm
pinMode 10 4	==	pinMode 10 servo

Más adelante veremos como se puede realizar la configuración de los pines de una manera mas cómoda (todos de una sola vez)

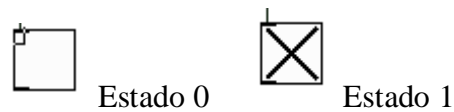
pd DIGITAL-INPUT:



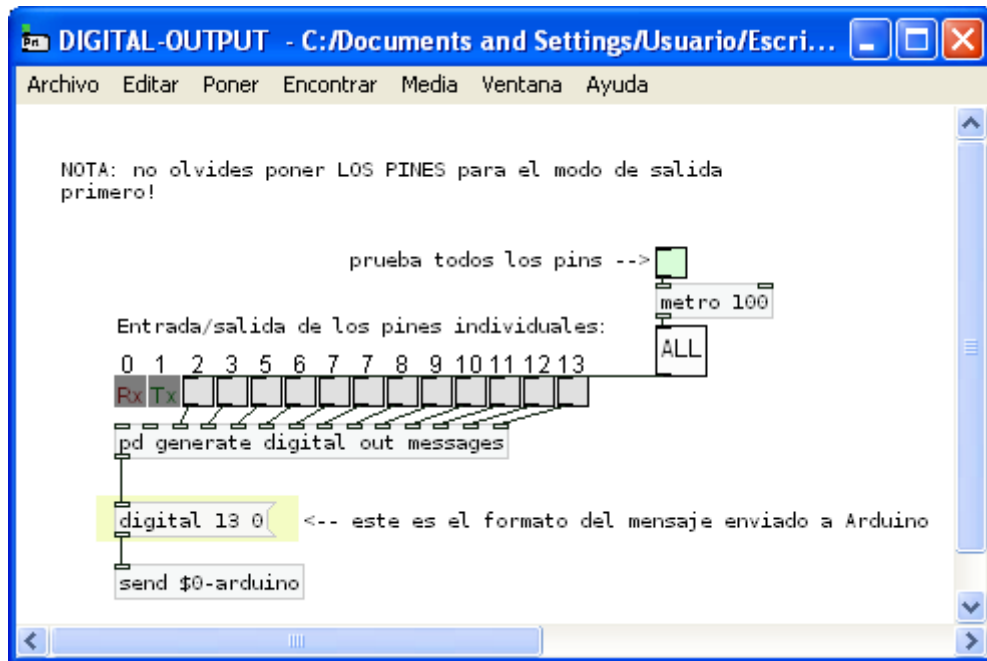
En este caso vemos la manera de leer los datos de entrada digital de la tarjeta **Arduino** (estado de pulsadores, interruptores, etc.. conectados al los pines que hayamos configurado previamente como entrada).

Vemos que lo primero que se hace es utilizar una función de recepción de datos, *r \$0-arduino-out* que entrega su contenido una vez que lo toma del propio bloque principal “**arduino**” al bloque *route digital* que extrae los valores de las señales digitales de entrada y mediante el siguiente bloque *route 0 1 2 3 4 5 6 7 8 9 10 11 12 13* lo pone de manera individualizada en un objeto **toggle**.


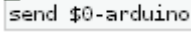
Cada vez que cambie el estado de una entrada se verá reflejado el valor en el objeto **toggle**



pd DIGITAL-OUTPUT:

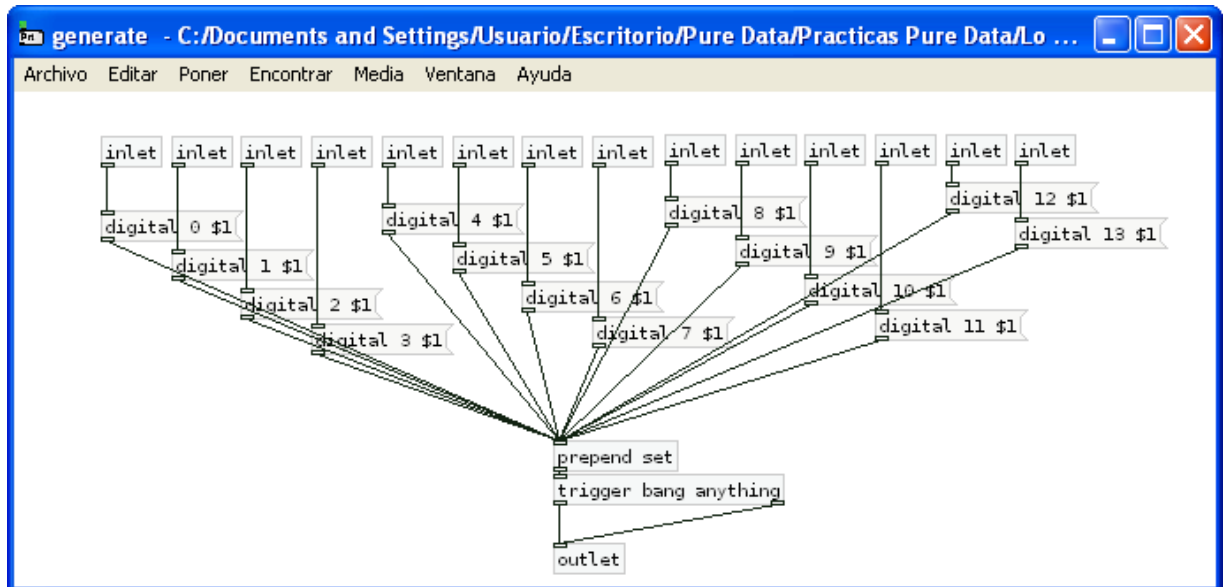


En este caso vemos la forma de enviar datos a las salidas digitales de **Arduino**.

Lo que se hace es recoger el estado de los bloques **toggle** (interruptores)  y componerlos juntos en un “telegrama” que se envía mediante la instrucción **message** a la función `send $0-arduino`  que lo entrega al bloque principal **arduino**.

El bloque que realiza esta operación es un **sub-patch** denominado *pd generate digital out messages* 

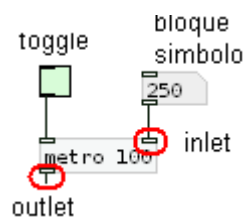
Este es contenido del **sub-patch** *pd generate digital out messages*



Se ha colocado un generador de impulso metro 100 que gobernado por un **toggle** activa todas las salidas (las que estén programadas como tales).

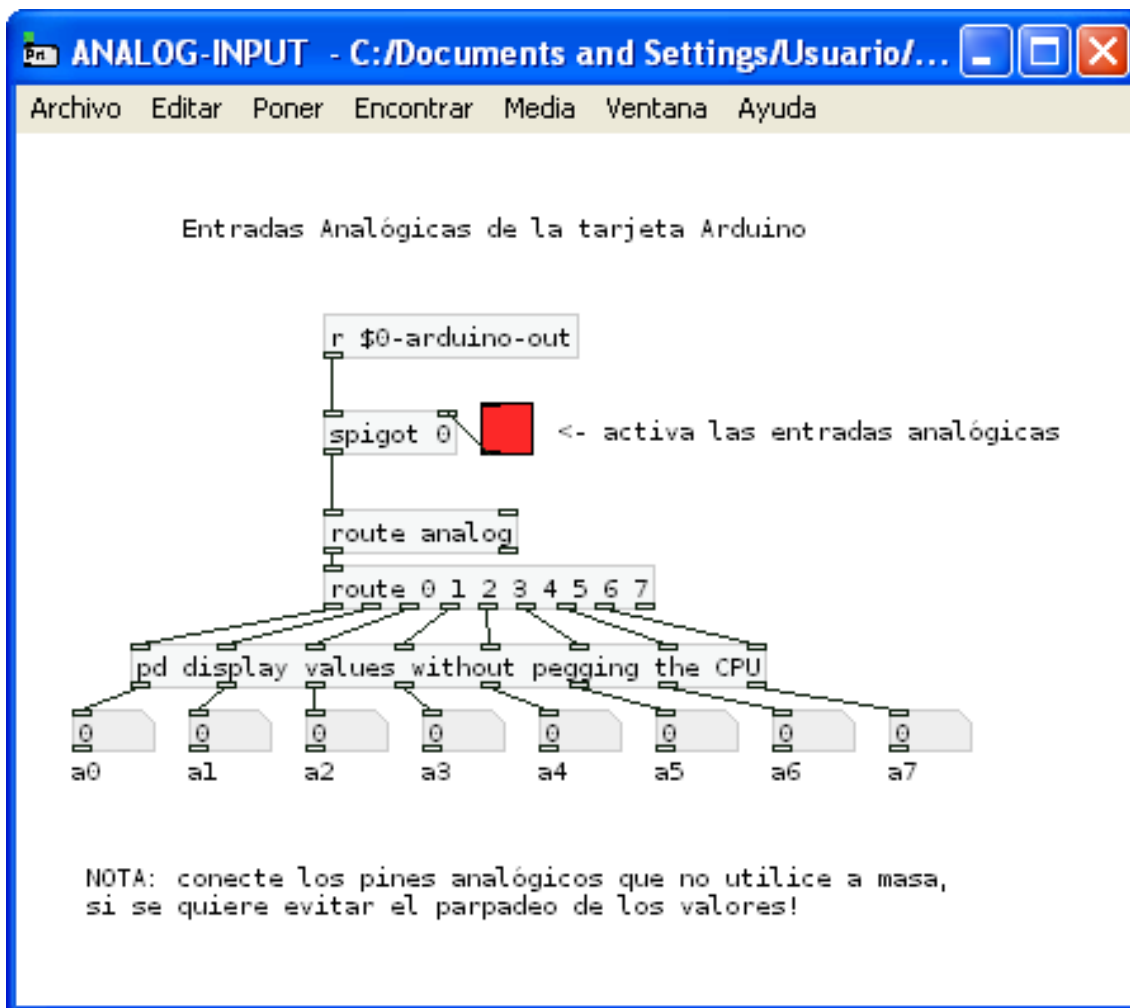


La función **metro** es muy interesante para nuestro trabajo. Esta función genera impulsos de una duración que podemos fijar: *metro 100* (impulsos de valor 100 ms.) Podríamos colocar una entrada *inlet* derecho del bloque y poder variar con ella la duración simplemente desplazando sobre ella el puntero del ratón pulsando el botón izquierdo.



Generador de impulsos variables

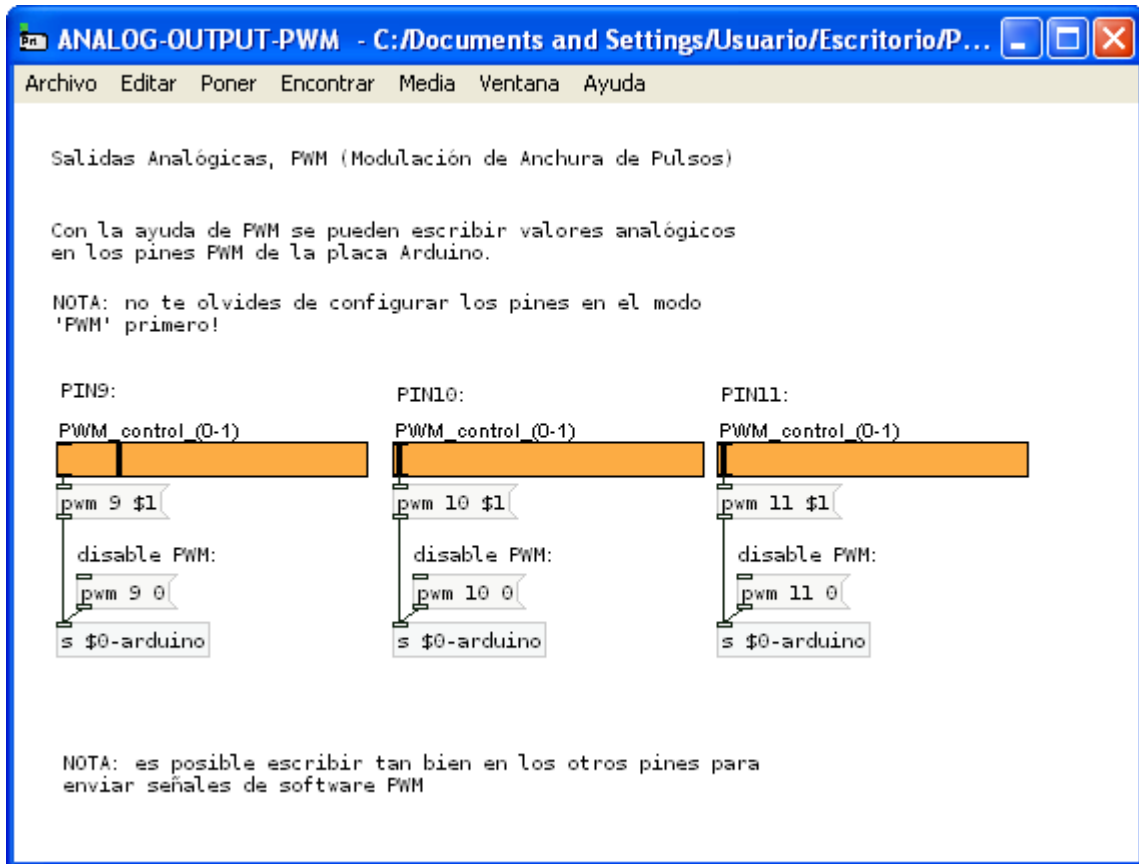
pd ANALOG-INPUT:



Para leer el estado de las entradas analógicas de **Arduino** recurrimos al uso de la función **read** `r $0-arduino-out` que recoge los valores analógicos y los entrega al bloque **route analog** que a su vez los entrega al **sub-patch** `pd display values without pegging the CPU` que los saca de manera individual mediante los bloques de **simbolo**. La función intermedia `spigot 0` sirve para establecer el corte de tramas de datos utilizando el 0 como separador de trama, pero esta no es necesaria en las últimas versiones de la librería **Pduino**.

Veremos luego una forma de configurar de una sola vez la lectura de los canales analógicos seleccionados.

pd ANALOG-OUTPUT-PWM:



Para el gobierno de una salida analógica sabemos que **Arduino** realice una modulación de anchura de pulso PWM que viene a ser la emulación de una señal analógica y que es posible configurar en las salidas PIN3, PIN5, PIN6, PIN9, PIN10 y PIN11 para la versión Arduinio UNO.

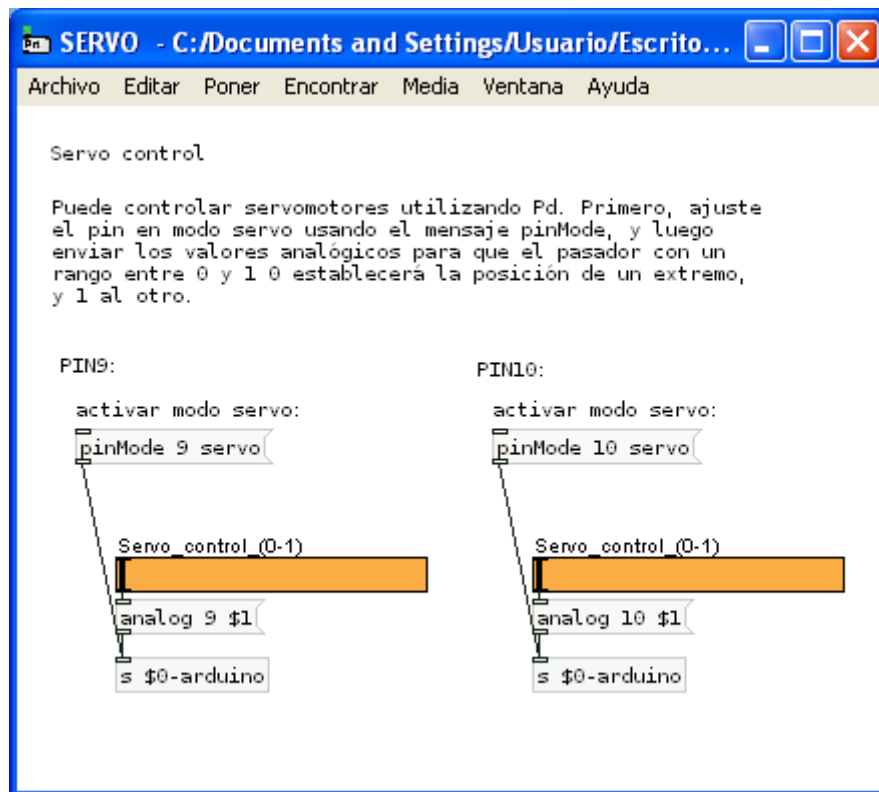
En este caso vemos como ejemplo el gobierno de las salidas PIN9, PIN10 y PIN11 mediante unos sliders. La orden que enviamos al bloque arduino es un message con el contenido

pwm x \$1 en donde **x** representa el numero de pin y **\$1** el valor enviado.

Obsérvese que el valor a enviar esta comprendido entre **0** y **1** lo cual significa que de modo real ese rango de **0** a **1** convierte el sistema de **0** a **255** que como sabemos es el valor máximo de estas salidas **PWM**

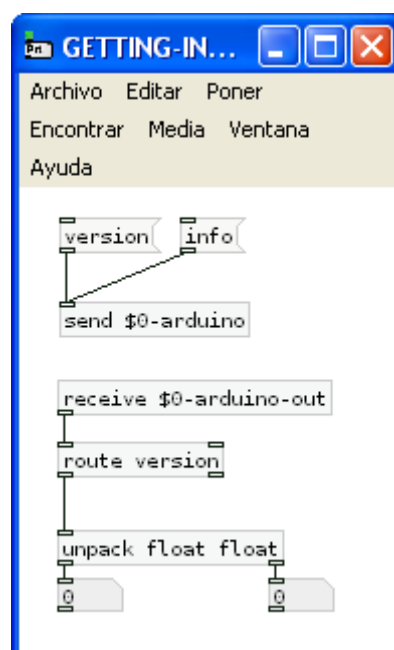
Por lo demás el resto de bloques ya los conocemos si bien es cierto que se ha añadido un bloque **message** que pone a cero la salida al hacer clic sobre el, enviando el mensaje *pwm x 0*

pd SERVO:



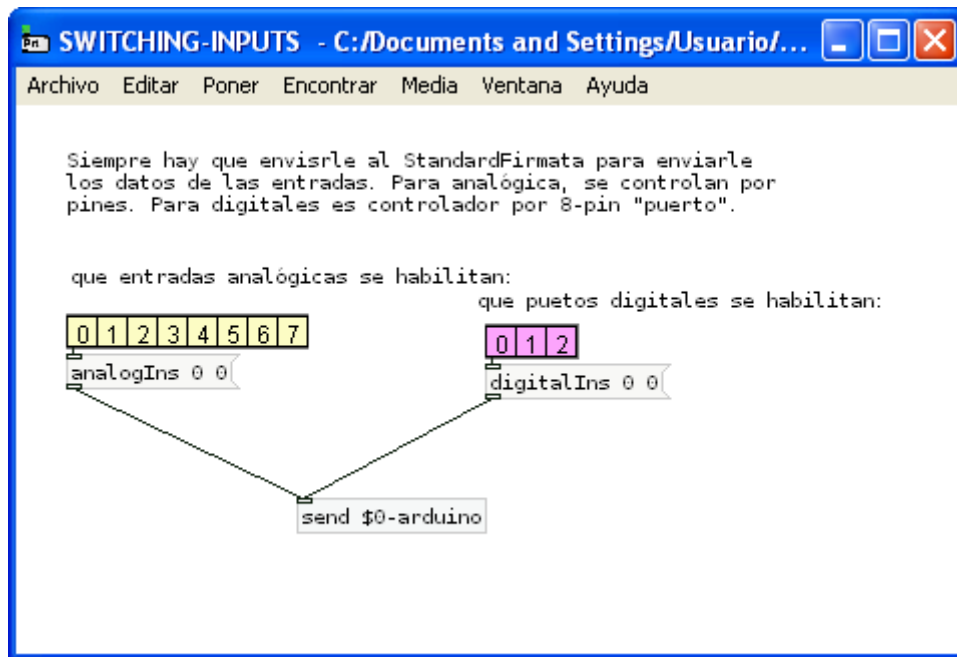
Basta enviar el mensaje *pinMode 9 servo* para configurar el PIN9 de **Arduino** en formato de servo y después lo que debemos hacer es mandar una señal de valor entre 0 y 1 mediante el mensaje *analog 9 \$1* para que en este PIN podamos colocar un servo y moverlo.

pd GETTING-INFO:



Con estos comandos enviados en modo mensajes a arduino se puede saber la versión de firmadata y otras informaciones. Más adelante lo ampliaremos.

pd SWITCHING-INPUTS:



Con estas utilidades podemos habilitar tanto los puertos analógicos como los digitales.

Para habilitar basta enviar los mensajes *analogIns x* y o *digitalIns x* y en los que x representa el número de canal e y el estado (o deshabilitado y 1 habilitado)

3. Trabajando con la librería Pduino

Ejemplos de uso de comandos Pduino.

En adelante vamos a ir mostrando una serie de ejemplos que nos permitan comprender mejor el modo de trabajo de la librería **Pduino**.

Software necesario:

Para la elaboración de todas las prácticas que figuran en este manual he utilizado las siguientes herramientas.

IDE Aduino Ver. 1.0.5 Que se puede descargar en
<http://arduino.cc/en/Main/Software>

PD Ver 0.43-4 Extended. Que se puede descargar en
<http://puredata.info/downloads>

La libreria Pduino. Que se puede descargar en
<http://at.or.at/hans/pd/objects.html>

El firmware Firmata Que se puede descargar en
http://firmata.org/wiki/Main_Page

NOTA IMPORTANTE:

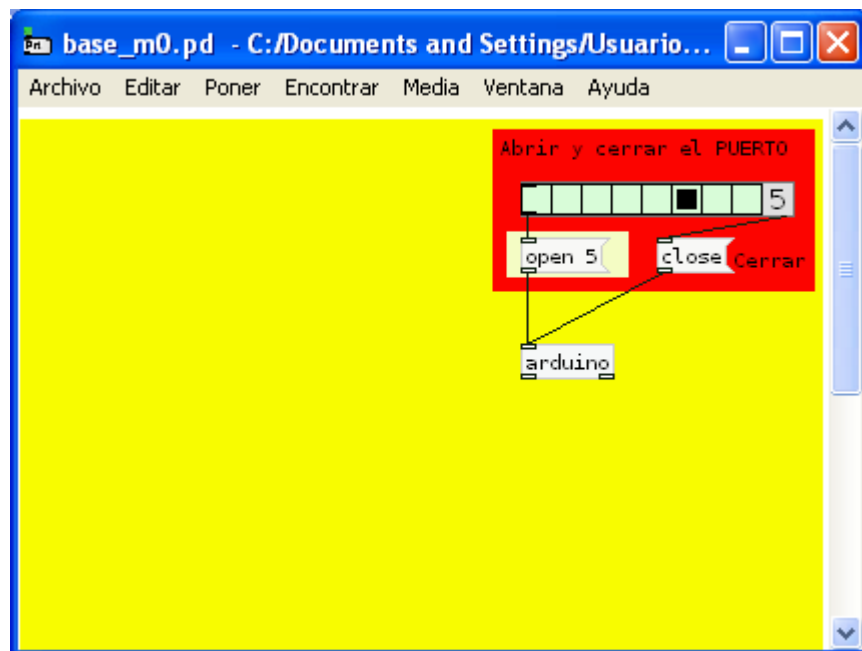
Estableceremos hasta cuatro métodos de utilización de la librería para cada uno de los cuales mostraremos una serie de ejemplos. La idea es facilitar el trabajo trabajar a la vez que flexibilizar su uso. Para cada uno de estos tres métodos he creado un fichero de base para cada uno (**base_m0.pd**, **base_m1.pd**, **base_m2.pd** y **base_m3.pd**) que son de solo lectura y que cuando queramos realizar algún ejemplo solo tenemos que abrir, realizar el alambrado y después guardad con el nombre que queramos darle. De este modo facilito algunas tareas repetitivas como es la selección del canal de comunicación o la designación de entradas salidas. En los ficheros que se adjuntan a este manual he colocado los ejemplos en carpetas separadas con los nombres metodo0, metodo1, metodo2, metodo3. En todas ellas he colado los ficheros **arduino.pd** y **arduino-help.pd** porque son necesarios para cualquier aplicación que hagamos ya que serán invocados por nuestra aplicación. En alguna carpeta pueden aparecer otros ficheros que contienen librerías con objetos útiles que se pueden copia y pegar en nuestras aplicaciones, como por ejemplo displays de 7 segmentos.

Recuerdo que previamente a todo debemos tener cargado el firmaware Firmata en nuestra tarjeta **Arduino**, tal como ya hemos explicado anteriormente.

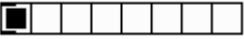
Es recomendable que ante un mal funcionamiento de nuestra aplicación pulsemos el reset de la tarjeta **Arduino**.

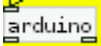
3.1. METODO DE TRABAJO 0

En este método he reducido al máximo los elementos en el escritorio de trabajo de tal manera que solo he dejado la “selección del puerto de comunicación y el propio objeto de la librería “arduino”. A continuación vemos abierto el fichero **base_m0.pd** que contiene la base para escribir en ella nuestra aplicación (recuérdese que este fichero es de solo lectura)



Fichero: *base-m0.pd*

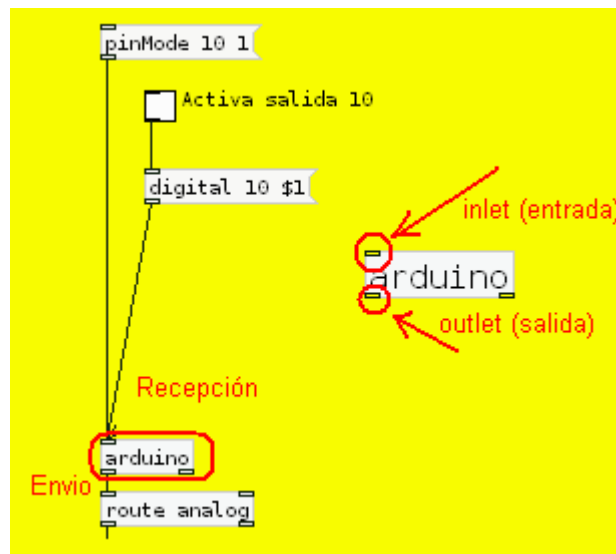
Para seleccionar el número de puerto bastará con pulsar en uno de las casillas y automáticamente se indica el puerto seleccionad a la derecha del selector y queda realizada la comunicación, si no es el puerto adecuado nos avisar de que no lo encontró. Para detener la conexión pulsaremos en el objeto “close”. Se ha contado con un máximo de 8 puertos (desde el COM0 al COM7) si fuera necesario más sería cuestión de editar el objeto “Hradio” y añadirle más 

No olvidemos que el objeto  esta asociado a un fichero **arduino.pd** que debe estar en la misma carpeta en la que estamos trabajando. He traducido parte de la ayuda de este objeto para facilitar la comprensión de su funcionalidad. La idea fundamental cosniste en enviar y recibir de este obajto la informaicon necesaria para comunicarnos con el y podr gobernar nuestra tarjeta **Arduino**.

Para la comunicación con arduino recurriremos básicamente al envío y recepción de “mensajes” a él haciendo uso de una sintaxis que iremos explicando poco a poco.

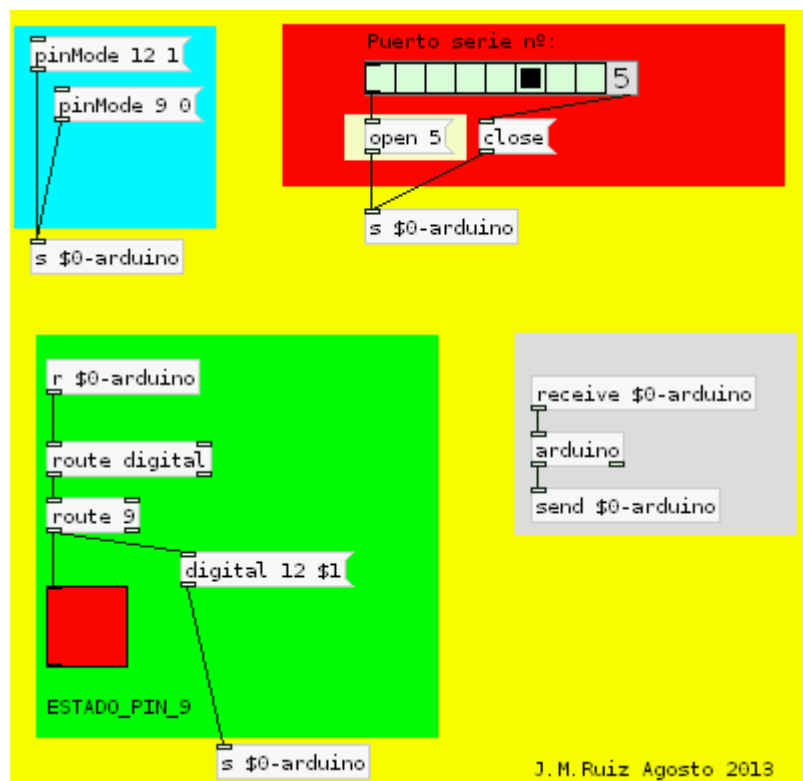
El envío se puede hacer por dos métodos distintos:

A) Envío de y recepción de mensajes mediante cableado directo con el objeto.



Los pines superiores de un objeto en la terminología de PD reciben el nombre de “inlets” y los de salida “outlets”.

B) El segundo método es utilizando los bloques de envío y recepción de datos sin cableado. Vemos en la figura un sencillo ejemplo.



Se puede ver que el objeto arduino tiene conectados un receptor y un emisor de datos: recibe \$0-arduino y send \$0-arduino que son los que reciben y emiten la información.

Vemos que en cualquier lugar de nuestro esquema de trabajo podemos colocar bloque que envía la información a arduino mediante los correspondientes bloque se emisión o recepción según sea el caso: *s \$0-arduino* y *r \$0-arduino*. Hemos puesto *r* y *s* en lugar de *receive* o *send* porque en la sintaxis de PD son equivalentes *r* y *s* a *receive* y *send*.

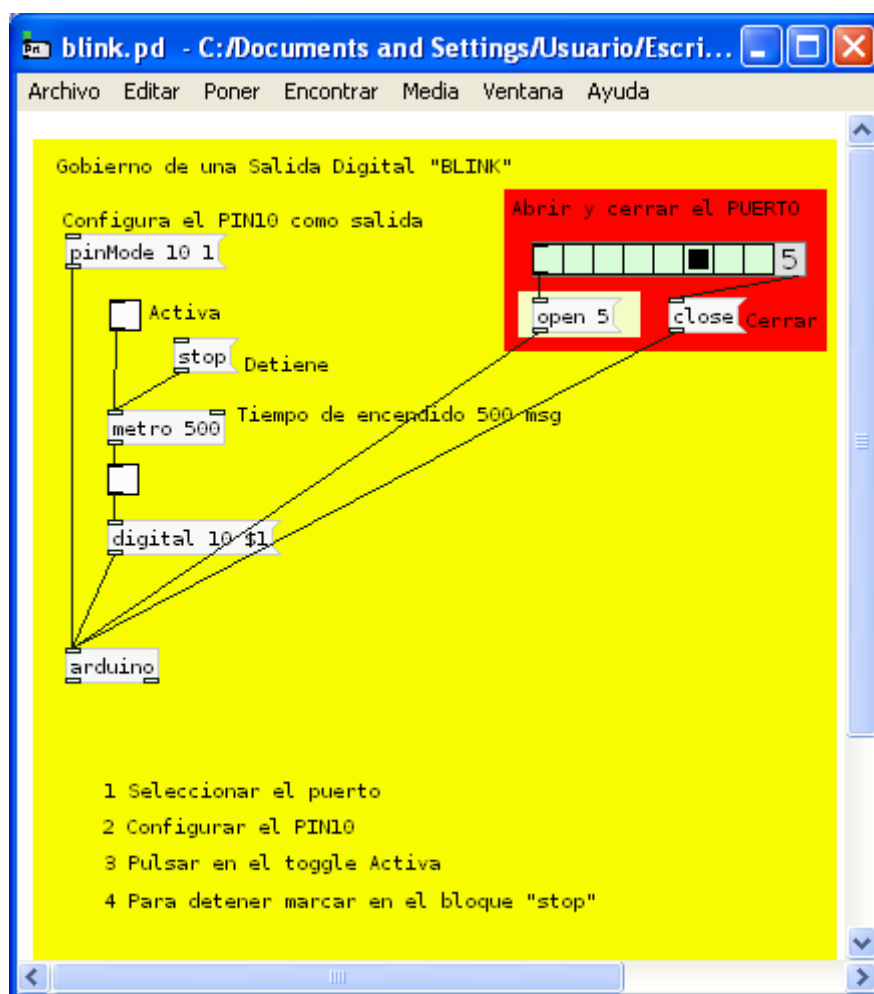
Ya sabemos lo suficiente para empezar. Ahora lo que debemos hacer es:

1. Cargar el firmware **Firmata** en la tarjeta **Arduino** con la ayuda del **IDE Arduino 1.0.5** (vale el que trae en sus librerías ya incorporado)
2. Arrancar PD y cargar el fichero **base_m0.pd** y empezar a colocar bloque y cablearlos siempre en el modo “**Edicion**” de PD. No olvides recordar en que puerto tienes colocada la tarjeta **Arduino** para luego poder seleccionarlo y por supuesto mantener conectado **Arduino** al puerto USB de tu PC.

3.1.1. Activación de una salida en modo intermitente “Blink”.

Con este primer ejemplo vamos a gobernar la salida digital PIN10 con una señal intermitente que se generará mediante el cocido bloque “metro 500” de PD que lo que hace es generar retardos en la activación y desactivación de su salida que a través de un bloque tipo “toggle” da la orden de enviar la variable \$1 (la propia entrada del bloque) al bloque de mensaje mediante la orden “digital 10 \$1” que lo que hace es poner el en el PIN10 , que ahora actúa como salida el valor que recibe a través de la variable \$1.

Previamente se debe configurar el PIN10 como una salida y esto se hace enviando el comando “pinMode 10 I” que también podría escribirse como “pinMode 10 output”



Fichero: *m01-blink.pd*

Vemos que las salidas de los bloque de envío de mensajes se han unido directamente al bloque “**arduino**”.

El procedimiento para ejecutar la aplicación el indicado en la figura.

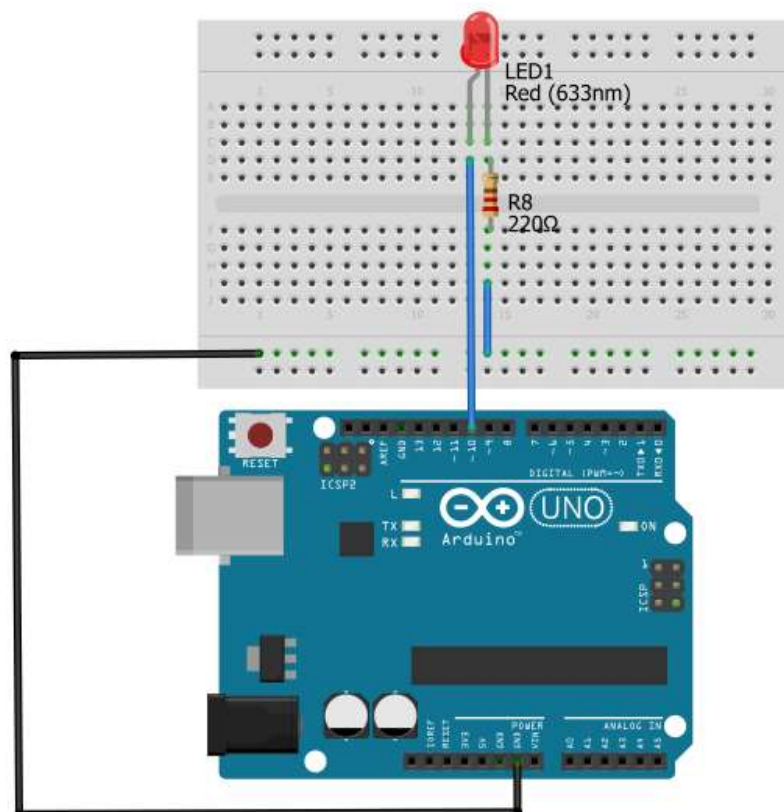
En el terminal de PD aparecerán los mensajes que nos indican que el puerto se abre. Es posible que aparezca algún mensaje de no reconocimiento de instrucción llegada al puerto pero esto es irrelevante y se debe a la sincronización de eventos en el sistema.

Bloque de instrucción usados:

El bloque sub-patch de gobierno de puerto que ya viene preestablecido en el fichero `base_m0`

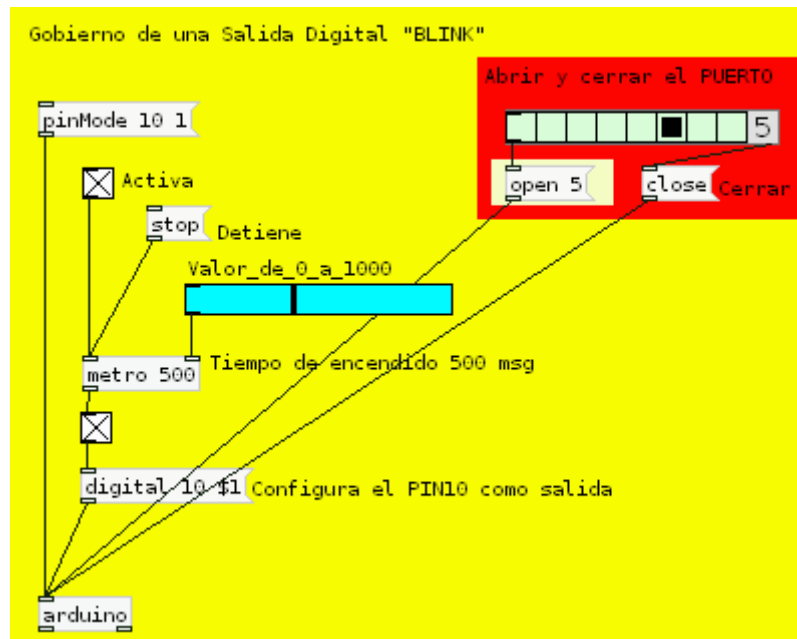
arduino:	Bloque del objeto de librería Pduino
mensaje: <i>pinMode 10 1</i>	Programa el pin 10 como salida
toggle:	Bloque tipo interruptor
stop:	Bloque tipo mensaje que detiene el envío de pulsos
metro: <i>metro 500</i>	Genera impulsos de 500 ms de duración.
mensaje: <i>digital 10 \$1</i>	Envía las ordenes de activación/desactivación de la salida PIN10

Esquema de montaje:



Variación sugerida:

Puedes probar a modificar el valor 400 de la instrucción metro y podrás notar como varia la cadencia de impulsos. También es posible colocar un sencillo slider y poder variar a tu gusto la velocidad.

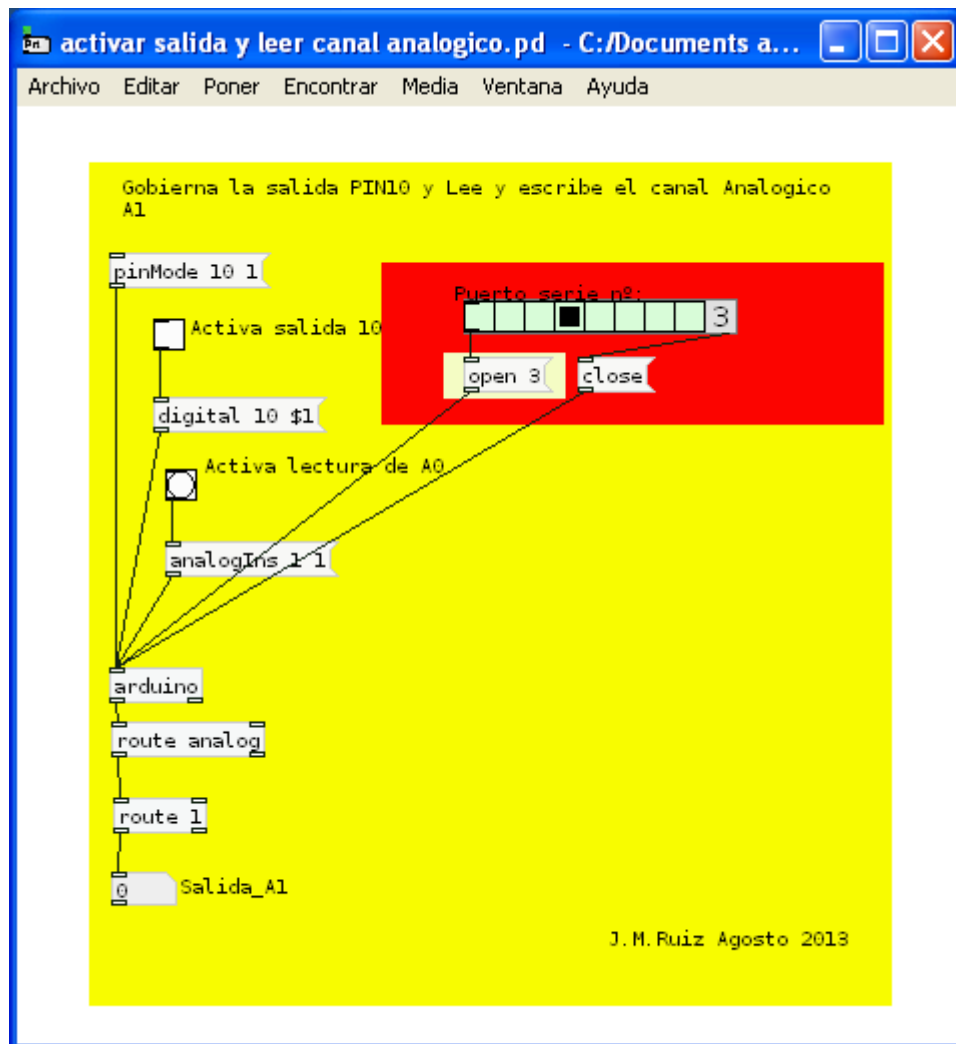


Fichero: *m01-blink tiempo variable*.pd

3.1.2. Activación de una salida y lectura de un canal analógico

Veamos un ejemplo sencillo de manejo de **Arduino** mediante el envío y recepción de comandos a través del bloque de función (patch de pd) **Arduino**.

Se trata de gobernar el PIN10 como salida digital y a la vez poder leer el canal analógico de entrada A1



Fichero *m02-activar salida y leer canal analógico.pd*

Cargamos este ejemplo y lo probamos.

La manera de actuar sería la siguiente:

1. Seleccionamos el puerto serie
2. Hacemos clic sobre el bloque **mensaje** *pinMode 10 1* para decirle a **Arduino** que el *PIN 10* será una salida
3. Pulsamos sobre el bloque **mensaje** *analogIn 1 1* para configurar el canal *a1* como salida analógica

4. Pulsamos varias veces sobre el botón y vemos como cambia el valor de la salida PIN10
5. Vemos también que en el bloque de salida numérica de abajo aparece el valor del cana (mide de 0 a 1).

Para indicar el canal de conexión de **arduino** lo hacemos mediante el bloque **mensaje** *open x* (*x* es el numero de canal seleccionado mediante el bloque **Hradio**

Con el bloque **mensaje** *close* cerramos el puerto.

Instrucciones:

mensaje: *pinMode 10 1* Programa el pin 10 como salida

mensaje: *digital 10 \$1* Envía un 1 a la salida digital y cuando pulsamos el botón envía el estado de este

mensaje: *analogIns 1 1* Este comando habilita el canal analógico A1 para ser leído

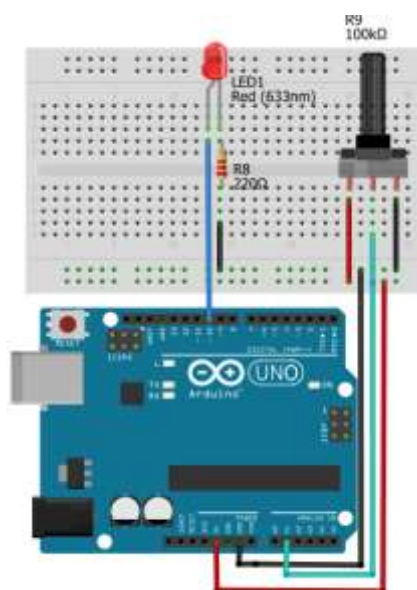
Este es el bloque librería de **Arduino** No es necesario ponerle ningún parámetro, aunque se le podría poner el numero de puerto y la velocidad de comunicación.

route analog: Bloque que lee los valores de los canales analógicos

Selecciona el canal 1 del conjunto que entrega el bloque anterior.

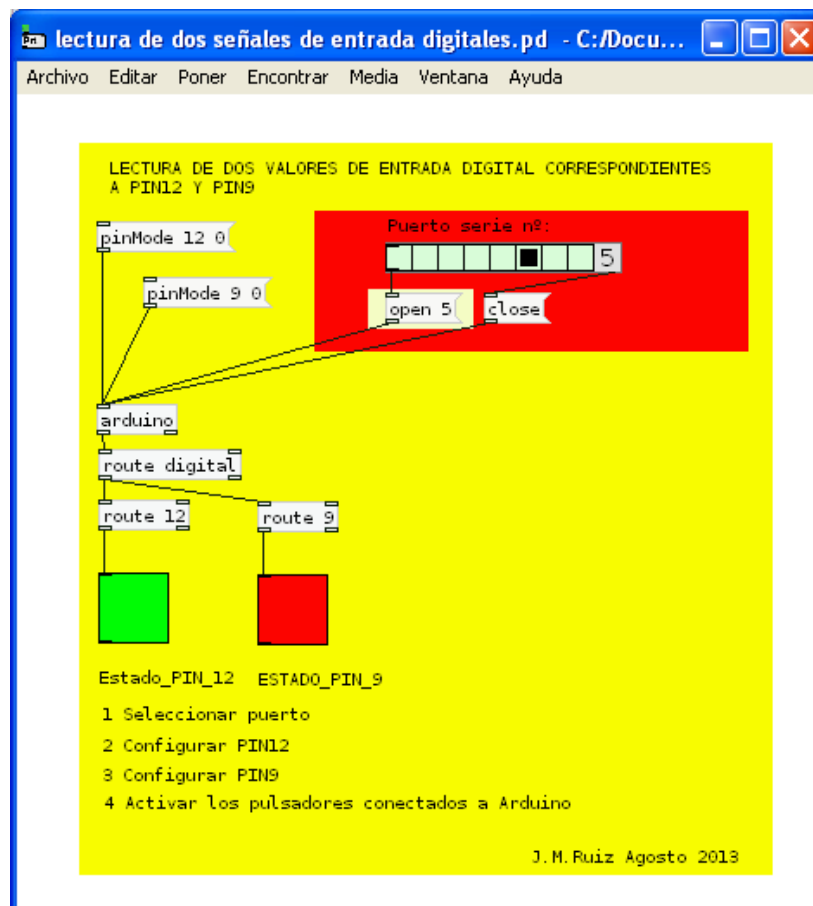
Si queremos modificar el contenido de una de las instrucciones **mensaje** solo debemos poner PD en modo “edición” y cambia lo que deseemos

Esquema de montaje



3.1.3. Lectura de dos canales de entrada digital a la vez.

En este ejemplo deseamos poder leer dos canales de entrada digital de **Arduino** y mostrar su contenido en la pantalla. PIN12 y PIN9.



Fichero: *m03-lectura dos entrada digitales.pd*

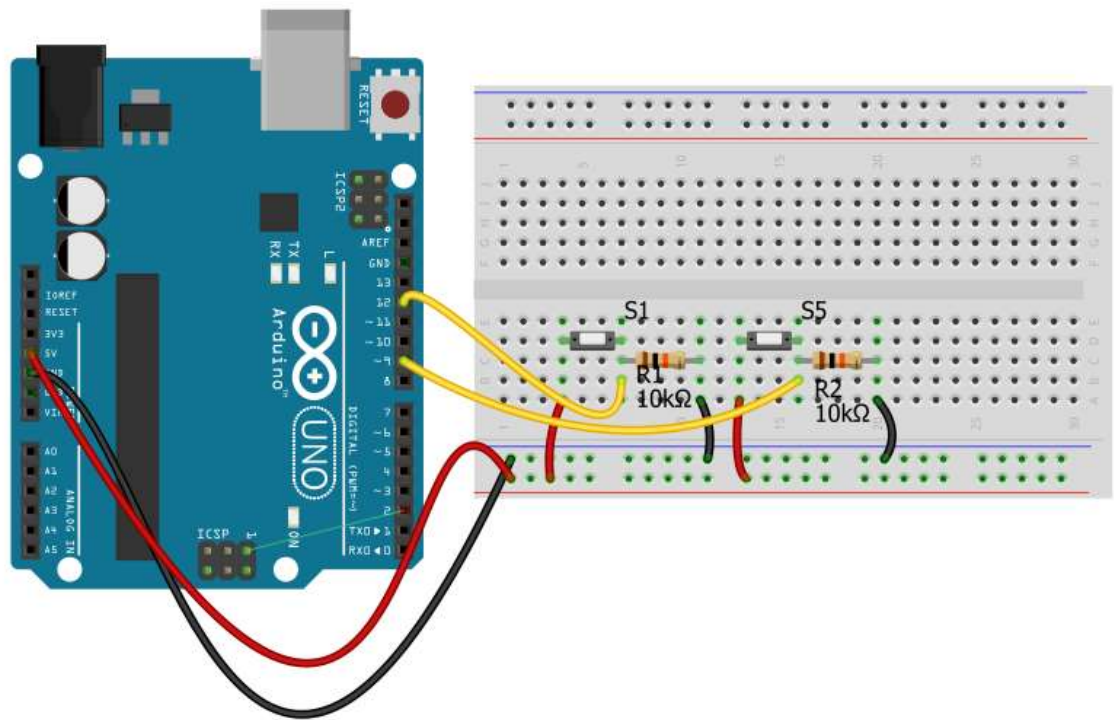
Con esta aplicación queremos demostrar como es posible leer 1 o más (hasta los 11) canales operativos de entrada digital de **Arduino**. Vemos que lo único que debemos cuidar es de realizar la extracción del canal a leer mediante la función “*route X*” en donde *X* representa el número de canal (numero de PIN). Esta función recibe toda la trama de canales que le envía la función “*route digital*” recogida del bloque **arduino**.

En la parte de “inlets” del bloque **arduino** lo que hacemos es cablear las ordenes de configuración de los canales digitales que quiero leer “*pinMode 12 0*” y “*pinMode 9 0*”.

En la figura se recuerda el orden de actuación en la ejecución del patch PD.

Cuando se lee un 1 se muestra  y cuando se lee un cero 

Esquema de montaje:

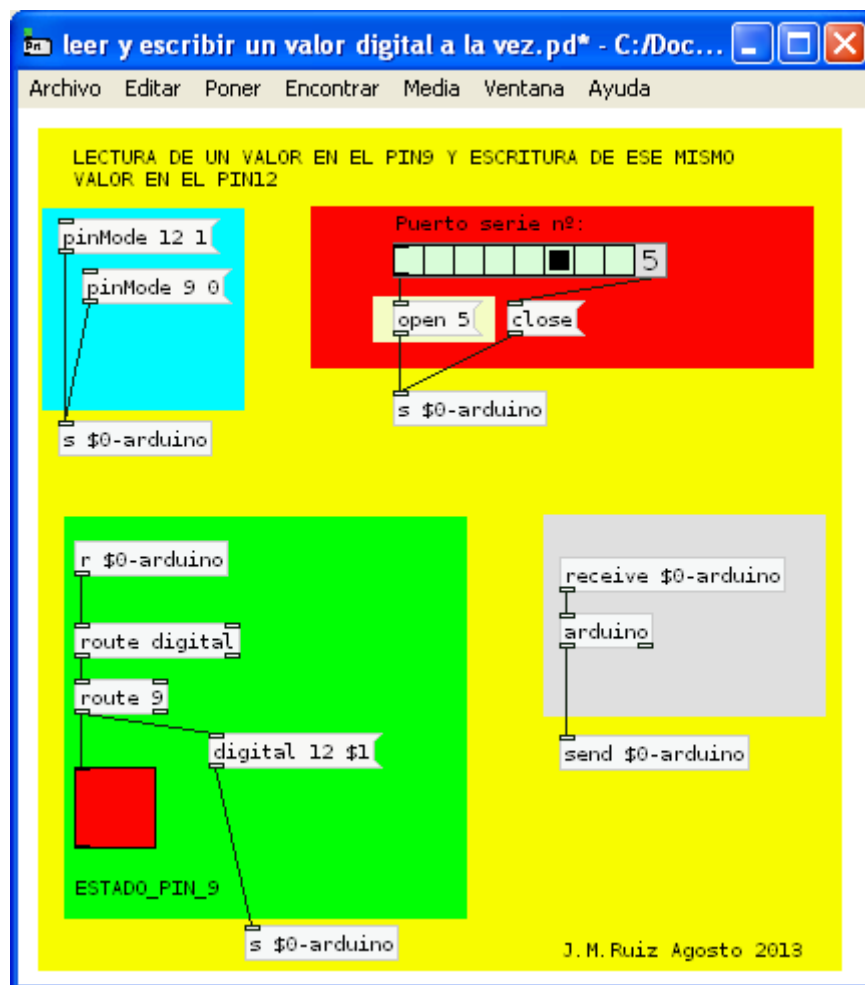


3.1.4. Leer y escribir un mismo valor digital

Con este ejemplo vamos a realizar la realimentación de una salida con una entrada de **Arduino**. El valor del **PIN19** lo trasladaremos al **PIN12**, es decir que cuando, mediante un pulsador activemos la entrada **PIN12** a la vez se activara la salida **PIN9**.

PIN9 → PIN12

Quiero además probar en este ejemplo la posibilidad de enviar y recibir datos del objeto arduino haciendo uso de las funciones **receive (r)** y **send (s)**.



Fichero: *m04-leer y escribir señal digital a la vez.pd*

He enmarcado en “lienzo” de color cada bloque de acciones para facilitar la explicación.

El bloque de color rojo alberga, como ya sabemos, la función de activación y desactivación del canal.

En el bloque azul se ha colocado la configuración de los canales mediante dos bloques de mensaje: *pinMode 12 1* (ponemos 1 pero podríamos poner output que indica salida) y *pinMode 9 0* (ponemos 0 pero podríamos poner input que indica entrada) que definen la forma de trabajo de los pines implicados. Vemos que las salidas de estos bloques que contienen los comandos de configuración van a un bloque que envía la información “*s \$0-arduino*” este bloque se podría escribir también como “*send \$0-arduino*” vemos que su parámetros son **\$0** la variable que contiene el mensaje propiamente dicho y el nombre del bloque al que va dirigido este mensaje, **arduino**. No sería preceptivo poner **\$0**

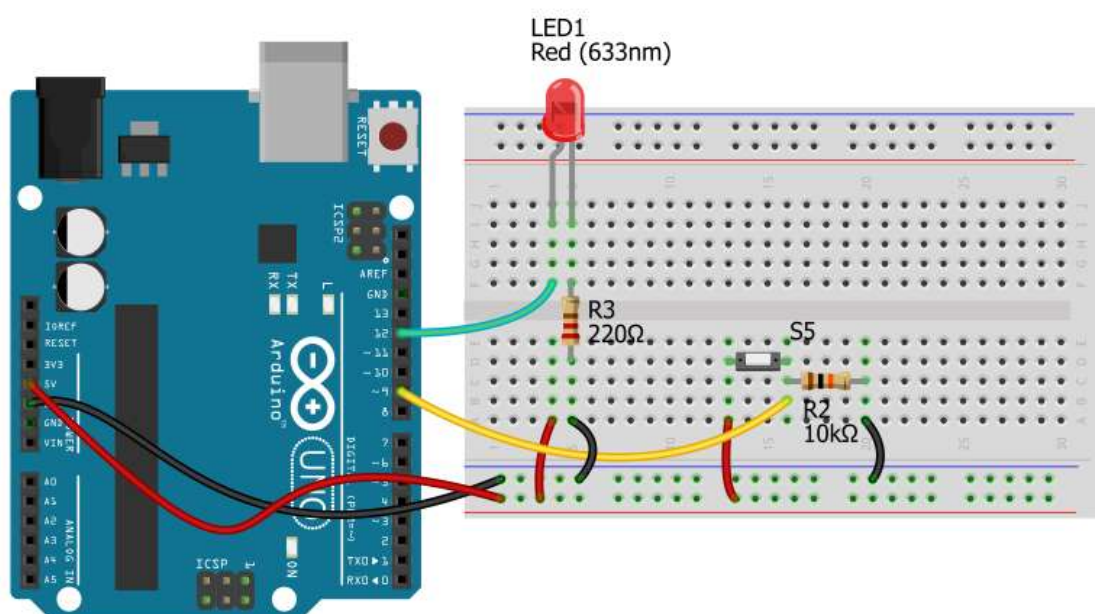
El bloque verde contiene los elementos necesarios para realizar la lectura del valor de entrada **PIN9**. Estos bloques son: El bloque “*r \$0-arduino*” que lee el mensaje de **arduino**, “*route digital*” que separa los valores digitales, “*route 9*” que separa de los valores digitales el valor del **PIN9**. Vemos que este bloque contiene también la función mensaje “*digital 12 \$1*” que recoge el estado de la variable leída (PIN9) y lo envía a **arduino** para activar o desactivar según el caso la salida **PIN12** a través de un bloque de envío e datos “*s \$0-arduino*”.

El bloque gris contiene el bloque principal de la librería **Pduino arduino** y a el se han unido los bloque de envío y recepción de datos: *receive \$0-arduino* y *send \$0-arduino*.

Se puede realizar la prueba sin olvidar el protocolo de siempre:

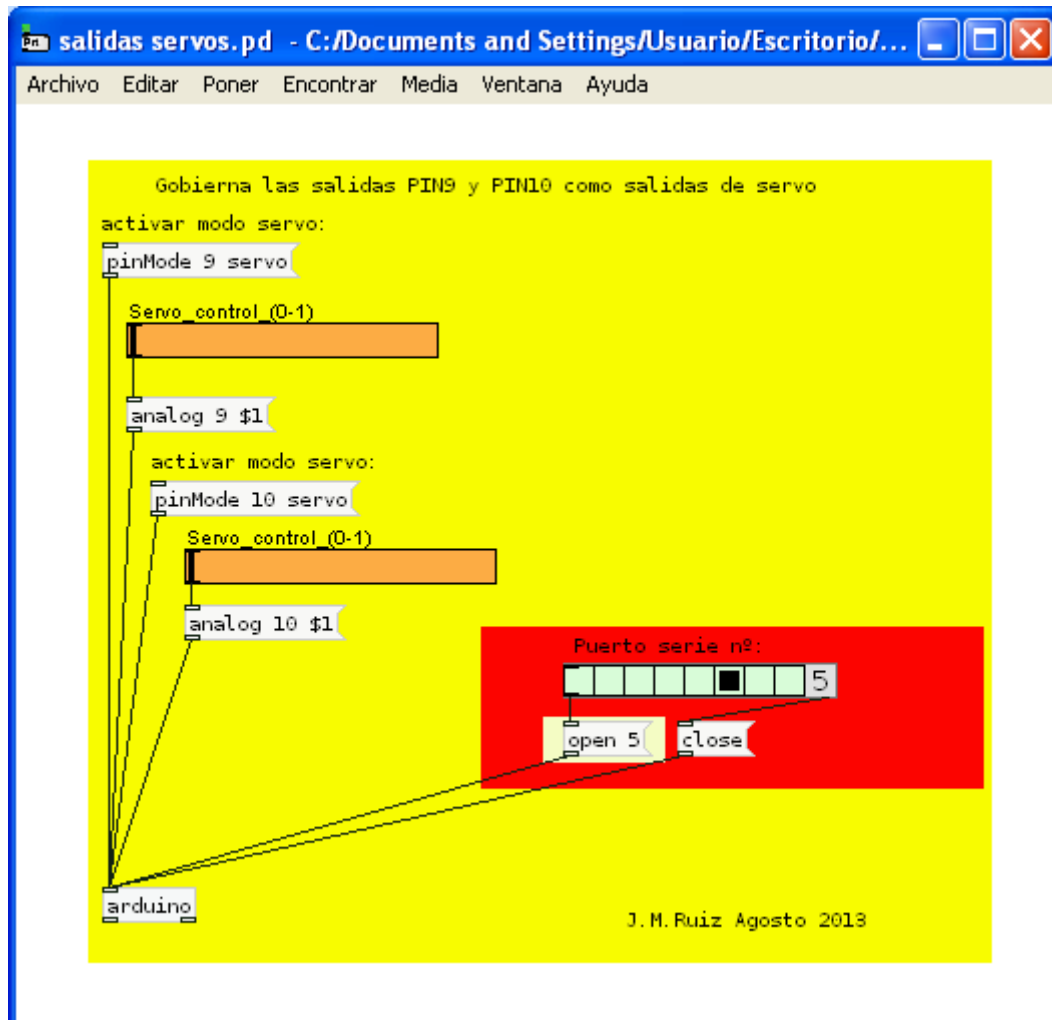
1. Seleccionar el canal
2. Configurar el modo de trabajo de los pines
3. Realizar las manipulaciones sobre el pulsador del PIN9 y observar el comportamiento del sistema.

Esquema de montaje:



3.1.5. Gobierno de dos servos conectados a las salidas PIN9 y PIN10

Vamos a controlar dos servos colocados en los pines digitales PIN9 y PIN10.

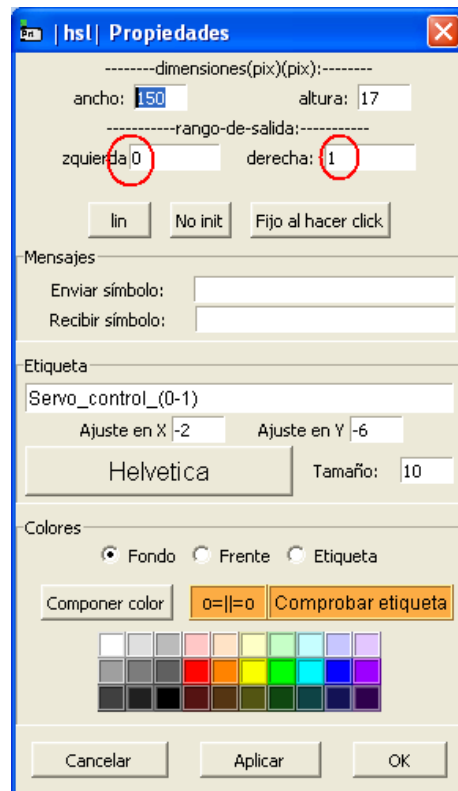


Fichero: *m05-servos.pd*

El montaje es muy sencillo. Se trata de configurar los pines mencionados en modo salida servo: *pinMode 9 servo* y *pinMode 10 servo*

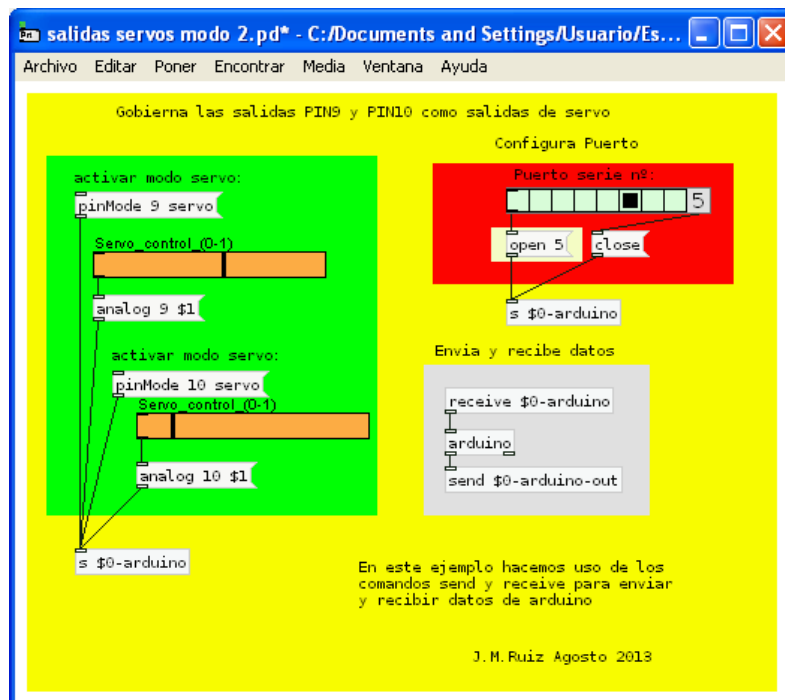
Posteriormente, para facilitar la simulación colocamos dos sliders con valores *min=0* y *max=1* que generarán la señal que se enviara mediante los mensajes *analog 9 \$1* y *analog 10 \$1* al bloque **arduino**

Para cambiar los parámetros de los objetos **hslider** colocados nos ponemos con el ratón sobre ellos y pulsamos el botón derecho abriéndose una venta de parámetros como la indicada en la figura.



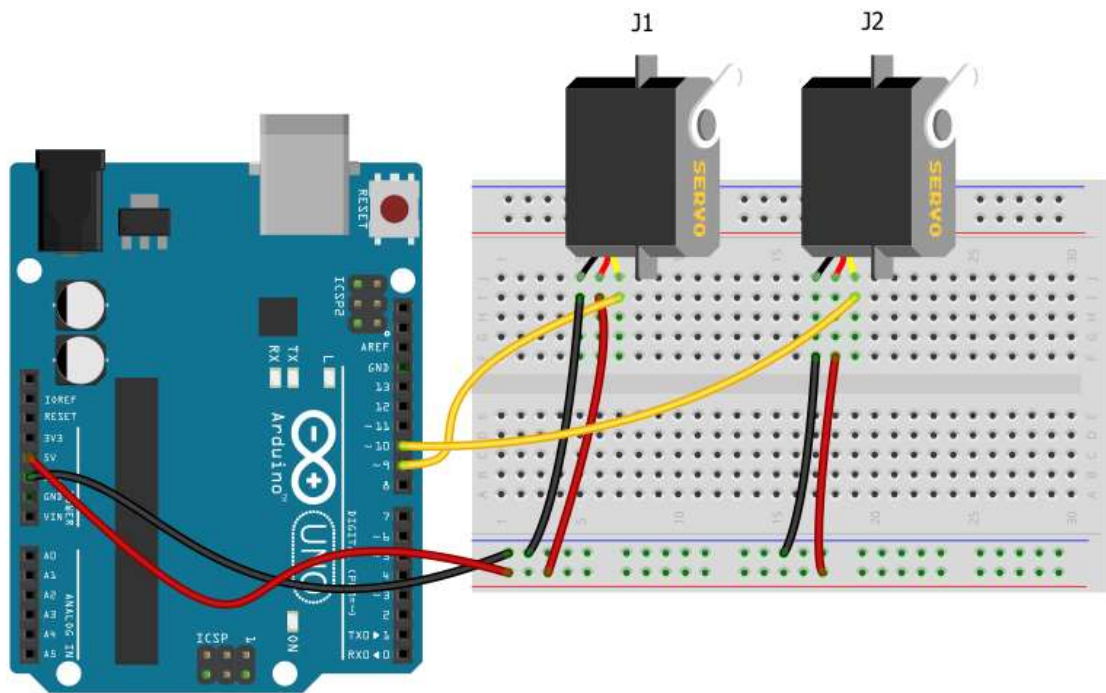
Ventana de parámetros del bloque **hslider**

Una posible variación que sugiero en este ejemplo es realizar la conexión de bloques con la ayuda de las funciones **receive** y **send**



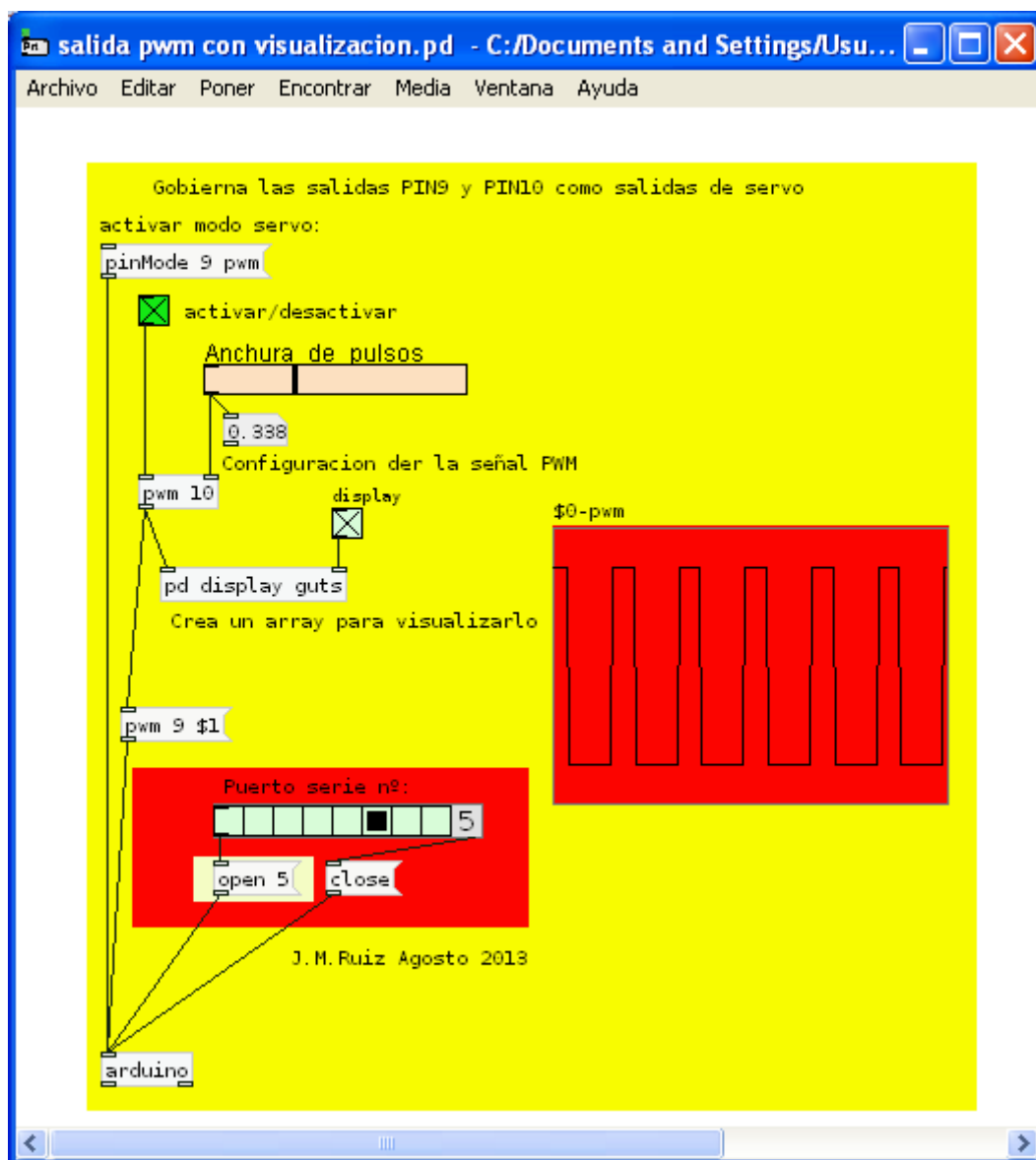
Fichero *m05-servos modo 2.pd*

Esquema de montaje:




3.1.6. Salida PWM con visualización gráfica.

En este ejercicio vamos a gobernar la salida PIN9 en modo PWM y para ello nos valdremos de un slider con el que variar la anchura de pulsos. Que como sabemos es el parámetro principal del sistema pwm. Además utilizaremos un sencillo visualizador de señal que nos permitirá comprobar la naturaleza de la señal y verificar como se amplía o disminuye la anchura de pulso en función de la señal que entregue el slider. Se ha utilizado para esto un patch recogido en el foro de PD que nos viene muy bien.

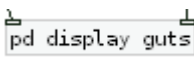


Fichero: *m06-salida pwm con visualización.pd*

En esta aplicación tenemos dos sub-patch que son muy útiles y que no explicare dado que esa explicación se escapa al ámbito general de este manual. Son patch creados por miembros del foro de PD, algunos de ellos van firmados por su autores y todos ellos se deben usar bajo licencia GNU.

Sub-patch *pwm X* 

Este se encarga de generar la señal **PWM** propiamente dicha para enviarla a **arduino**.

Sub-Patch *pd display guts* 

Se encarga de mostrar la señal en el grafico *\$0-pwm*.

El resto de bloques de función son del tipo **mensaje**:

activar modo servo:
 

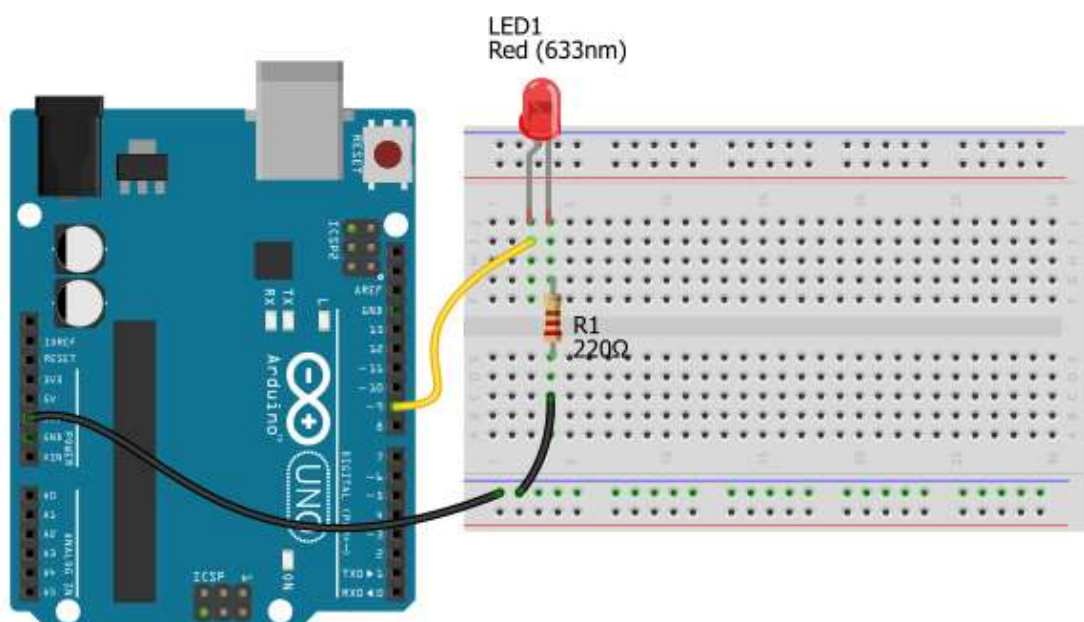
Que sirven para configurar el **PIN9** como salida **PWM** y la otra para enviar la señal que genera el bloque *pwm 10*

La prueba de este ejemplo es muy sencilla, bastará con:

1. Abrir el puerto
2. Configurar la salida **PIN9** como **PWM**
3. Pulsamos el botón **toggle** Activar/desactivar
4. Activar la visualización del grafico (pulsando el bloque **toggle display**)

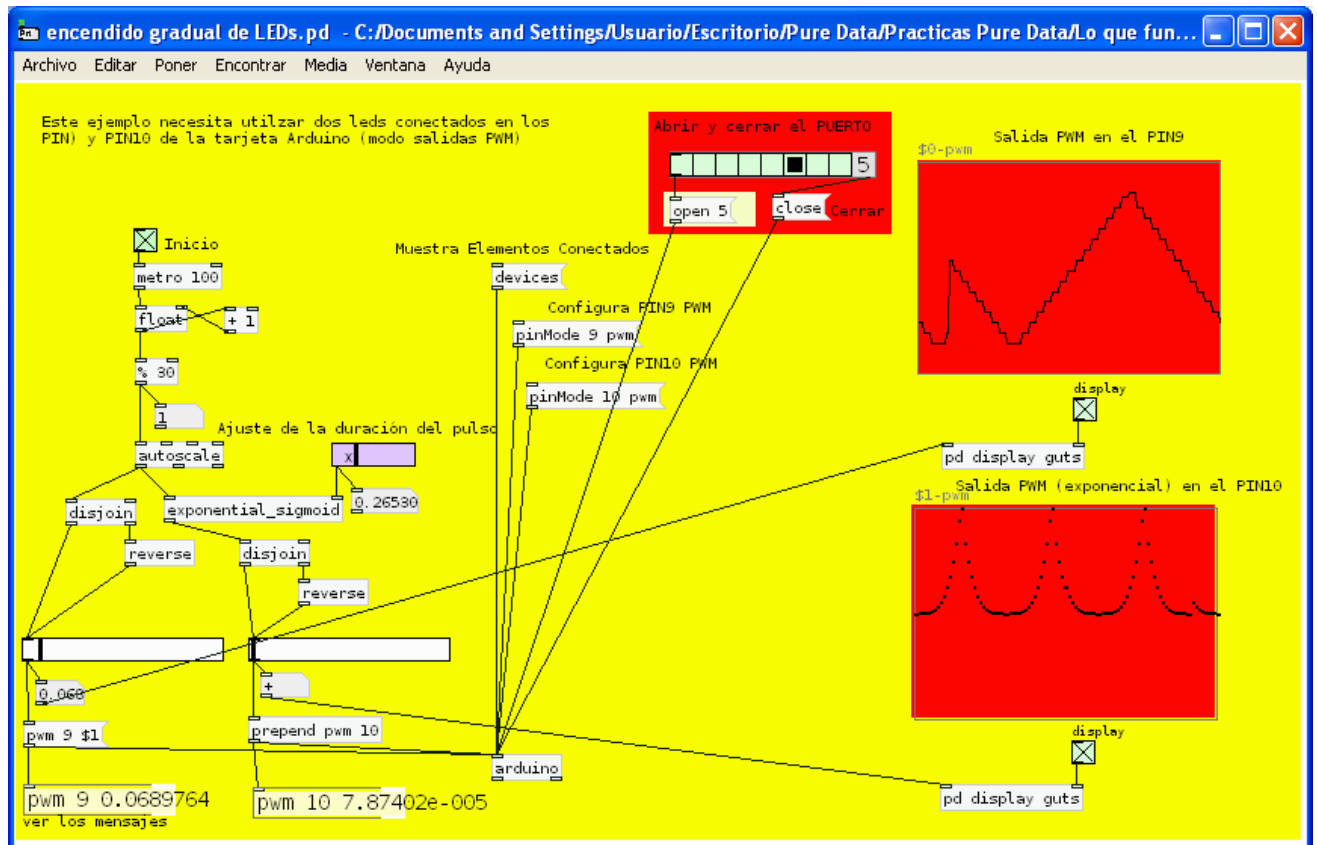
Seguidamente mover el slider “anchura de pulsos” y observar el comportamiento tanto en el grafico como en la salida física de **Arduino** en la que podemos colocar a modo de test un diodo Led.

Esquema de montaje:



Encendido gradual de dos diodos leds.

Este ejemplo, igual que el anterior está relacionado con el gobierno de salidas PWM pero en este caso de distinto tipo una ira al PIN9 y la otra al PIN10.



Fichero: *m07-encendido gradual de LEDs.pd*

Se han utilizado básicamente dos bloque de creación externa a PD que son los que indicamos a continuación:

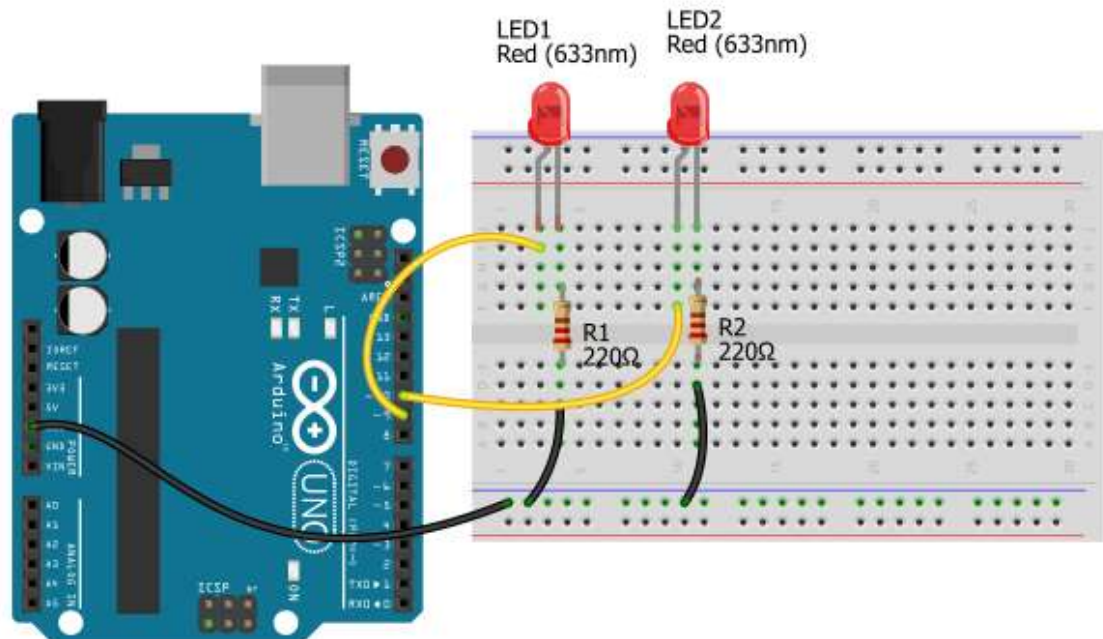
Sub-patch *exponential_sigmoid*: Genera una función exponencial tipo sigmoid

`exponential_sigmoid`

Sub-patch *pd display guts*: Genera un grafico de los valores que le ponemos en su entrada

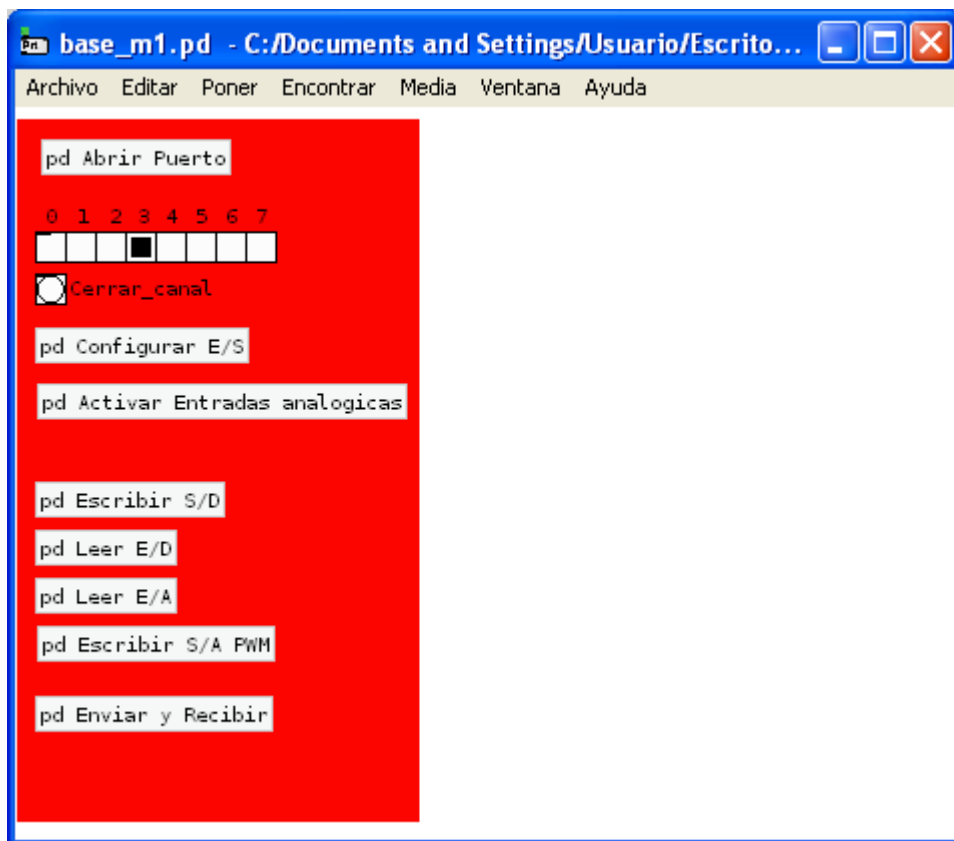
`pd display guts`

Esquema de montaje:



METODO DE TRABAJO 1

En este método he incluido ciertas utilidades integradas en forma de sub-patch en el modelo de trabajo con el fin de facilitar la configuración y la versatilidad en el trabajo. A continuación vemos abierto el fichero **base_m1.pd** que contiene la base para escribir en ella nuestra aplicación (recuérdese que este fichero es de solo lectura)



Fichero: *base_m1.pd*

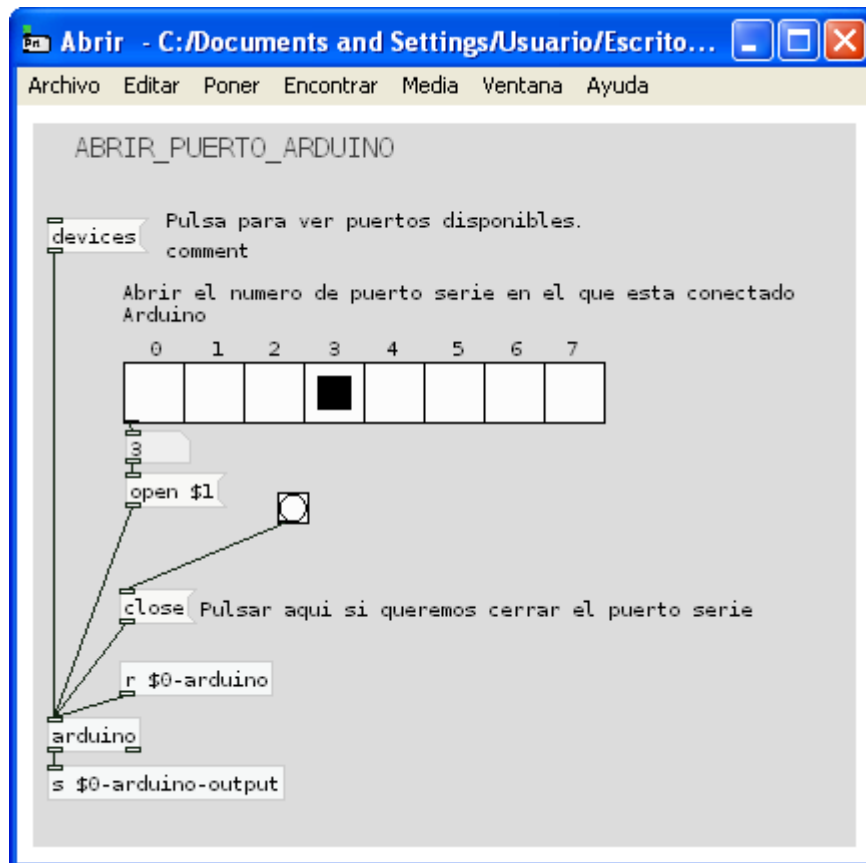
Elementos integrados en este modelo de trabajo:

Apertura de puerto:

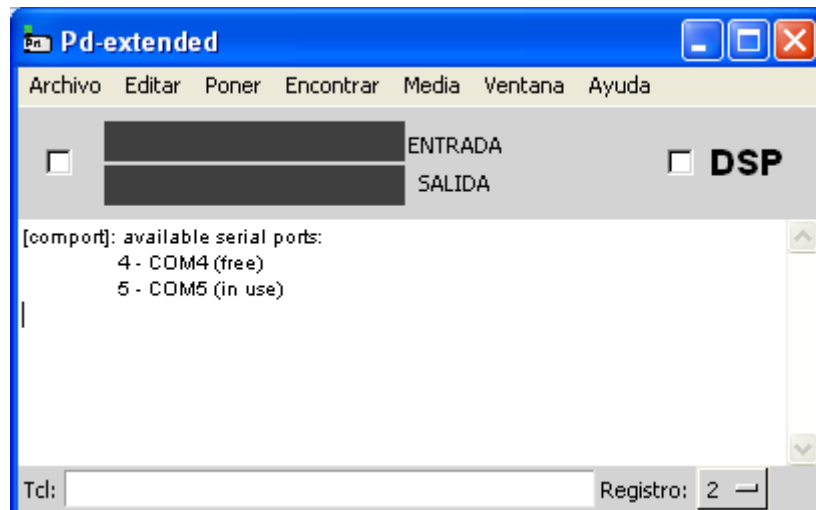
Se incluye un sub-patch en el que vemos los bloques necesarios para realizar la configuración selección del puerto.

Debajo de este bloque he colocado de manera resumida el cajetín de selección de puerto y bel botón de cierre de puerto.

Este patch ya lo hemos utilizado de alguna manera aquí lo único que he pretendido es dejarlo más claro con el fin de facilitar el aprendizaje de quienes lean este manual.



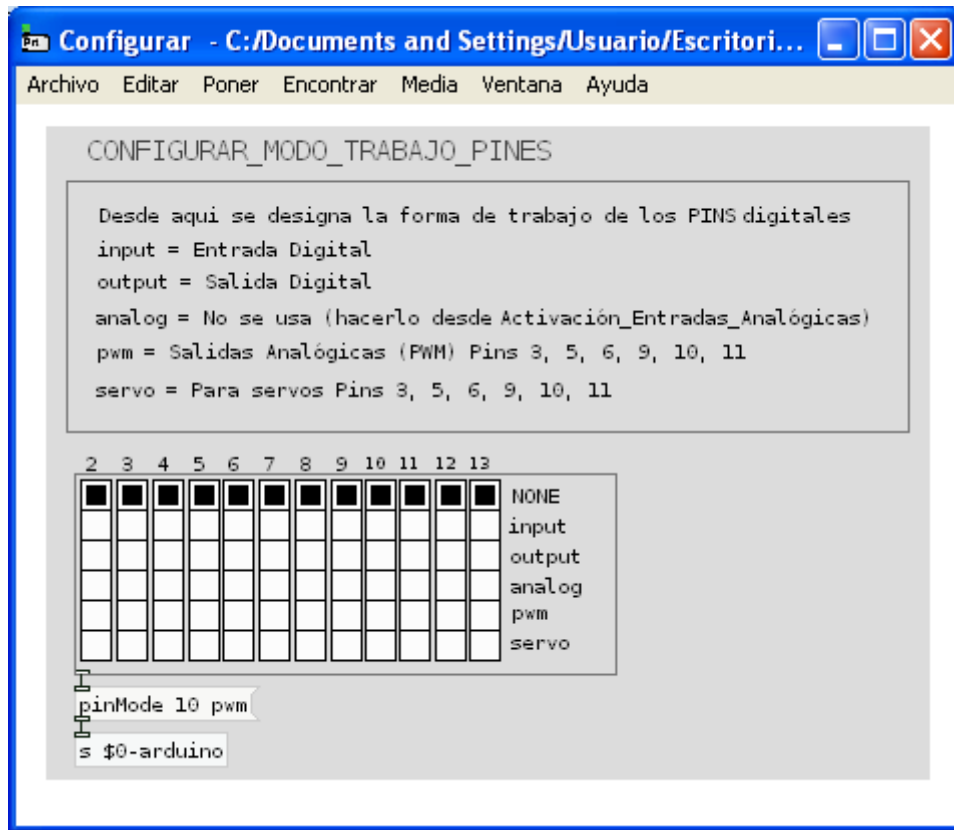
Al pulsar sobre el mensaje “devices” nos lista en el terminal de salida de PD los elementos conectados.



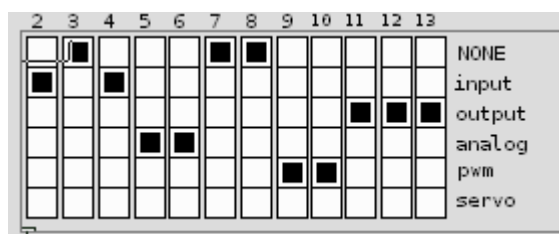
Configuración de modo de trabajo de los pines

En este caso vemos que con este sub-patch incluido en el entorno de trabajo podemos realizar la programación del modo de trabajo de los pines de **Arduino** (pines digitales del PIN2 al PIN13).

Se ha colocado un conjunto de elementos **Hradio** (cajas con varias casillas) mediante las cuales se puede marcar el modo de trabajo de cada pin. Las casillas verticales pertenecen a los modos de trabajo programables de cada uno de los pins (*NADA*, *entrada*, *salida*, *analógica*, *pwm*, *servo*). Es importante tener en cuenta que las señales analógicas, pwm o servo solo podrán ser los pines **3,5,6,9,10,11** en el modelo **Arduino UNO**



La selección que realicemos se enviara a través de un bloque de función **mensaje** *pinMode X* y que se envía mediante un bloque del tipo **send** *s \$0-arduino*.

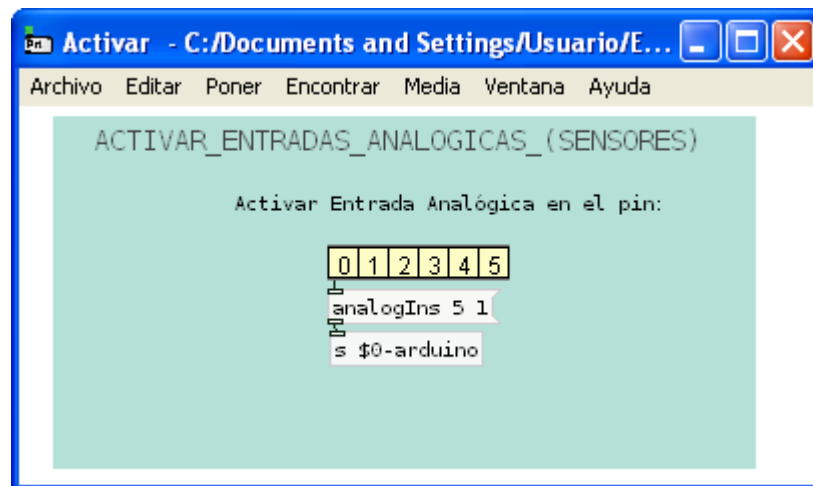


En este ejemplos se han programado:

- PIN2 y 4 Entradas
- PIN5 y 6 Salidas analógicas
- PIN9 y 10 Salidas pwm
- PIN11, 12 y 13 Salidas digitales
- Pin7 y 8 No se han programado (por defecto serán salidas digitales)

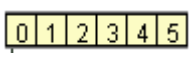
Activar entradas analógicas

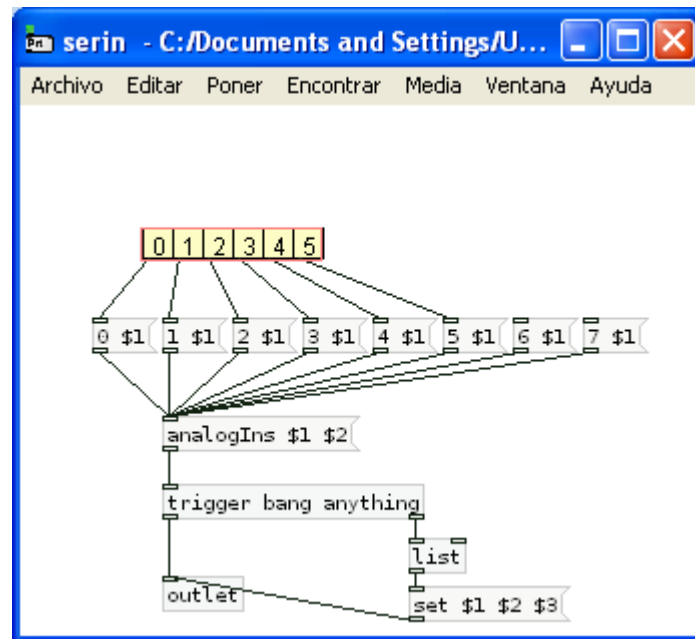
En este caso desde el **sub-patch** podemos seleccionar las salidas analógicas que deseamos sean conducidas a la salida del bloque arduino. No siempre necesitaremos disponer de las seis salidas con las que cuenta **Arduino a0** a la **a5**.



Bastará marcar sobre el número de salida y automáticamente podremos rescatar el valor en PD. Para ello hacemos uso del bloque de envío de mensajes que ya hemos

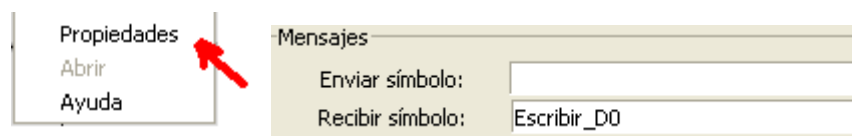
explicado  y del bloque **mensaje** que recoge el valor del selector

El selector  también es un **sub-patch** que mostramos a continuación y que se encarga de crear el “telegrama” de envío.



Escribir valores en las salidas digitales

Con este sub-patch podemos escribir valores digitales (0 o 1) en aquellas salidas que están configuradas como tales. Se han colocado dobles bloques de tipo toggle en cada salida porque uno de ellos se encarga de recibir los valores que pongamos desde cualquier lugar simplemente con la opción "Recibir símbolo" en cuya casilla pondremos el valor de la variable que podrá recibir. Esta configuración se realiza desde la opción "Configurar" que estando en modo edición se activa poniendo el ratón sobre el bloque y pulsando el botón "Propiedades"

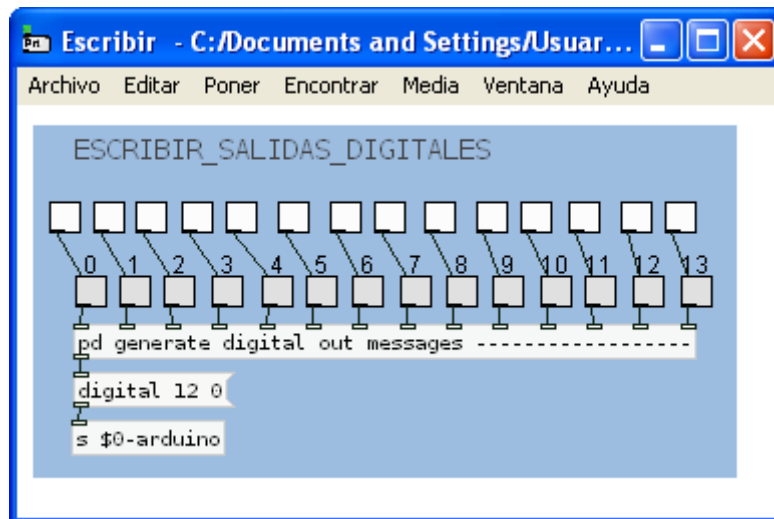


Es oportuno que tomes nota de cómo se denominaran las salidas digitales en lo sucesivo, para cuando trabajemos con este modo de trabajo recogido en el fichero base_m1.pd.

Nombramiento de las señales de salida Digital:

Escribir_D0, Escribir_D1, Escribir_D2, Escribir_D3..., Escribir_D13

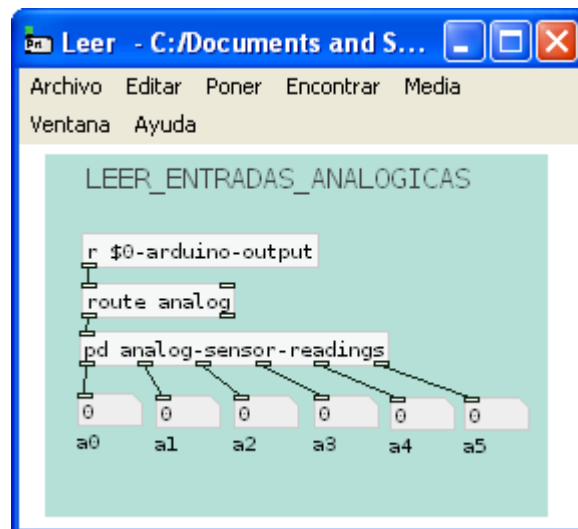
pero no olvidemos que las señales de los **PIN0** y **PIN1** no se utilizarán. Por lo tanto solo están útiles los pines **PIN2** al **PIN13**



Leer valores de entradas analógicas

Cuando deseamos leer los valores analógicos de los canales de entrada o lectura de **Arduino** lo podemos hacer abriendo este sub-patch. No olvidemos que para leer previamente debemos haber seleccionado los canales a leer con la opción que nos brinda el sub-patch anteriormente explicado de “Activar entradas analógicas”.

Realmente se trata de leer a través de un bloque “**receive**” los datos y mediante un enrutador de estos sacarlos en salidas independientes

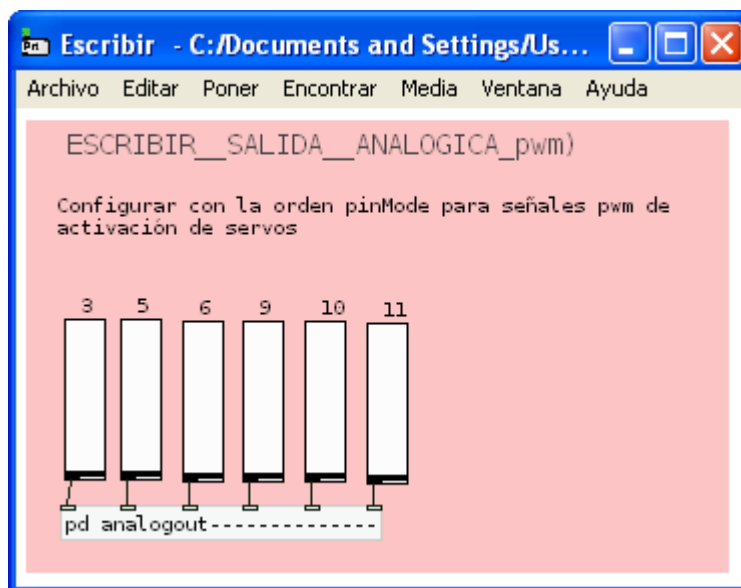


En este caso, como hemos hecho en el anterior debemos tomar nota de cual será el nombre con el que podremos invocar a los 6 canales analógicos. Pues debes tomar nota de ello

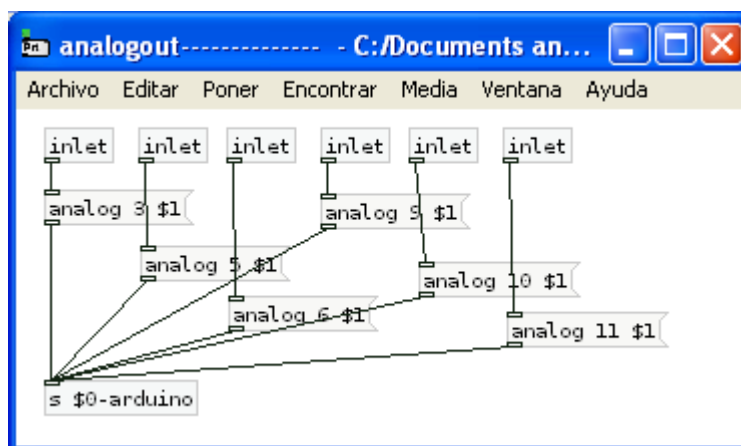
Los nombres de los canales analógicos de entrada de **Arduino** son:

a0, a1, a2, a3, a4, a5

Escribir salidas analógicas pwm



A la hora de escribir datos sobre las salidas de tipo **PWM** de **arduino** lo podemos hacer directamente desde este **sub-patch**. Básicamente se trata de utilizar un **sub-patch** de recogida de datos *pd analog -----* que mostramos en la figura siguiente y que a la vista resulta fácil de interpretar.



Es **MUY IMPOTANTE** que no se olvide que de todos los pines digitales de arduino los que tienen la capacidad de ser salidas analógicas del tipo PWM son

PIN3, PIN5, PIN6, PIN9, PIN10 y PIN11

La manera de nombrar las señales de este tipo es:

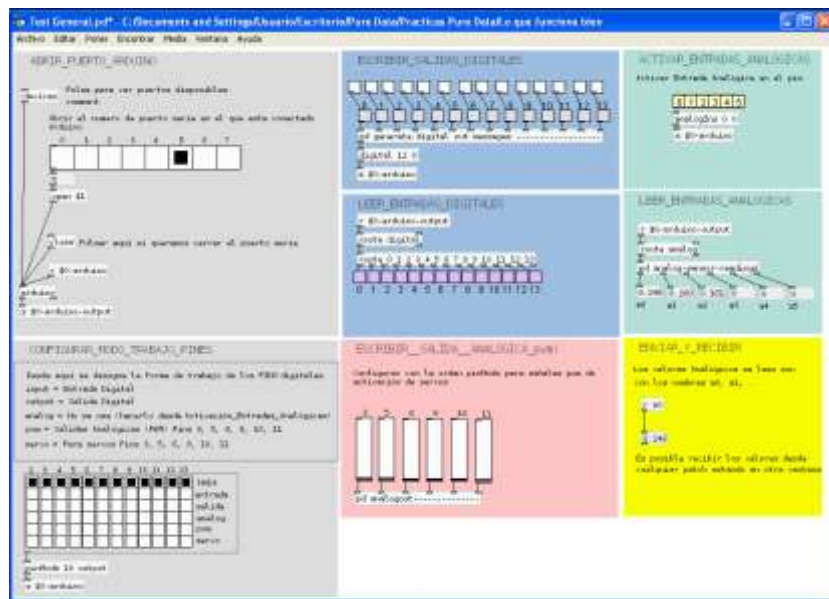
PWM3, PWM5, PWM6, PWM9, PWM10 y PWM11,

Resumen de todos los elementos arduino-test

He creado un patch en PD para resumir todos estos bloques al que he llamado:

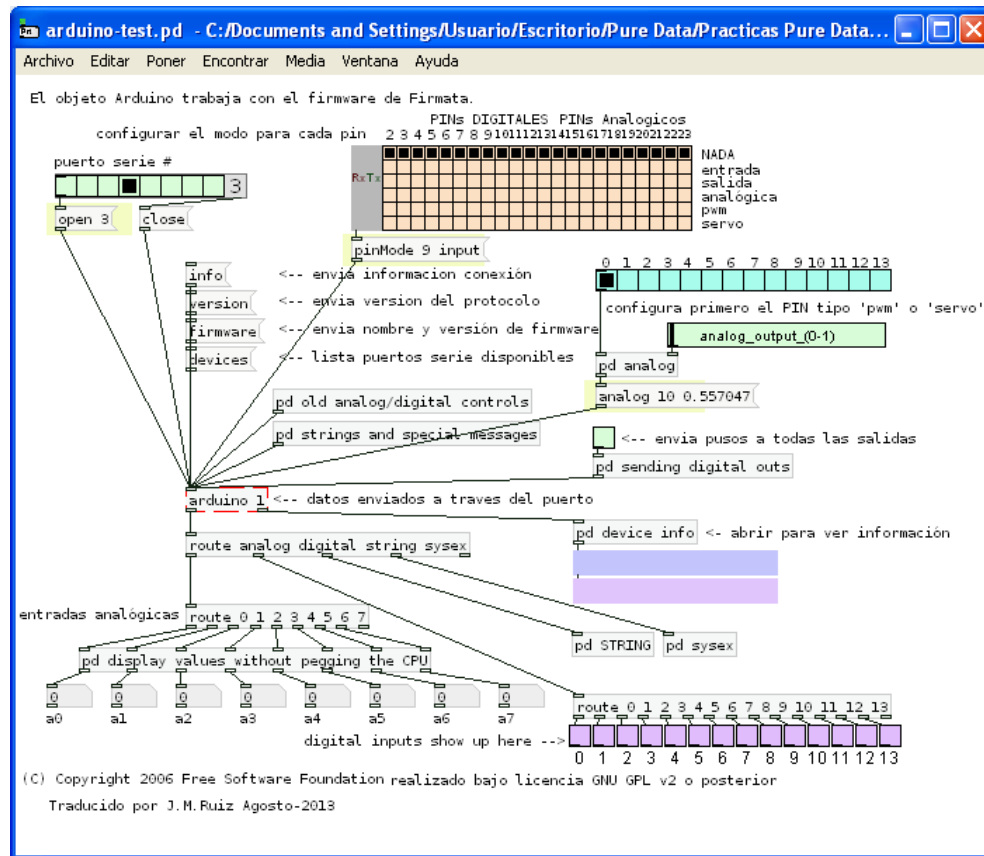
Test General.pd

Sugiero que abras este **patch** y pruebes todas y cada una de las opciones que acabo de explicar, de este modo podrás comprender mejor el funcionamiento de **Pduino**. Debo dejar claro que este patch es una traducción y adaptación del que viene en la propia librería **Pduino**



Fichero: *m10-Test General.pd*

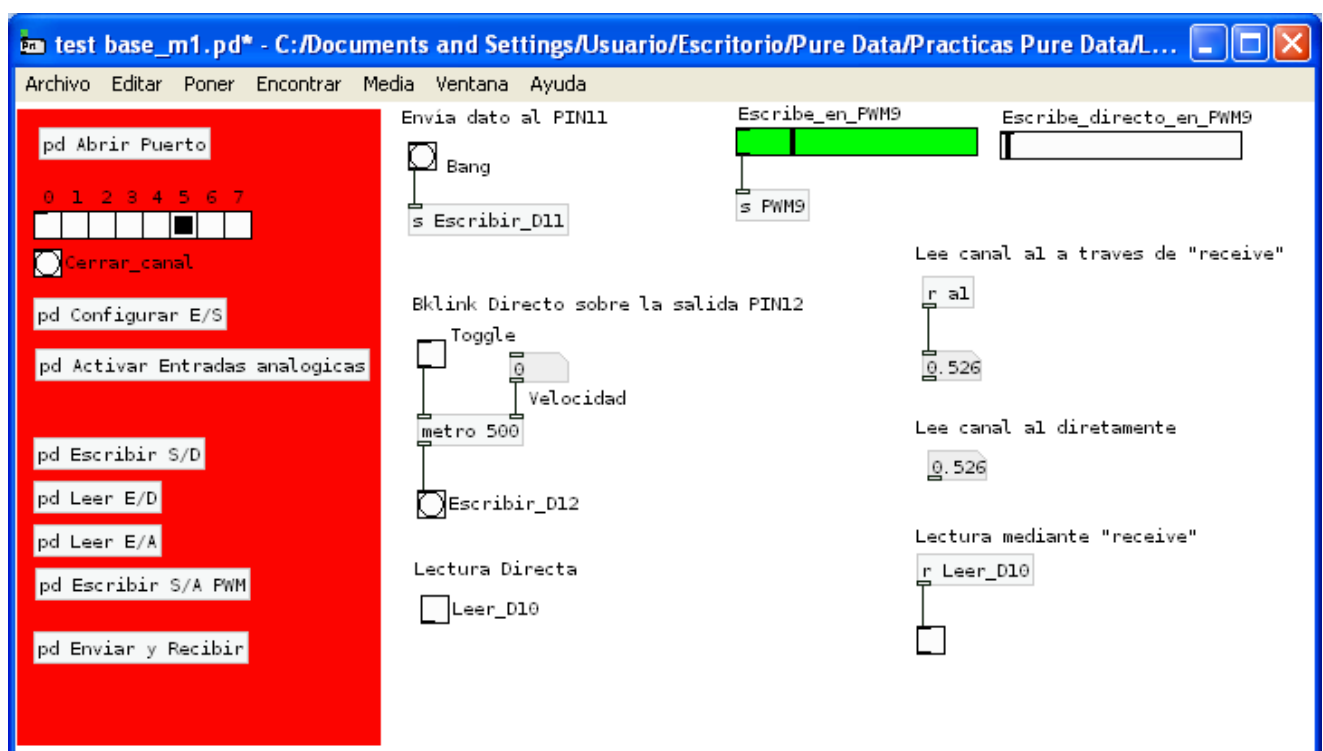
Hay un segundo fichero de pruebas y test que viene con la librería **Pduino** y que he traducido también. Se denomina *arduino-test.pd* y viene a ser lo mismo que el anterior su mecánica de trabajo es la misma. A continuación pongo una imagen del fichero abierto.



Ejemplo general de comandos con el modelo de trabajo base m1.pd

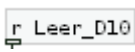
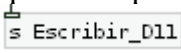
Con este ejemplo propongo que te inicies en el manejo de este modelo de trabajo *base_m1.pd*.

Recuerda, ya lo hemos explicado, que en la parte derecha están los bloques (**sub-patch**) de **PD** que te permiten realizar las operaciones que ellos mismos te indican con sus nombres, pero lo interesante en este caso es que no será necesario que los abras, salvo para realizar la configuración de los pines digitales mediante el **sub-patch Configurar E/S** y en el caso de trabajar con señales analógicas debes abrir el **sub-patch Activar entradas analógicas** con el fin de marcar aquellas con las que deseas trabajar, por lo demás bastará que en los bloque objeto-función de PD invoques las correspondientes señales analógicas y digitales de entrada y salida, etc...



Fichero: *m10-test base_m1.pd*

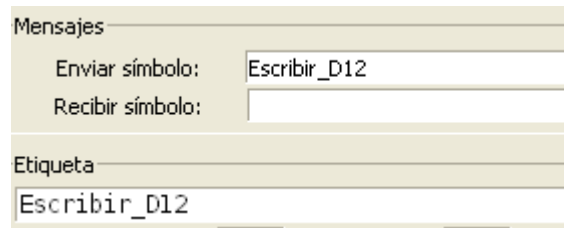
En este ejercicio se han colocado distintos ordenes sobre las variables de **Arduino**.

He utilizado bloques del tipo **receive**  para recoger señales de arduino y bloques de tipo **send**  para enviar señales, también he colocado directamente bloque en de activación (**Toggle** o **Bang** en los que directamente, mediante la opción propiedades he colocado para enviar o recibir, según el caso las propias variables. De cualquiera de las dos formas podemos realizar el gobierno y lectura de señales.

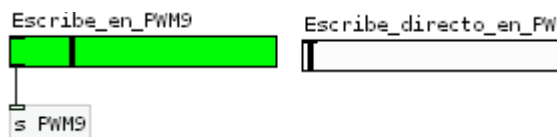


Activamos el PIN11 con un Bang y mediante la función **send** *s Escribir_D11*

Con un objeto **Toggle** (interruptor) activamos el bloque **metro** que genera impulso de duración 500 pero con posibilidad de variarla y recibe la señal un bloque **Bang** que la visualiza y además en sus **Propiedades** le hemos configurado como



(No olvidemos habilitar la lectura del canal mediante la opción del **sub-patch Configurar E/S**)

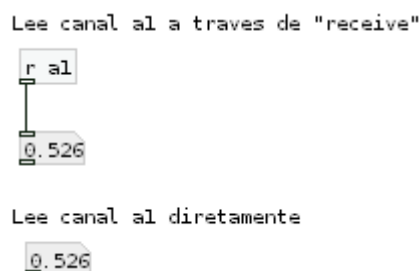


En este caso se envía una señal variable entre 0 y 1 mediante un **Hslider** y un bloque **send** y también con solo un **Hslider** en el que hemos puesto en propiedad la opción:

Enviar símbolo: *PWM9*

Etiqueta: *Escribe_directo_en_PWM9*

(No olvidemos habilitar la lectura del canal mediante la opción del **sub-patch Configurar E/S**)



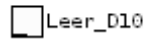
Aquí leemos el valor del canal analógico a0 de dos formas distintas: con la función **receive** *r a1* y directamente colocando en propiedades del bloque Numero

Recibir símbolo: *a1*

(No olvidemos habilitar la lectura del canal mediante la opción del **sub-patch Configurar E/S**)

Activar Entradas analógicas)

Lectura Directa

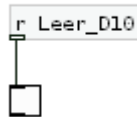


Leemos directamente en un bloque **Toggle** el estado de l PIN10 configurando en sus propiedades

Recibir símbolo: *Leer_D10*

(No olvidemos habilitar la lectura del canal mediante la opción del **sub-patch Configurar E/S**)

Lectura mediante "receive"

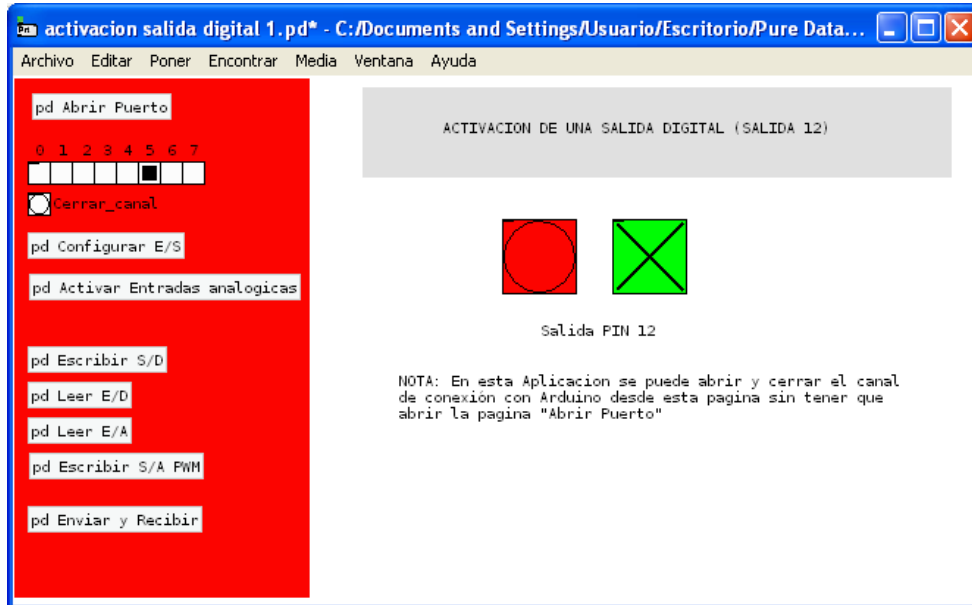


Leemos directamente el estado del PIN10 mediante un bloque **receive** *r Leer_D10* y lo llevamos el valor a un objeto **Toggle**

(No olvidemos habilitar la lectura del canal mediante la opción del **sub-patch Configurar E/S**)

3.1.7. Ejemplos de trabajo con el modelo “base_m1”

Activación de salida Digital



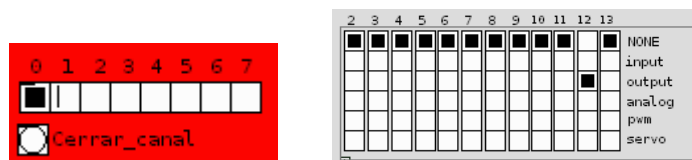
Fichero: *m11-activación salida digital 1.pd*

Este ejemplo es muy sencillo se trata de colocar dos objetos un **Bang** y un **Toggle** los cuales han de ser activados con el ratón envían el dato a la salida PIN12.



Configuraciones previas:

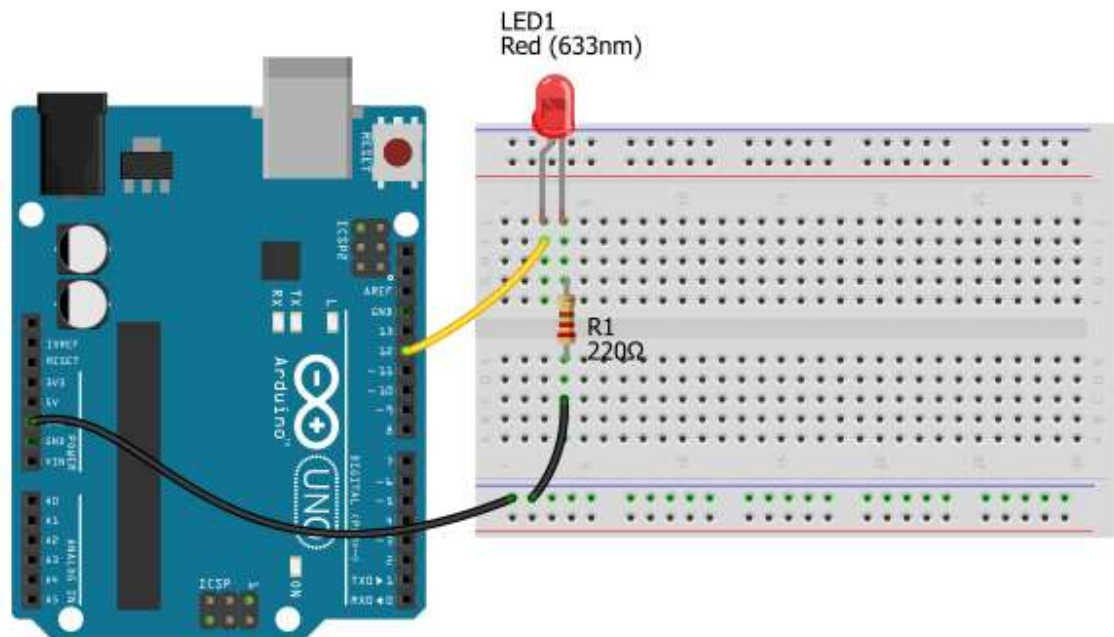
- Seleccionamos el puerto



- En esta aplicación no es necesario configurar las E/S digitales dado que por defecto, como sabemos **Arduino** considera sus pines digitales como salidas, pero se puede y quizá es recomendable configurar el **PIN12** como salida.
- Seleccionaremos el puerto al iniciar la ejecución.

Obsérvese la forma de trabajo y la diferencia entre los objetos **Bang** y **Toggle**, **Bang** actúa en modo *biestable* (un impulso activa y e siguiente desactiva) sin embargo **Toggle** actúa como un interruptor.

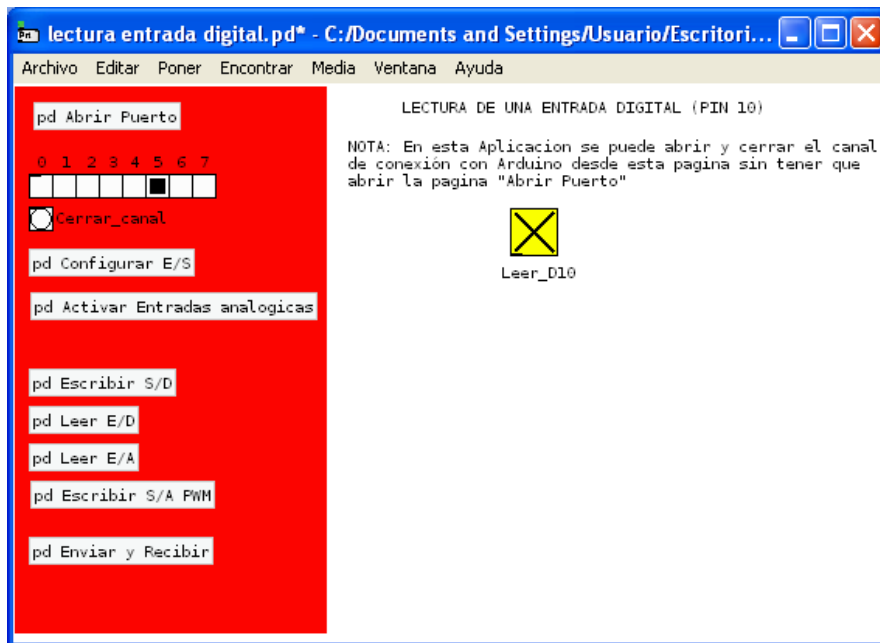
Esquema de montaje:



3.1.8. Lectura de una entrada digital

En este ejemplo sencillo se trata leer el estado de una entrada digital de **Arduino PIN10**.

Se recoge el valor en un objeto PD de tipo **Toggle**.



Fichero: *m12-lectura entrada digital.pd*

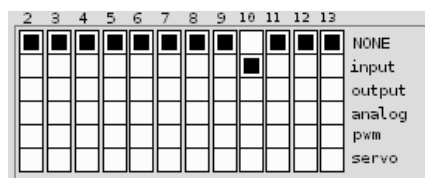


El **Toggle** se configurar de esta manera Leer_D10 **Propiedades**

Mensajes	
Enviar símbolo:	
Recibir símbolo:	Leer_D10

Configuraciones previas:

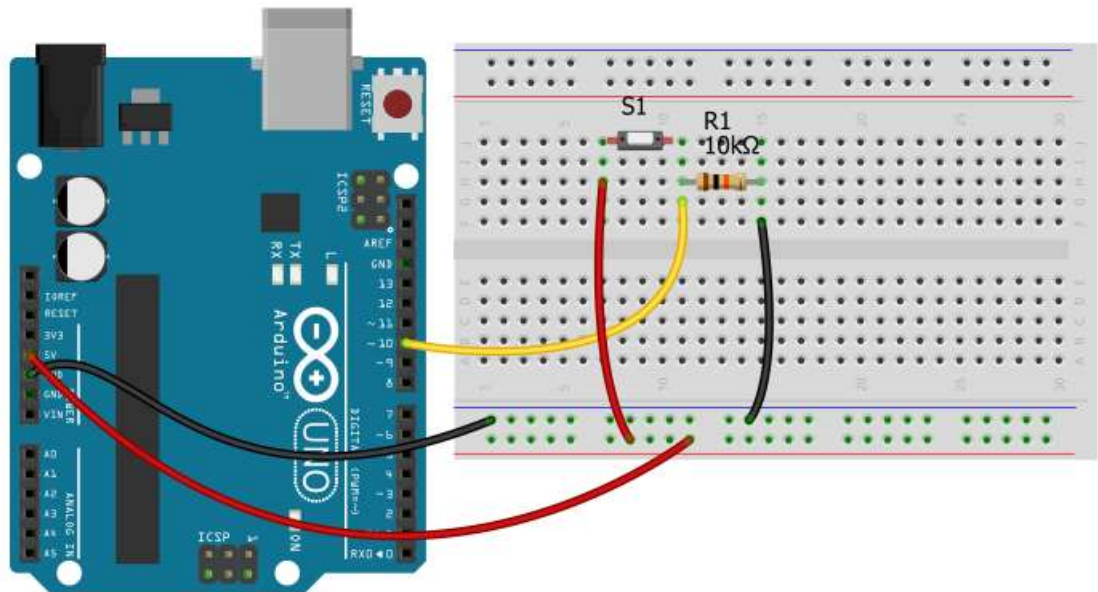
- Seleccionamos el puerto



- En esta aplicación configuraremos el **PIN10** como entrada.

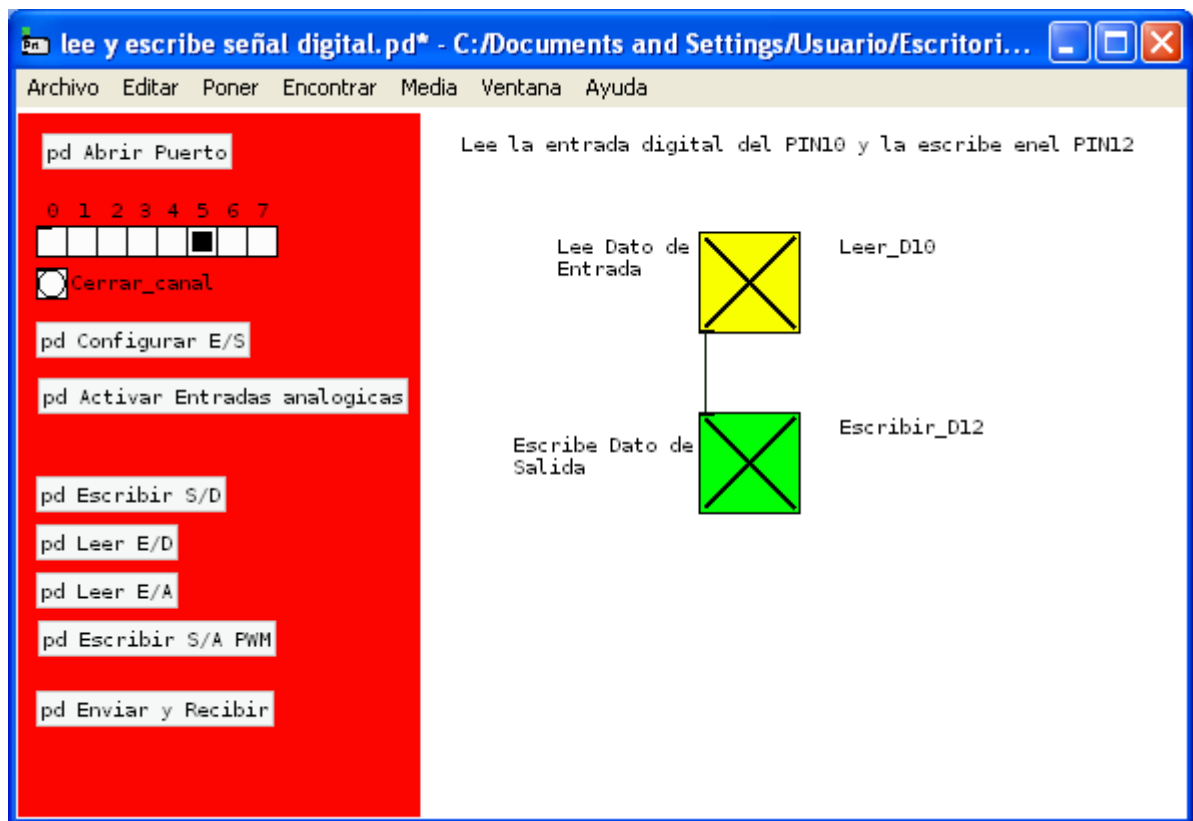
Obsérvese la forma de trabajo y la diferencia entre los objetos **Bang** y **Toggle**, **Bang** actúa en modo *biestable* (un impulso activa y e siguiente desactiva) sin embargo **Toggle** actúa como un interruptor.

Esquema de montaje:



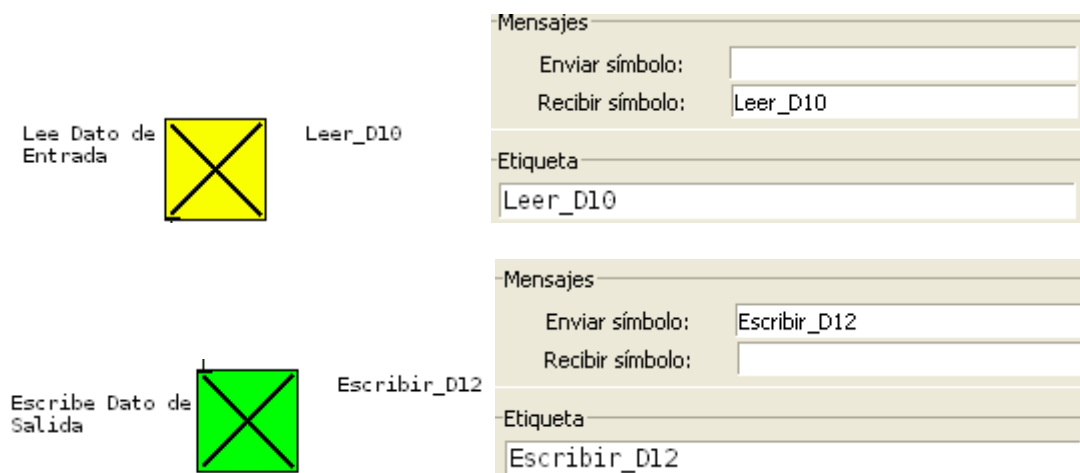
3.1.9. Lee y escribe señal digital

Leeremos una señal de la entrada **PIN10** de **Arduino** y este mismo valor lo sacaremos por el **PIN12** que actuará como salida.



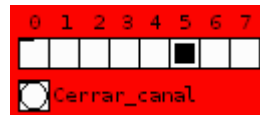
Fichero: *m13-lee y escribe señal digital.pd*

Este ejemplo es muy sencillo. Bastará configurar los objetos **Toggle** asociado a ellos las variables digitales mediante la opción Propiedades de cada uno de ellos (clic botón derecho estando sobre el objeto)

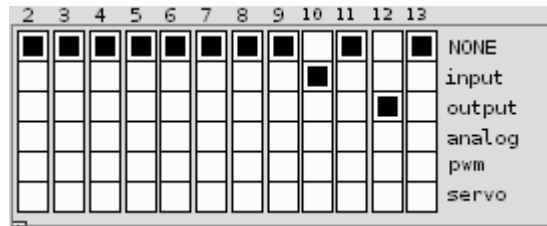


Configuraciones:

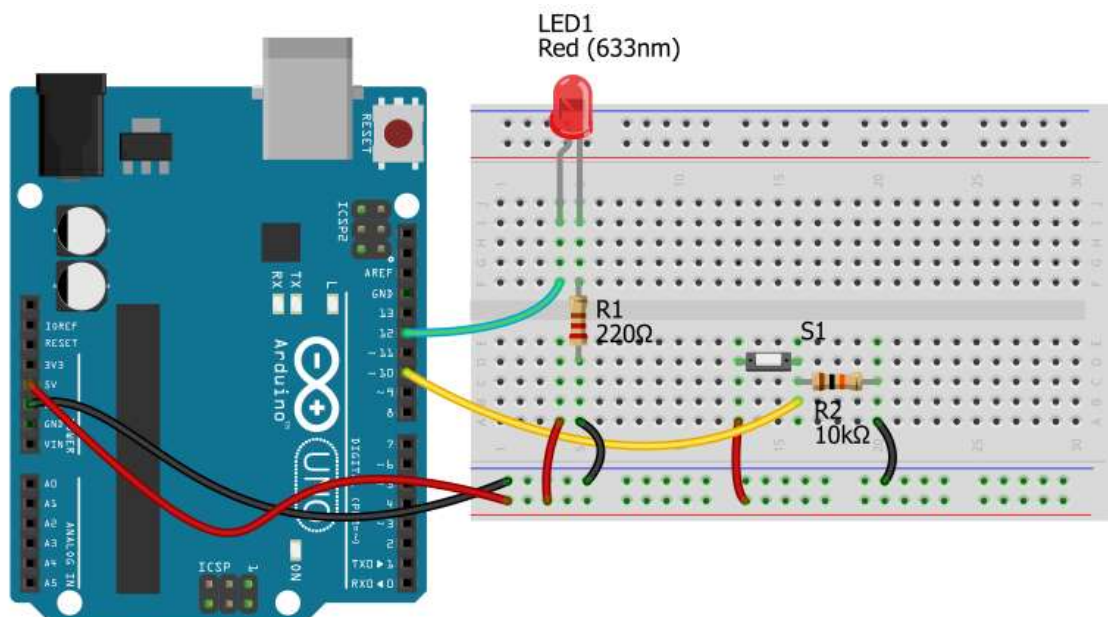
- Como siempre seleccionamos el puerto



- Se deben configurar los canales digitales mediante **Configurar E/S**

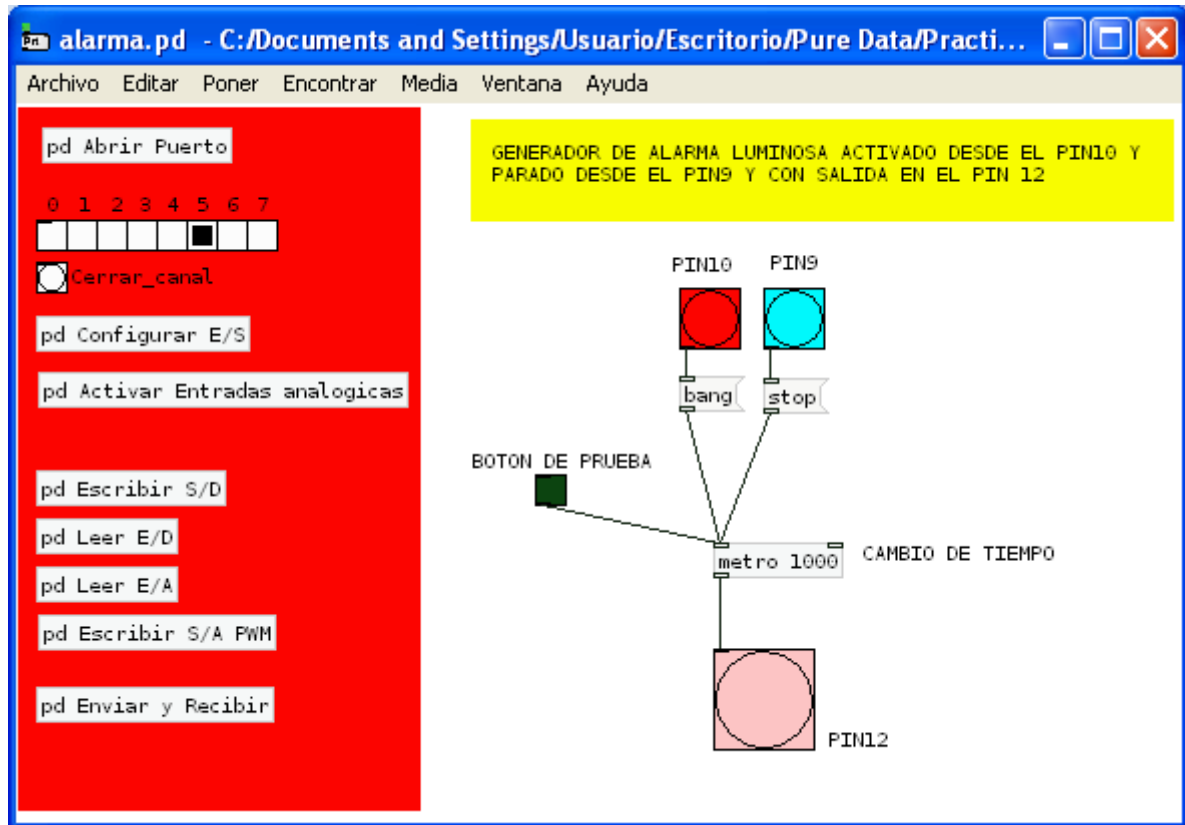


Esquema del montaje:



Alarma

Este ejemplo nos permite comprender el funcionamiento del objeto PD denominado **metro** que lo que hace es generar impulsos (metrónomo) a su salida en función del tiempo que nosotros le designemos “*metro 1000*” impulsos cada 1000 ms que hemos etiquetado como “CAMBIO DE TIEMPO”.



Fichero: *m14-alarma.pd*

Gobernamos la aplicación con dos elementos de tipo Bang que recogen sus valores (órdenes de actuación) de los pines **PIN10** Activar y **PIN9** Desactivar. Se ha colocado un botón de prueba para detener la señal del bloque *metro* “BOTON DE PRUEBA”

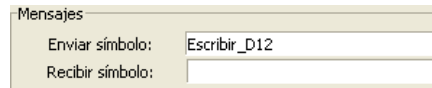
Los botones **Bang** se configuraran como elementos de envío de señal



Mensajes	
Enviar símbolo:	
Recibir símbolo:	Leer_D10



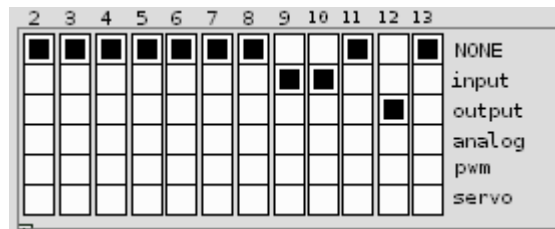
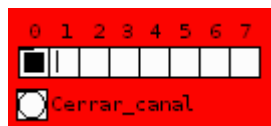
Mensajes	
Enviar símbolo:	
Recibir símbolo:	Leer_D9



La salida se realiza hacia el PIN12

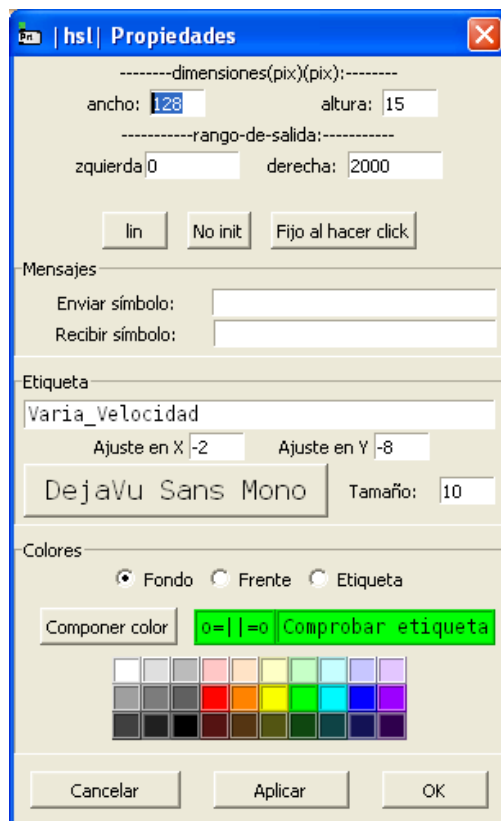
Configuraciones:

- Seleccionamos el puerto.



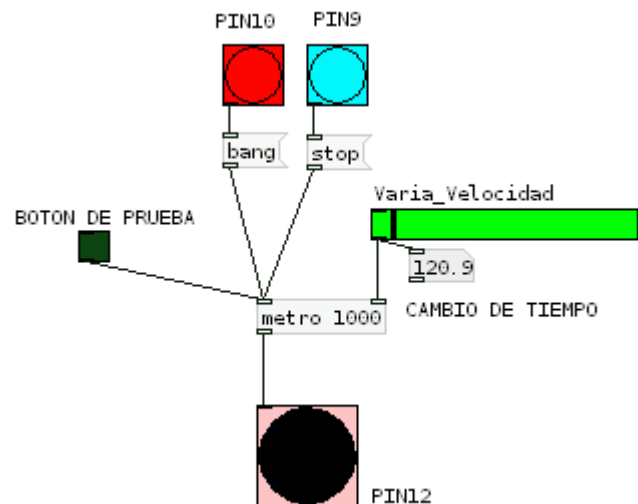
- Configuraremos las E/S digitales de esta forma

Sugiero que se modifique el tiempo del bloque metro bien editando el bloque o poniendo un objeto **Hslider** en su *inlet* derecho.



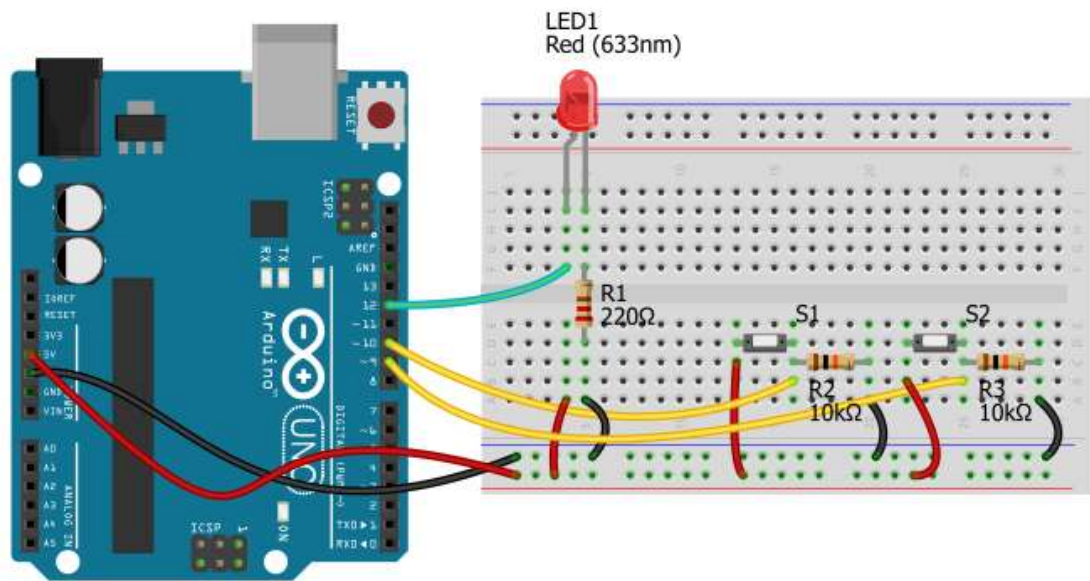
El **Hslider** se configuraría mediante la opción **Propiedades** de esta manera:

En la figura muestro como quedaría este:



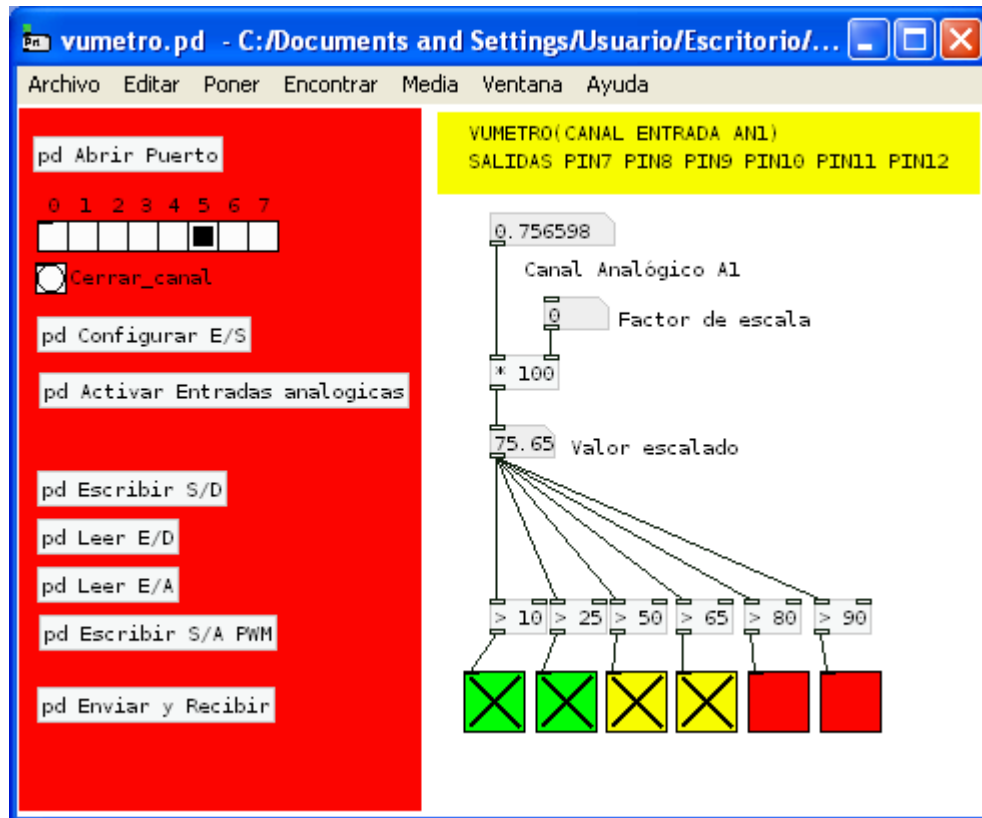
Fichero: *m14- alarma_1.pd*

Esquema de montaje:



3.1.10. Vúmetro

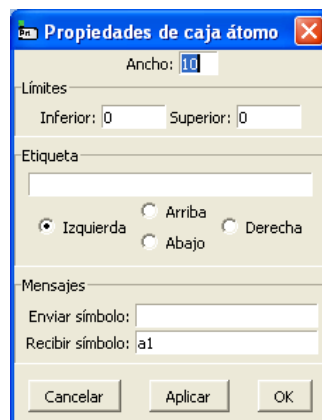
En el siguiente ejemplo implementaremos un sencillo vúmetro que medirá la señal recogida a través del canal analógico de **Arduino A1** y mediante una cadena de comparadores excitaremos hasta 6 salidas digitales pines **PIN7,PIN8,PIN9,PIN10,PIN11,PIN12**.



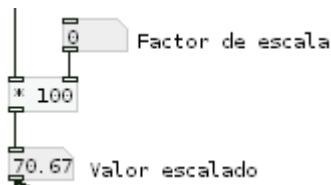
Fichero: *m15-vumetro.pd*

Se han dispuesto los siguientes bloques:

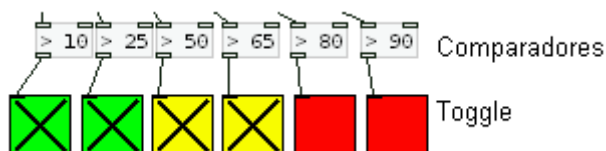
Bloque **Número** que recoge el canal a1



El bloque producto se ha colocado para multiplicar el valor de entrada (canal a1) que como sabemos **Pduino** entrega con un valor entre 0 y 1 a un valor más alto, en principio se multiplica por 100 pero con el bloque Numérico que tiene asociado se puede variar el valor simplemente pulsando y arrastrando el ratón sobre él.



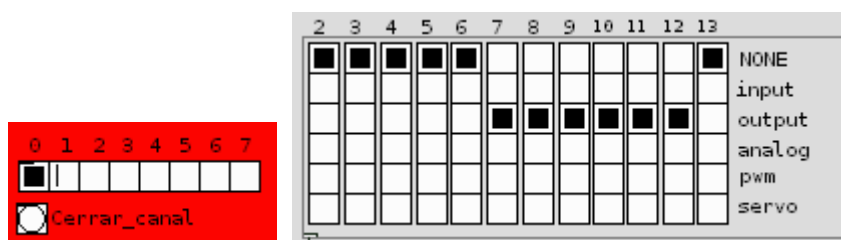
Finalmente se encuentran los bloques de comparación y los bloque **Toggle** de salida que visualizan las salidas a la vez que envían los valores a las salidas físicas de los pines de **Arduino** con las etiquetas:



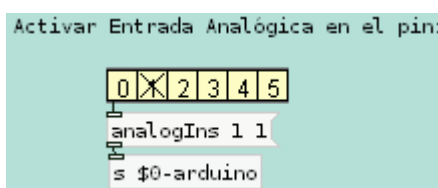
Propiedades-> Mensajes-> Enviar Símbolo: *Escribir_D7, ... Escribir_D12* (para cada uno de los **Toggle**) poniendo en cada uno de ellos el color adecuado (verde, amarillo y rojo)

Configuraciones:

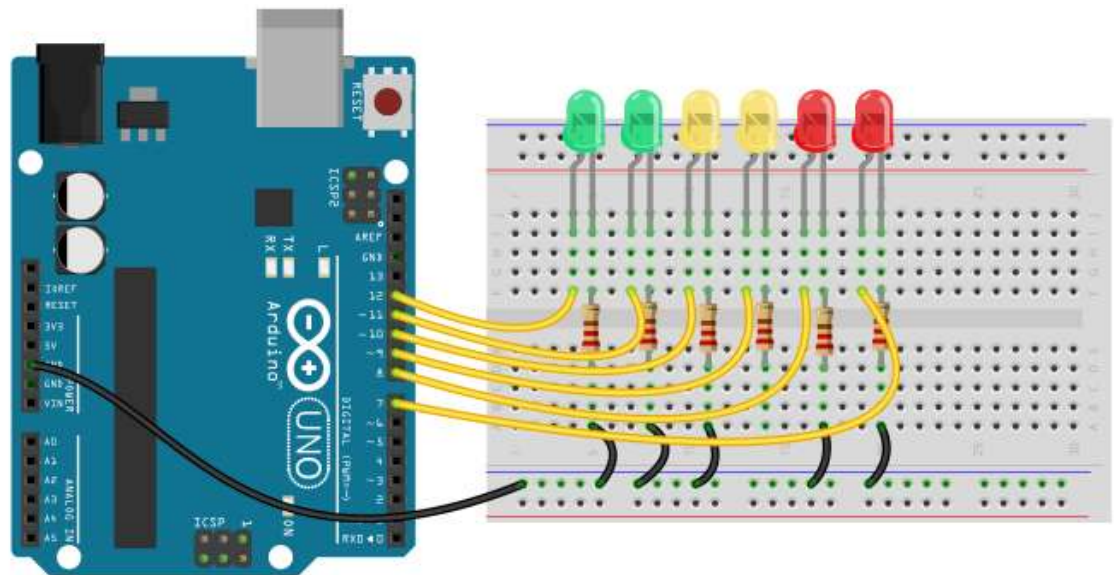
- Como siempre seleccionamos el puerto



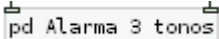
- Se deben configurar los canales digitales mediante **Configurar E/S**
- Con la opción **Activar Entradas** analógicas activamos el canal **a1**

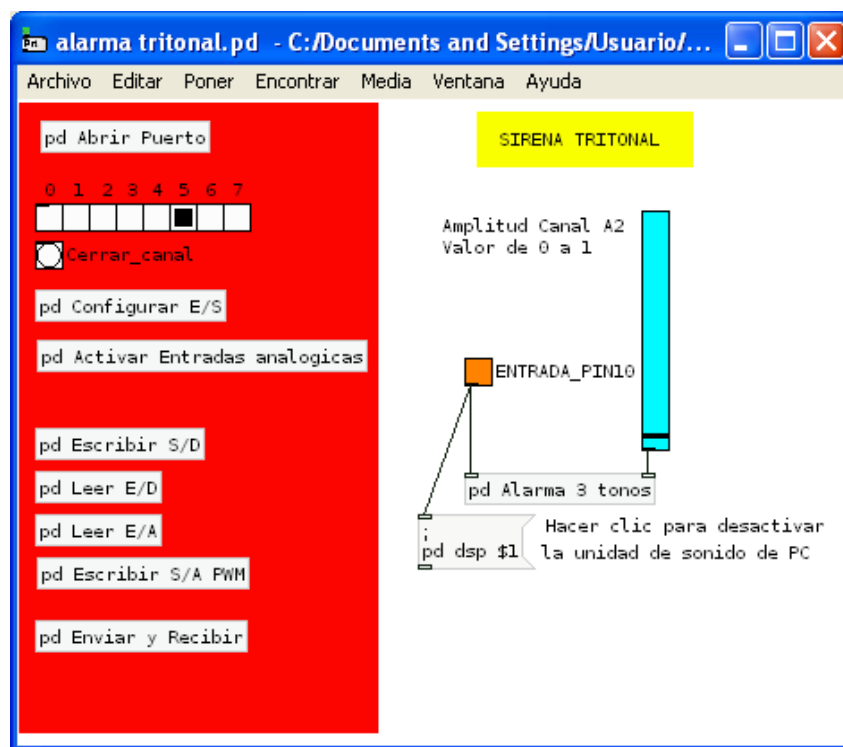


Esquema de montaje:



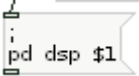
3.1.11. Alarma tritonal


Utilizando un **sub-patch** “pd Alarma 3 tonos”  de **PD**, encontrado en el foro, he realizado la implementación de una alarma que se activa a través del PIN10 de **Arduino** y en la que también se recoge el valor del canal analógico **a2** para utilizar la señal como control de amplitud de la salida de sonido

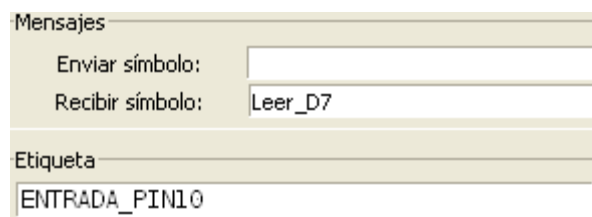


Fichero: *m16-alarma tritonal.pd*

En esta ocasión vamos a recurrir al uso de la herramienta DSP de PD que permite el control del sonido de nuestro PC para ello se genera el mensaje mediante un bloque

mensaje  con el contenido que se ve dentro del bloque, con este mensaje activamos el sonido del PC

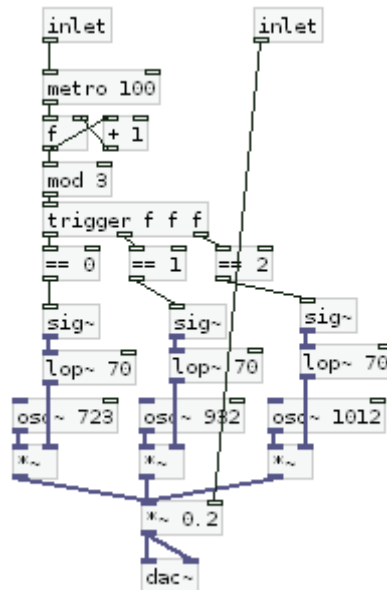
Se ha utilizado un bloque **Toggle** para recoger el valor del PIN10 que trabaja como entrada en la tarjeta **Arduino**  en el que se ha configurado mediante Propiedades (clic botón derecho estando sobre el objeto) como



El **Vslider** que recoge el valor “*Amplitud de Canal A2 Valor de 0 a 1*” quedaría configurado de la siguiente manera

Mensajes	
Enviar símbolo:	
Recibir símbolo:	a2

Ponemos a continuación el Sub-patch que genera el sonido *pd Alarma 3 tonos*:

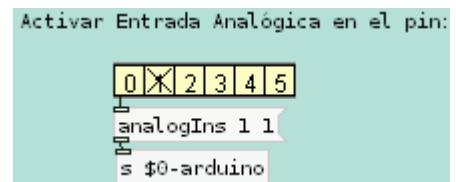
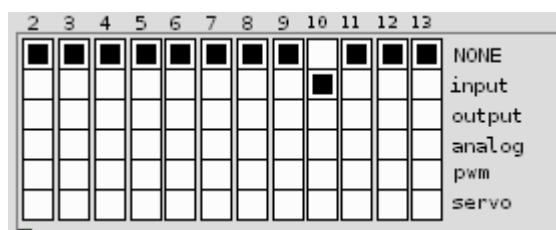


Configuraciones:

- Como siempre seleccionamos el puerto

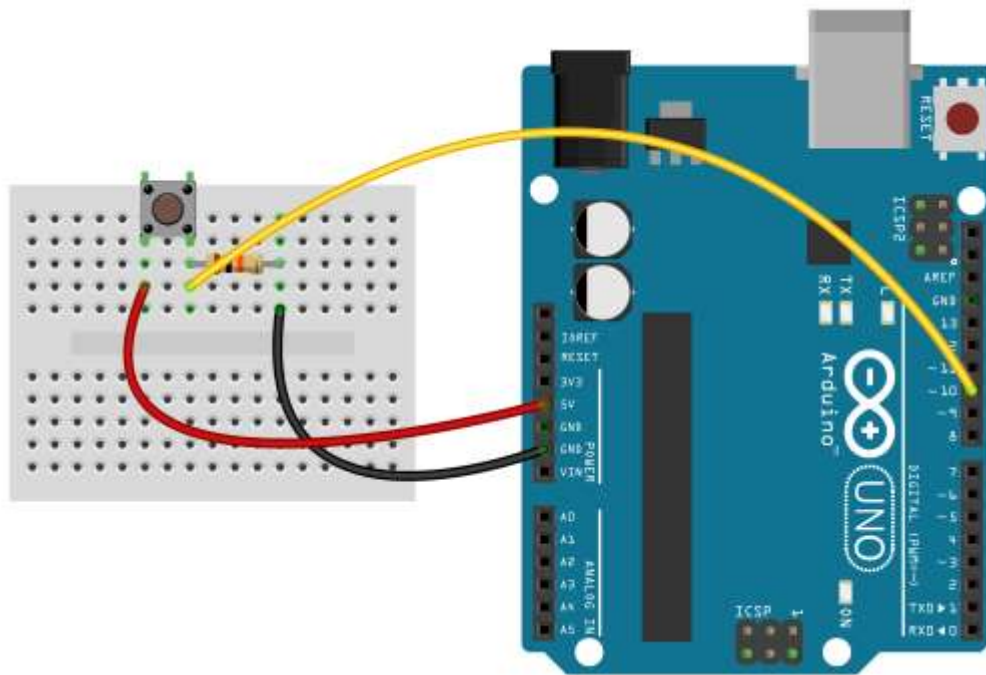


- Se deben configurar los canales digitales mediante **Configurar E/S**



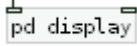
- Con la opción **Activar Entradas** analógicas activamos el canal **a1**

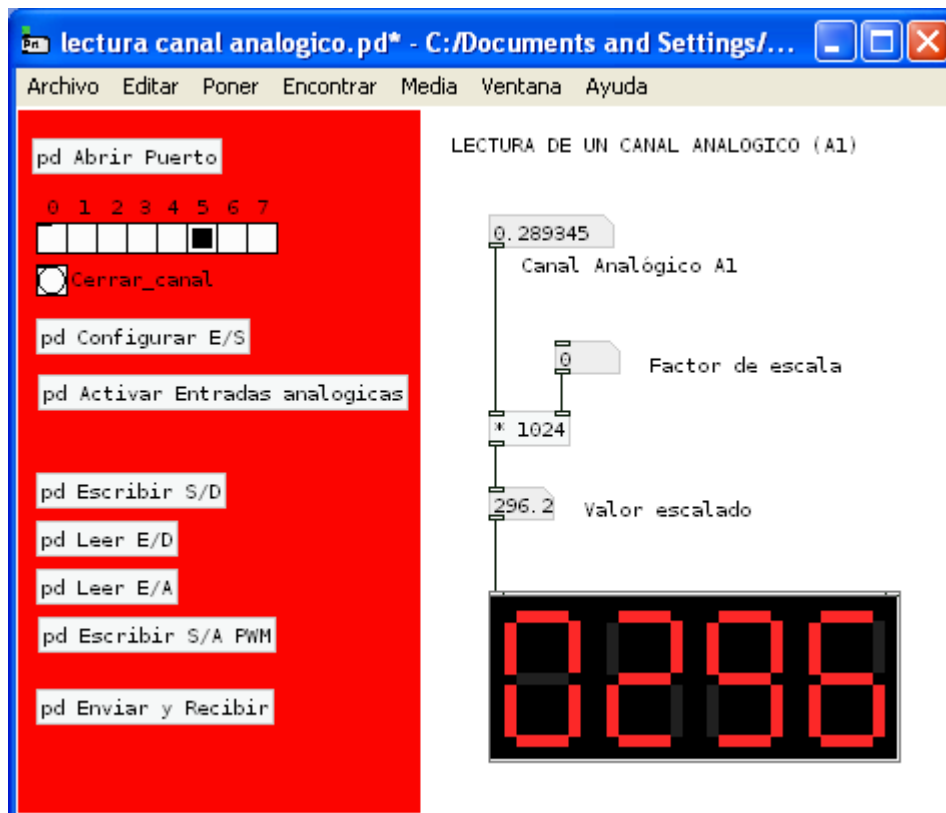
Esquema de montaje:



3.1.12. Lectura canal analógico.

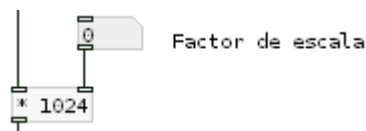
Leeremos un canal analógico, concretamente el canal A1 y lo mostraremos en un **display de cuatro dígitos** que hemos creado como **sub-patch** y hemos colocado en

nuestro ejemplo simplemente poniendo en un Objeto su nombre . Es fundamental que para invocar a este **sub-patch** exista en la carpeta en donde tengamos grabado nuestro ejemplo.



Fichero: *m17- lectura canal analogico.pd*

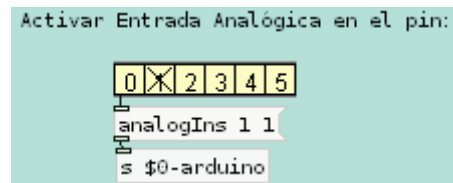
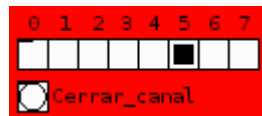
La señal que leemos tiene un margen de valor entre **0 y 1** y por lo que hacemos es escalarla de 0 a 1024 que como sabemos es el valor que entrega realmente **Arduino** pero que la librería **Pduino** reduce de escala de **0 a 1**. Para este escalado utilizamos un bloque **producto** al que le hemos colocado en su parte *inlet derecho* un bloque numerado que permite variar la escala si es que lo deseamos.



Seguidamente he colocado un bloque numérico para mostrar el valor y a la vez entregarlo al **sub-patch display4D.pd**

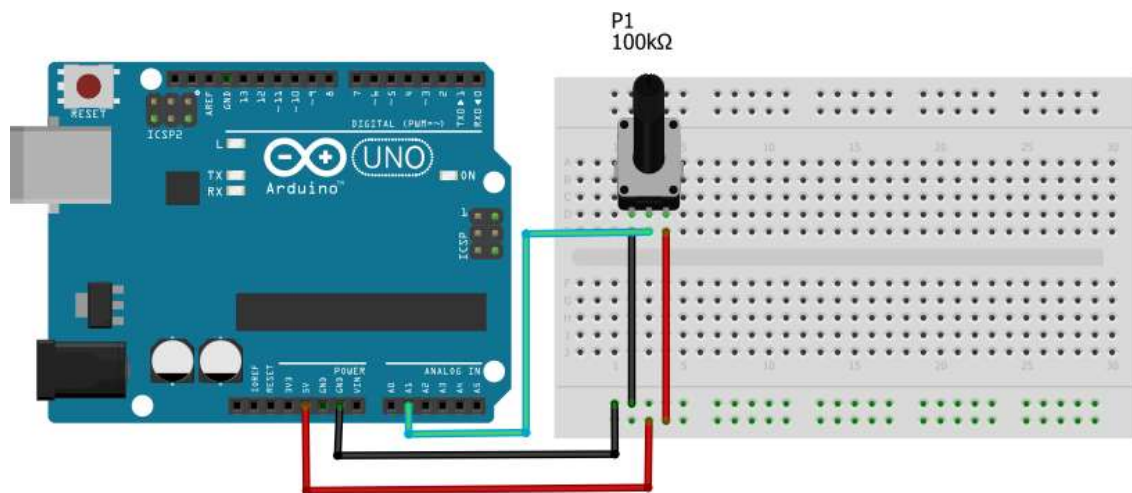
Configuraciones:

- Como siempre seleccionamos el puerto



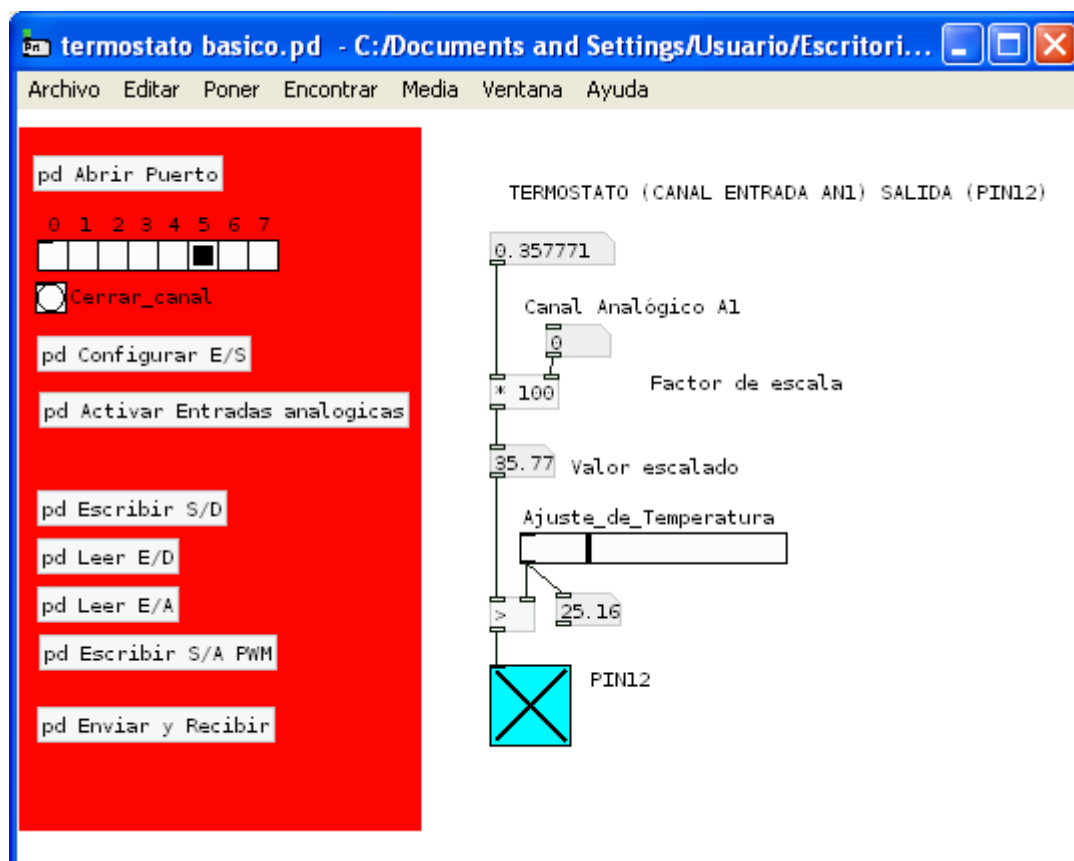
- Con la opción **Activar Entradas** analógicas activamos el canal **a1**

Esquema de montaje:

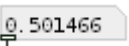


Termostato básico

Con el siguiente ejemplo voy a implementar un sencillo termostato. Se trata de recoger la señal del sensor de temperatura del canal analógico de entrada de **Arduino A1** etiquetado en nuestro entorno como **a1** y realizar la comparación con un valor de consigna que generará un **Hslider** de **PD** realizándose después la comparación de ambos valores en un bloque de comparación.



Fichero: *m18-temostato basico.pd*

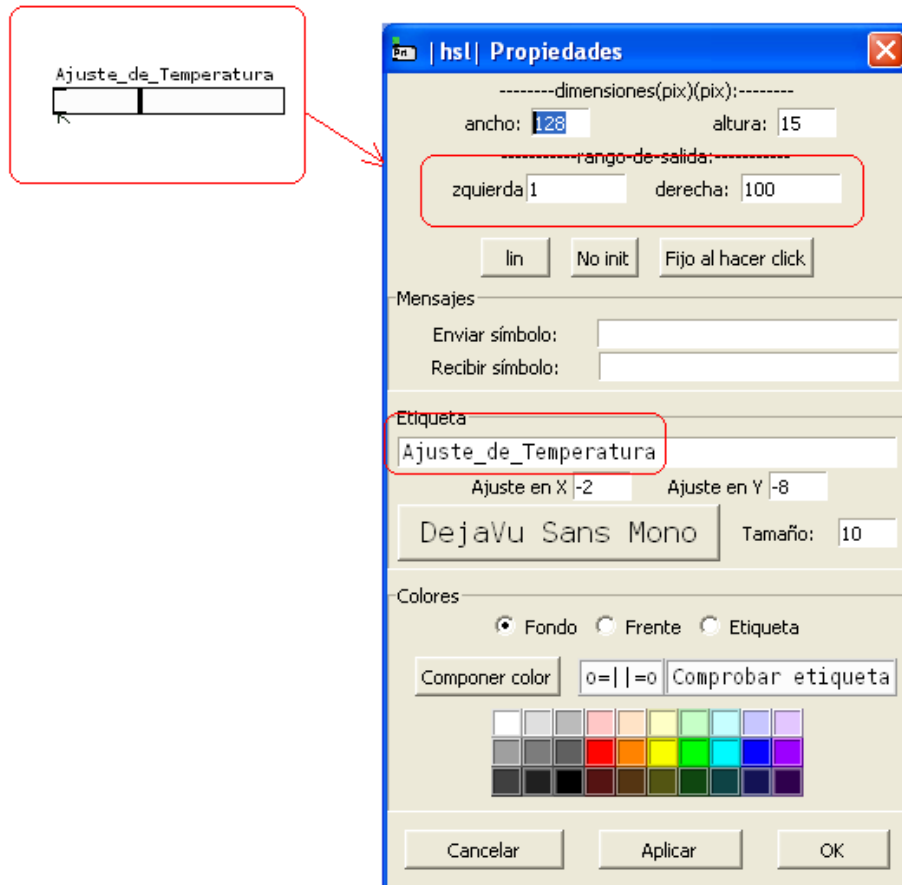
La señal se recoge en un bloque de tipo **Numero**  en el que hemos configurado en sus **Propiedades** la recepción del valor del canal *a1*, ya sabemos como realizar esta operación porque lo hemos hecho en los ejemplos anteriores varias veces.

La señal recogida en el bloque **Numero** pasa a un bloque **Producto** que la escala de 0 a 1



Sea multiplicada por 100, es decir llevándola a una escala entre 0 y 124, a este bloque le he colocado un bloque **Numero** (*inlet* derecho del bloque) con el fin de, si lo deseamos, poder modificar la escala (factor de producto).

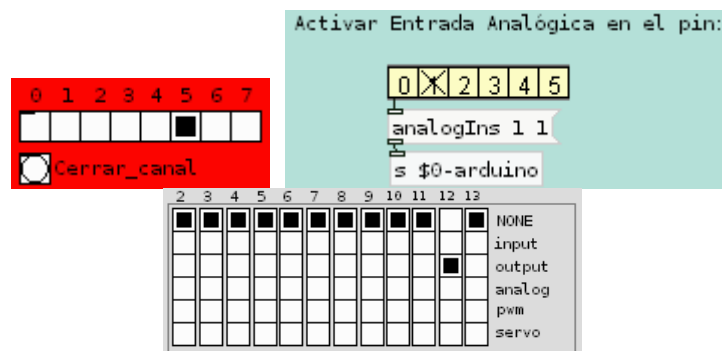
Finalmente la señal escalada se lleva aun **Comparador** en el que la comparamos con la consigna generada por el bloque **Hslider** que hemos programado sus **Propiedades** para un valor entre **0 y 124**



Programación del bloque **Hslider**

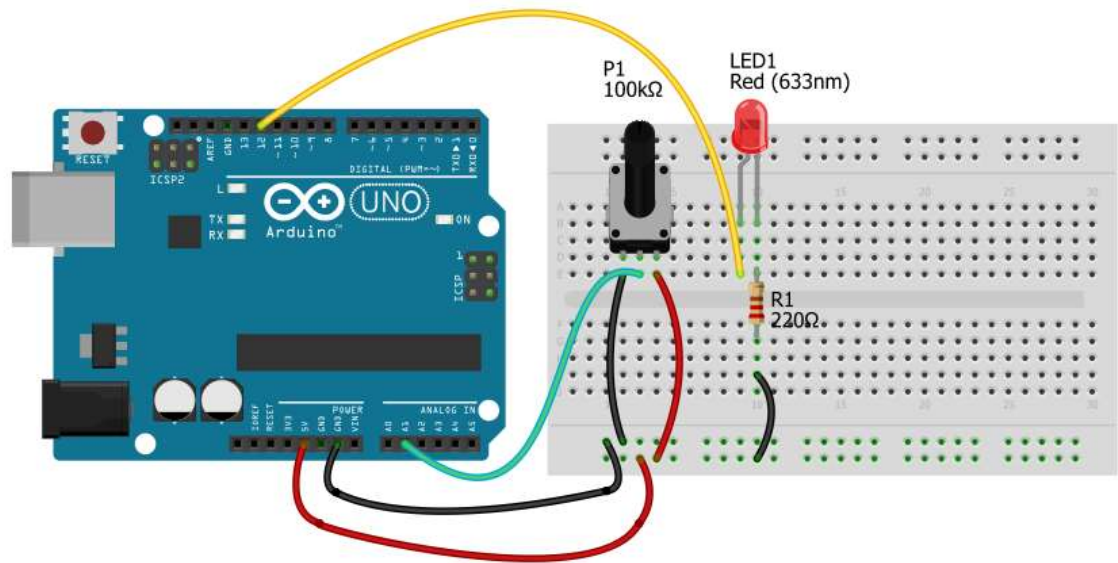
Configuraciones:

- Como siempre seleccionamos el puerto



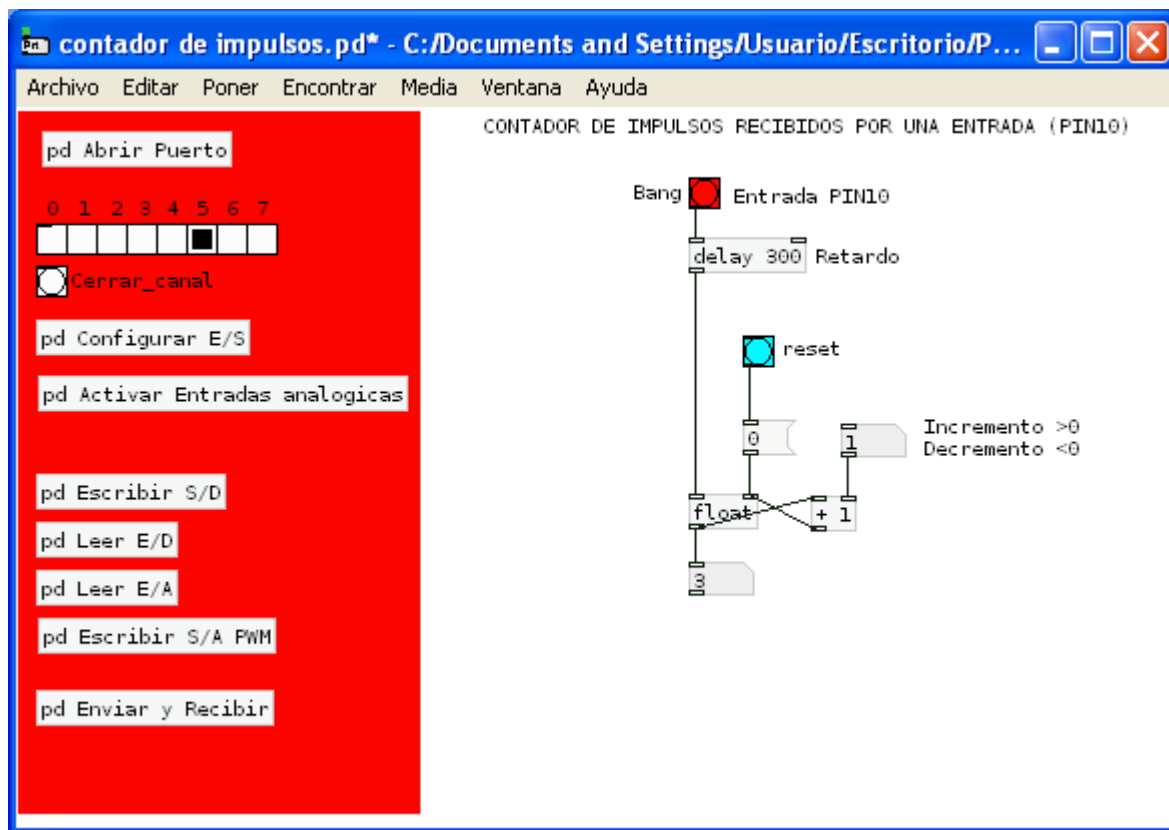
- Con la opción **Activar Entradas** analógicas activamos el canal **a1**
- Seleccionamos el **PIN12** como **salida** con la opción **Configurar E/S**

Esquema de montaje:



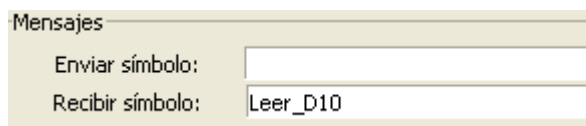
3.1.13. Contador de impulsos

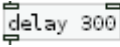
Esta vez vamos a contar impulsos que nos llegan desde una entrada de **Arduino**. En este caso será desde el **PIN10**





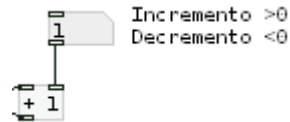
Fichero: *m19-contador de impulsos.pd*

Recogemos el valor del **PIN10** (entrada digital) mediante un objeto **Bang** en el que le hemos configurado la **Propiedad** de recibir dato:



Se ha colocado el objeto **delay**  Retardo para evitar “rebotes” en la señal de entrada.

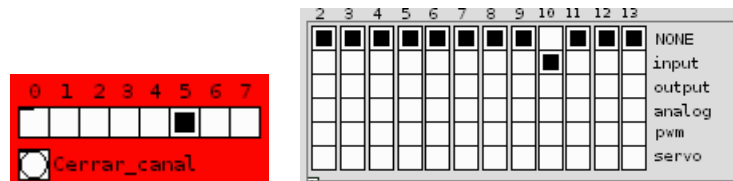
Se ha creado un contador con la conjunción de dos bloques uno de tipo **float**  y uno de **suma** . En el bloque **float** (*inlet* derecho) se ha colocado una puesta a cero y el bloque **suma** el valor de incremento en la suma, téngase en cuenta que se podría sumar (avanzar) o restar (retroceder) en las cuentas de impulsos.



IMPORTANTE: El reset se lleva a cabo cuando llega el siguiente impulso en la entrada

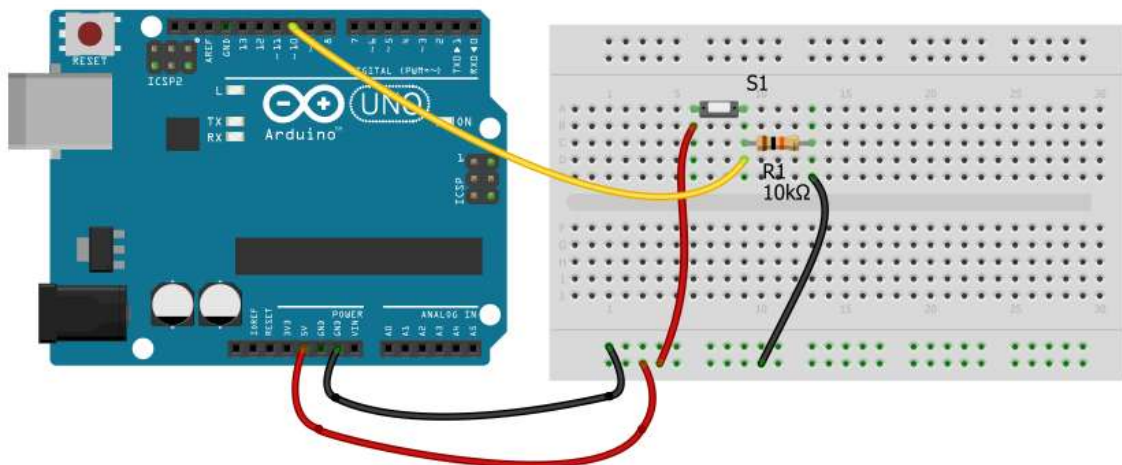
Configuraciones:

- Como siempre seleccionamos el puerto



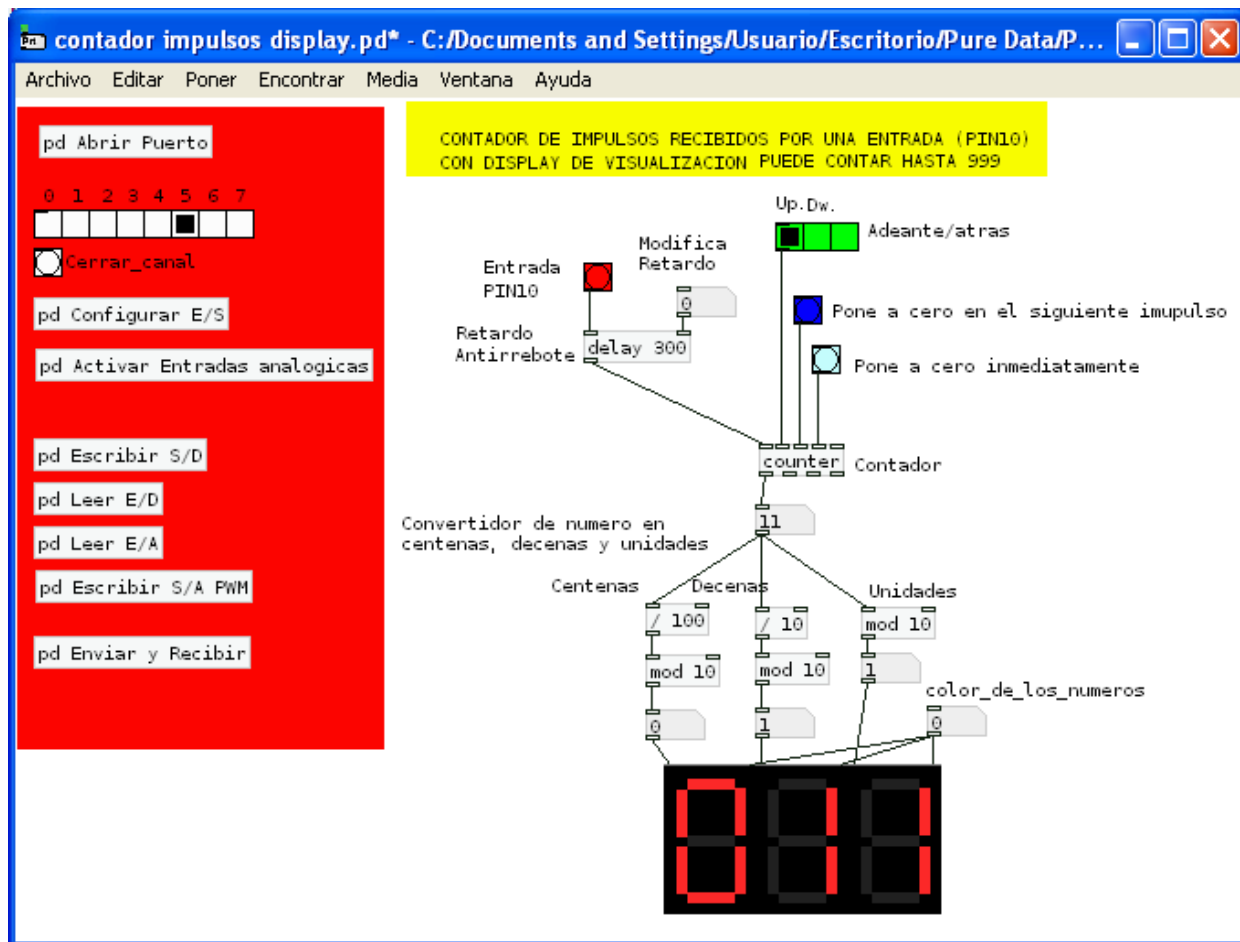
- Seleccionamos el **PIN10** como **entrada** con la opción **Configurar E/S**

Esquema de montaje:



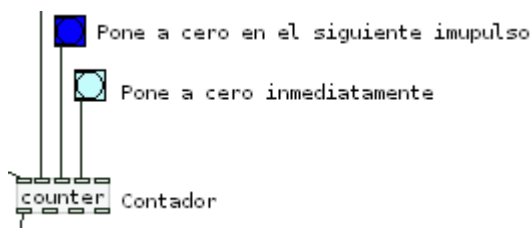
3.1.14. Contador de impulsos con display

PD tiene un bloque de función llamado **counter** que ofrece interesantes prestaciones para nuestro trabajo, por eso lo hemos elegido para esta nueva variación del contador de impulso en la que además hemos colocado un display de hasta 3 dígitos que permita mostrar la cuenta de manera vistosa.



Fichero: *m110-contador impulsos display.pd*

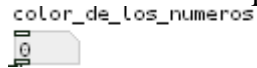
Los impulso, de la misma manera que en el anterior caso los recogemos por la entrada PIN10y los llevamos al contador. En el contador hemos utilizado tres de sus **inlets** con el fin de poder fijar el *modo de cuenta* (**Up,Dw** y **Ap-Dw**), *puesta a cero en el siguiente impulso* y *puesta a cero inmediata*.



Destaco también el conjunto de bloques que permiten el desdoble del numero de salida del contador en Centenas, Decenas y Unidades que como se puede ver es tan sencillo como hacer divisiones por 100, 10, 1 respectivamente utilizando el bloque **mod**.

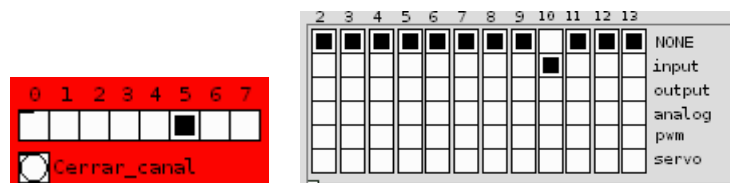


Obsérvese que se puede cambiar el color de los números del **display**



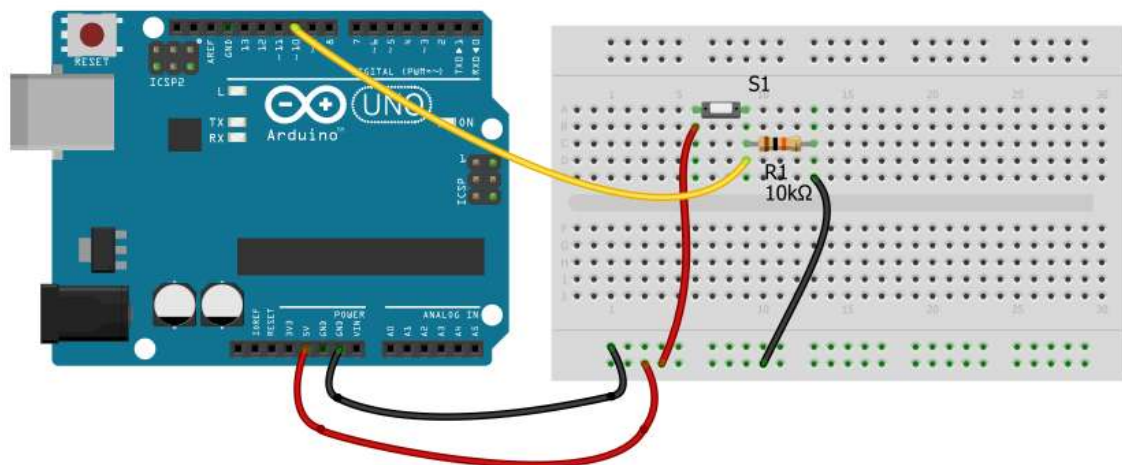
Configuraciones:

- Como siempre seleccionamos el puerto



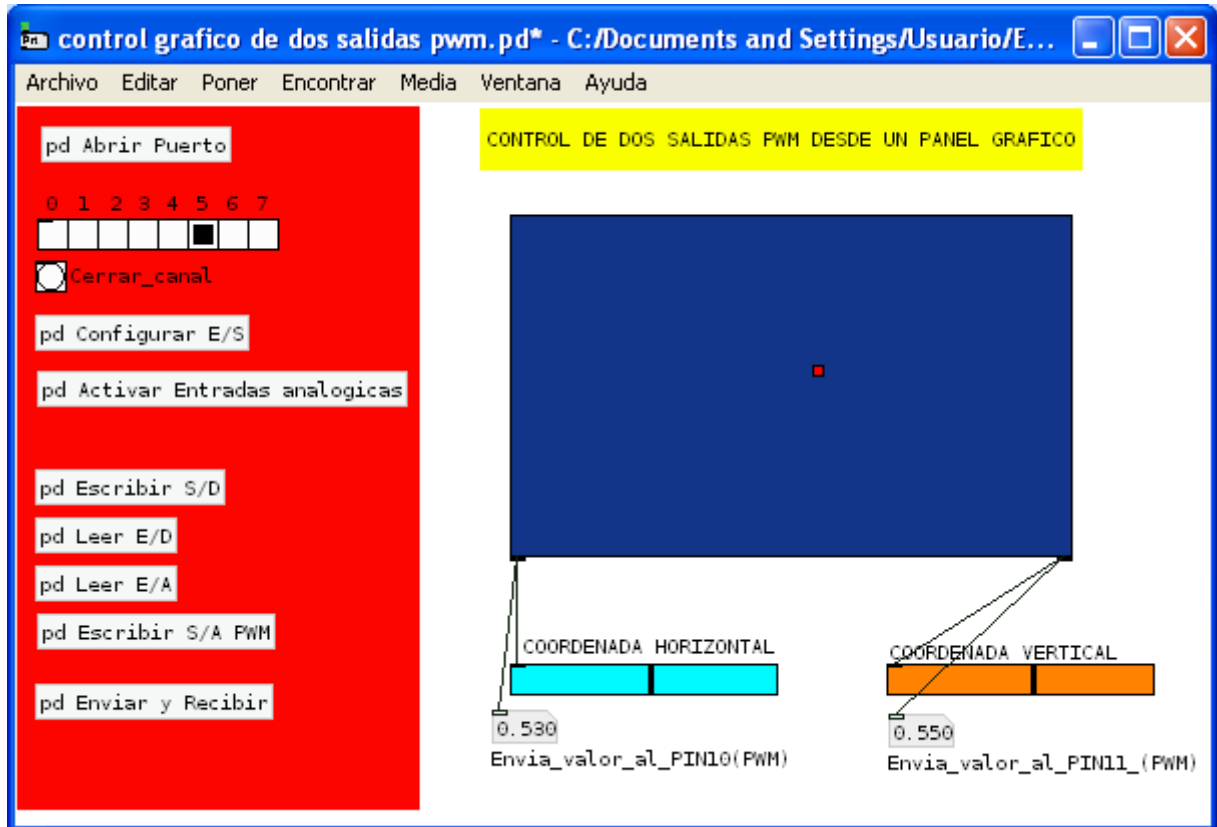
- Seleccionamos el **PIN10** como **entrada** con la opción **Configurar E/S**

Esquema de montaje:



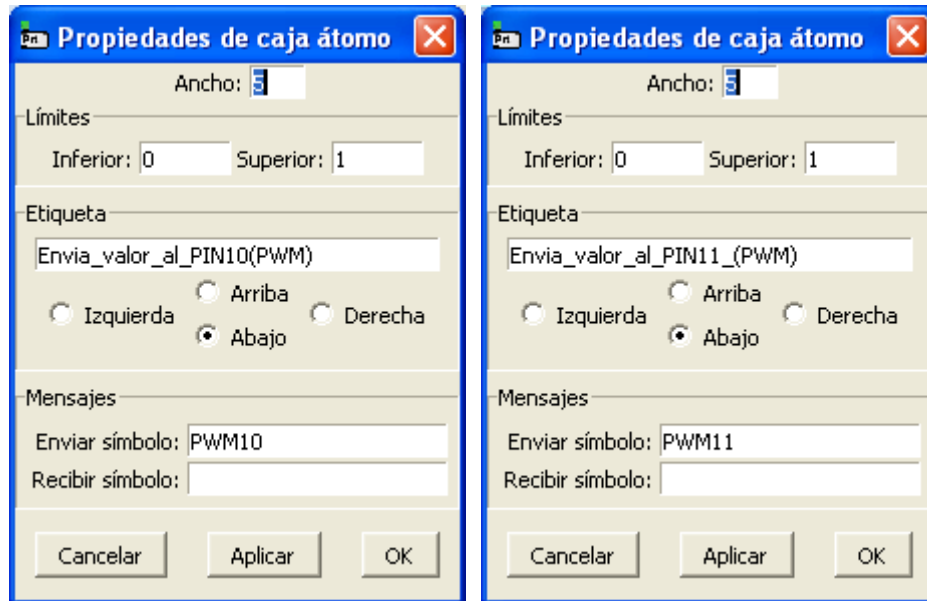
Control gráfico de dos salidas

PD tiene un bloque muy interesante que vamos a experimentar con el en este ejemplo. Se trata del bloque **grid** que despliega una caja en el área de trabajo dentro de la cual, al mover el ratón, se entregan en sus salidas *outlet* los valores de las coordenadas **x** e **y** de la posición del ratón en la caja.



Fichero: *m11-control grafico de dos salidas pwm.pd*

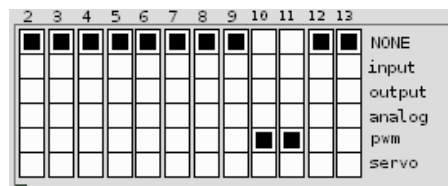
Lo que nosotros haremos será recoger estos valores y llevarlos a las salidas digitales **PIN10** y **PIN11** que configuraremos como salidas analógicas de tipo PWM. Hemos dispuesto para cada salida un slider de tipo **Hslider** que nos muestra el valor. Los valores se entregan **Arduino** mediante los bloques numéricos en los que hemos configurado su propiedad



Ventanas de parámetros de los bloques **Numero**

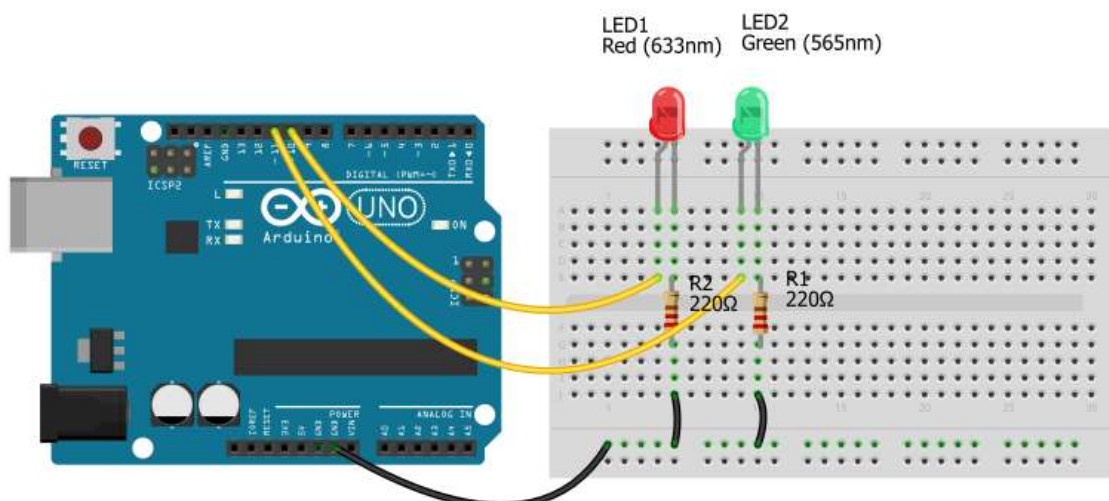
Configuraciones:

- Como siempre seleccionamos el puerto



- Seleccionamos el **PIN10** y **PIN11** como salidas de tipo PWM con la opción **Configurar E/S**

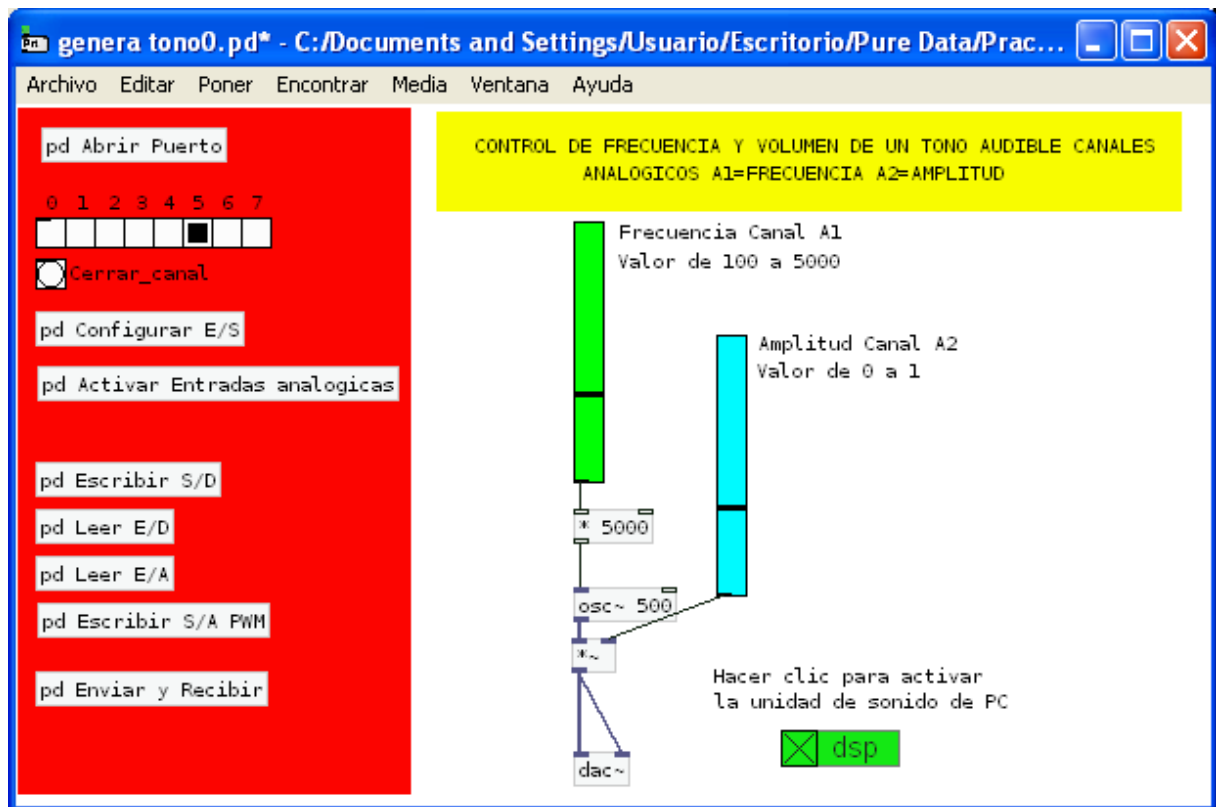
Esquema de montaje:




Generador de tono básico


PD es un lenguaje que incorpora poderosas herramientas orientadas al tratamiento de señales de audio. Quiero realizar un sencillo ejemplo en el que poder emitir un tono de frecuencia y amplitud variable haciendo uso de dos canales de entrada analógicos de **Arduino** los cales A1 y A2 (señales a1 y a2) con el primero controlar la frecuencia


generada por un bloque de tipo 



Fichero: *m12-genera_tono0.pd*

La señal que controla la frecuencia la debemos cambiar de escala ya que se nos entrega con un valor comprendido entre 0 y 1 y por eso la multiplicamos por 5000 para llevara a una escala superior .

El objeto producto  se encarga de multiplica la señal por el valor de la señal a2 (control de volumen) que varía entre 0 y 1 (esta vez se deja la señal tal como se nos entrega).

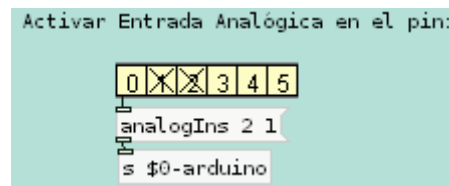
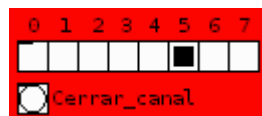
Finalmente la seña que es enviada al dispositivo de sonido mediante el bloque . El bloque de sonido en PD se denomina dsp y para activarlo se debe hacer o bien desde la ventana de visualización o en la propia ventana de trabajo poniendo el objeto de

Hacer clic para activar la unidad de sonido de PC

nombre **pddp/dsp** para luego aparece en forma y permitir fácilmente la conexión y desconexión de la unidad de audio. Ya hemos puesto otro método en este manual pero ahora pongo este para tener uno más.

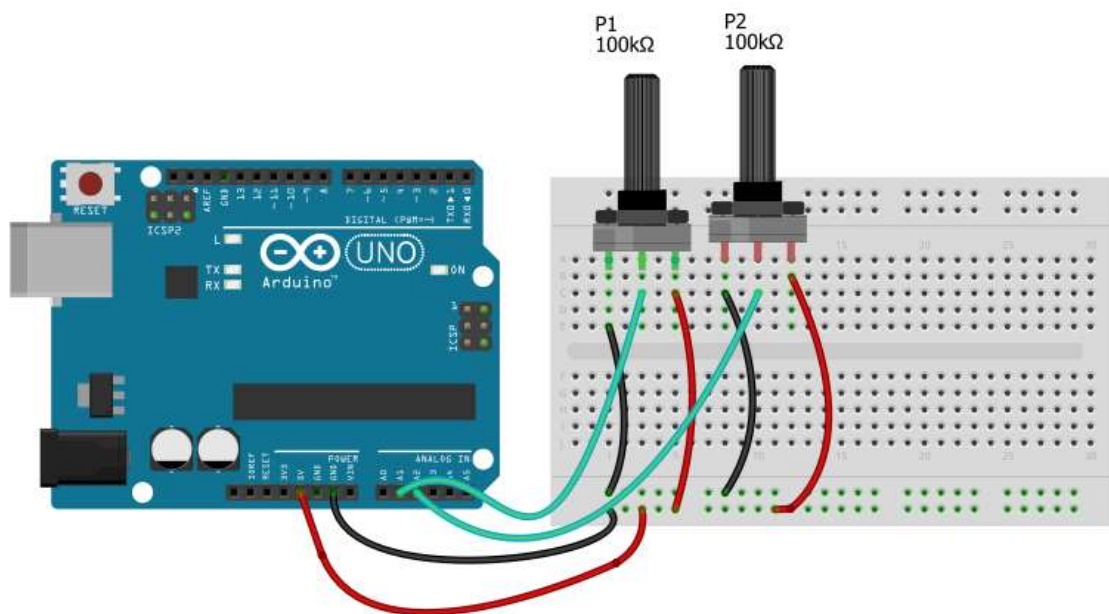
Configuraciones:

- Como siempre seleccionamos el puerto



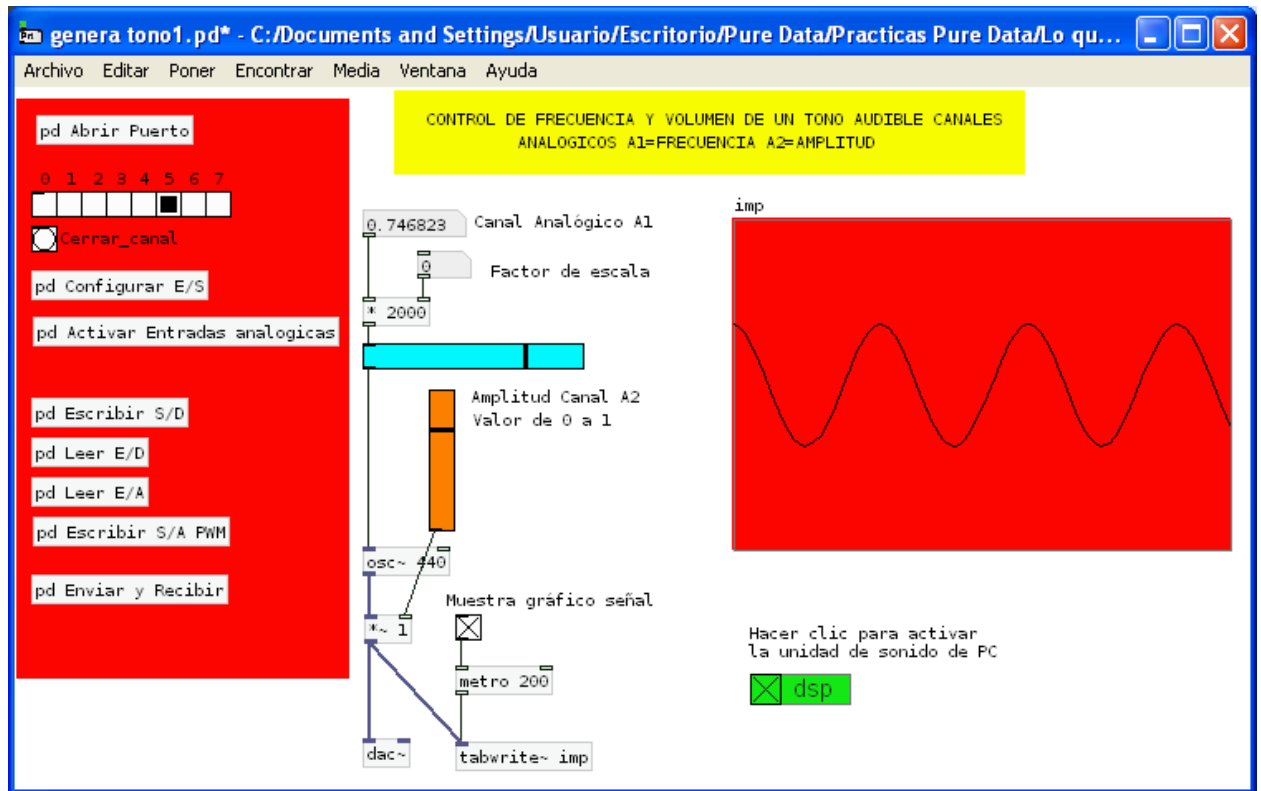
- Seleccionamos para leer los canales **a1** y **a2** **Activar Entradas analógicas**

Esquema de montaje:



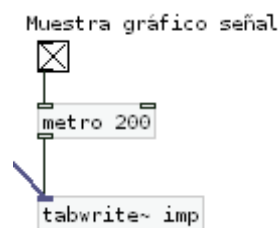
Generador de tono básico con visualización de señal.

En este ejemplo, ampliación del anterior, he incluido un bloque de representación grafica con el fin de visualizar las características de la señal que se produce.



Fichero: *m13-genera tono1.pd*

La parte de visualización se resuelve con un bloque **tabwrite**



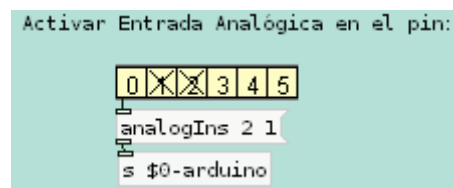
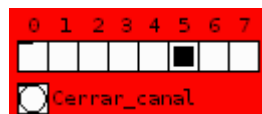
y el correspondiente bloque **Matriz** que es el que muestra el gráfico

Las propiedades del lienzo del grafico son:



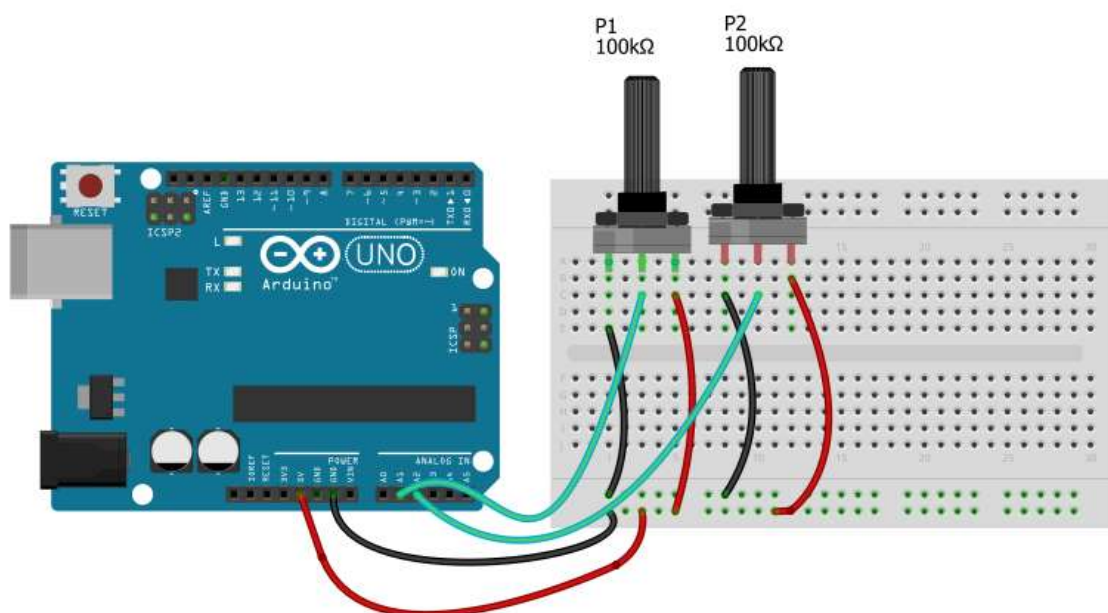
Configuraciones:

- Como siempre seleccionamos el puerto



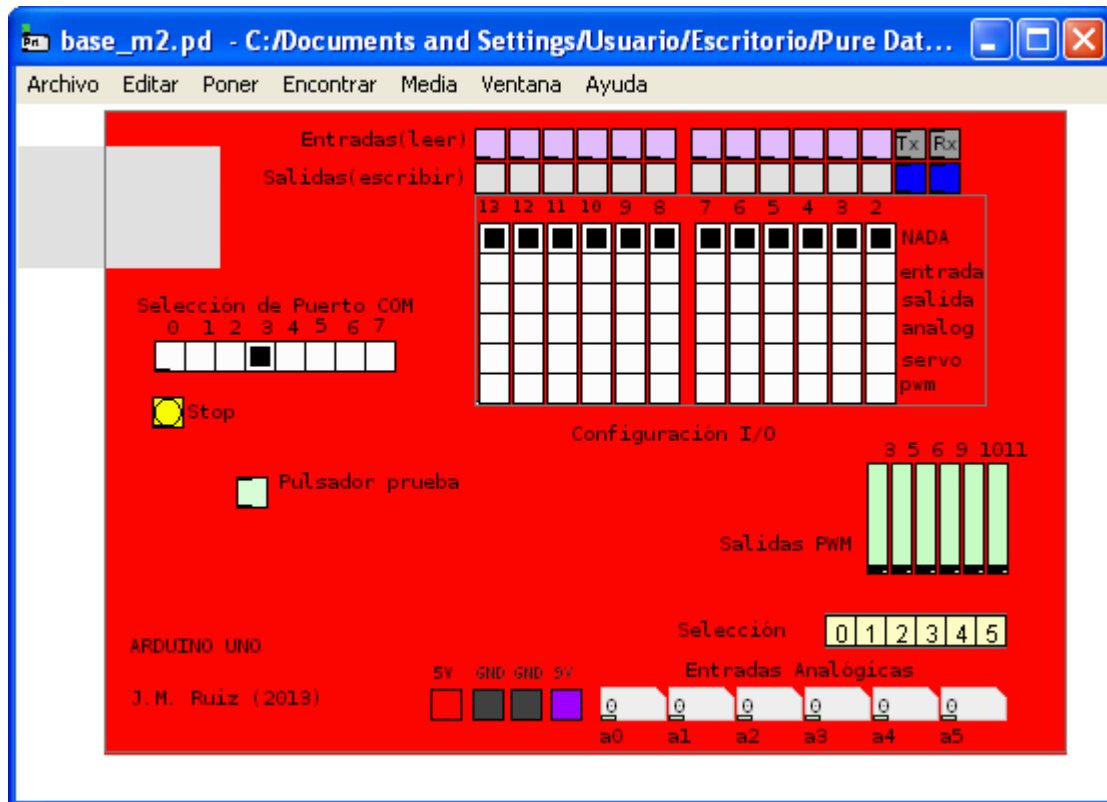
- Seleccionamos para leer los canales **a1** y **a2** **Activar Entradas analógicas**

Esquema de montaje:



METODO DE TRABAJO 2

En este método he incluido ciertas utilidades integradas en forma de **sub-patch** en el modelo de trabajo con el fin de facilitar la configuración y la versatilidad en el trabajo. A continuación vemos abierto el fichero **base_m2.pd** que contiene la base para escribir en ella nuestra aplicación (recuérdese que este fichero es de solo lectura).



Fichero: *base_m2.pd*

Uno de los objetivos de este modelo de trabajo con la librería **Pduino** es tener en el área de trabajo disponibles las señales propias de **Arduino** para poder conectar en las propias salidas los valores.

La denominación de las señales para ser tanto para leerlas como escribirles es la siguiente:

Entradas Digitales:

Leer_D2, Leer_D3... LeerD13 que se corresponden con lo Pines PIN2, PIN3, PIN4,...PIN13

Salidas Digitales:

Escribir_D2, Escribir_D3, Escribir_D4,.... Escribir_D13 que se corresponden igualmente con los pines PIN2, PIN3, PIN4,...PIN13

Entradas Analógicas:

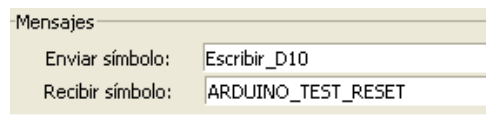
a0, a1, a2, a3, a4, a5 que se corresponden con los pines A0,A1, A2,A3, A4, A5 de **Arduino**

Salidas analógicas (PWM)

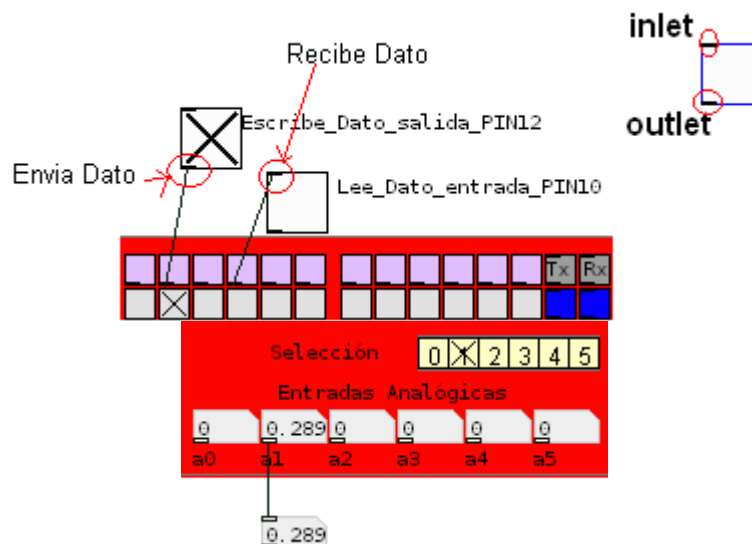
PWM3, PWM5, PWM6, PWM9, PWM10, PWM11 que se corresponderían con los pines digitales de **Arduino** PIN5, PIN6, PIN9, PIN10, PIN11

Acceso a las señales:

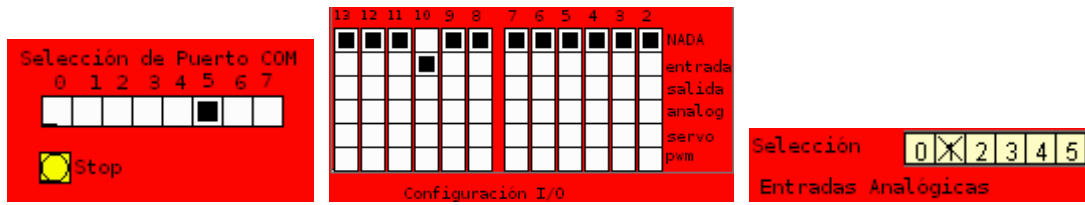
- Para acceder a las señales se puede hacer por los dos métodos: Mediante la opción de configuración de Propiedades del objeto que se trate y luego en la colocación del nombre de la variable que deseamos leer (Recibir símbolo) o en la que deseamos escribir un valor (Enviar símbolo).



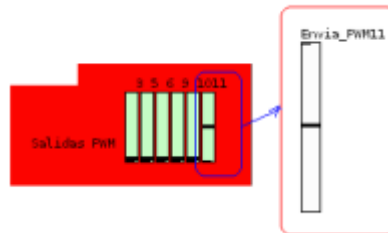
- La otra opción sería colocando directamente un cable de unión con la entrada o salida correspondiente, distinguiendo siempre lo que es una entrada (inlet) y una salida(outlet) en un bloque



En este modelo la selección del puerto se hace igualmente pulsando en el objeto multicaja y de la misma manera para configurar las E/S digitales se hace en las casillas correspondientes de configuración. La selección de canales analógicos a mostrar se hace en la multicaja “Selección”



Las salidas analógicas solo se pueden ver en los sliders pero no se puede realizar la conexión con ellos.

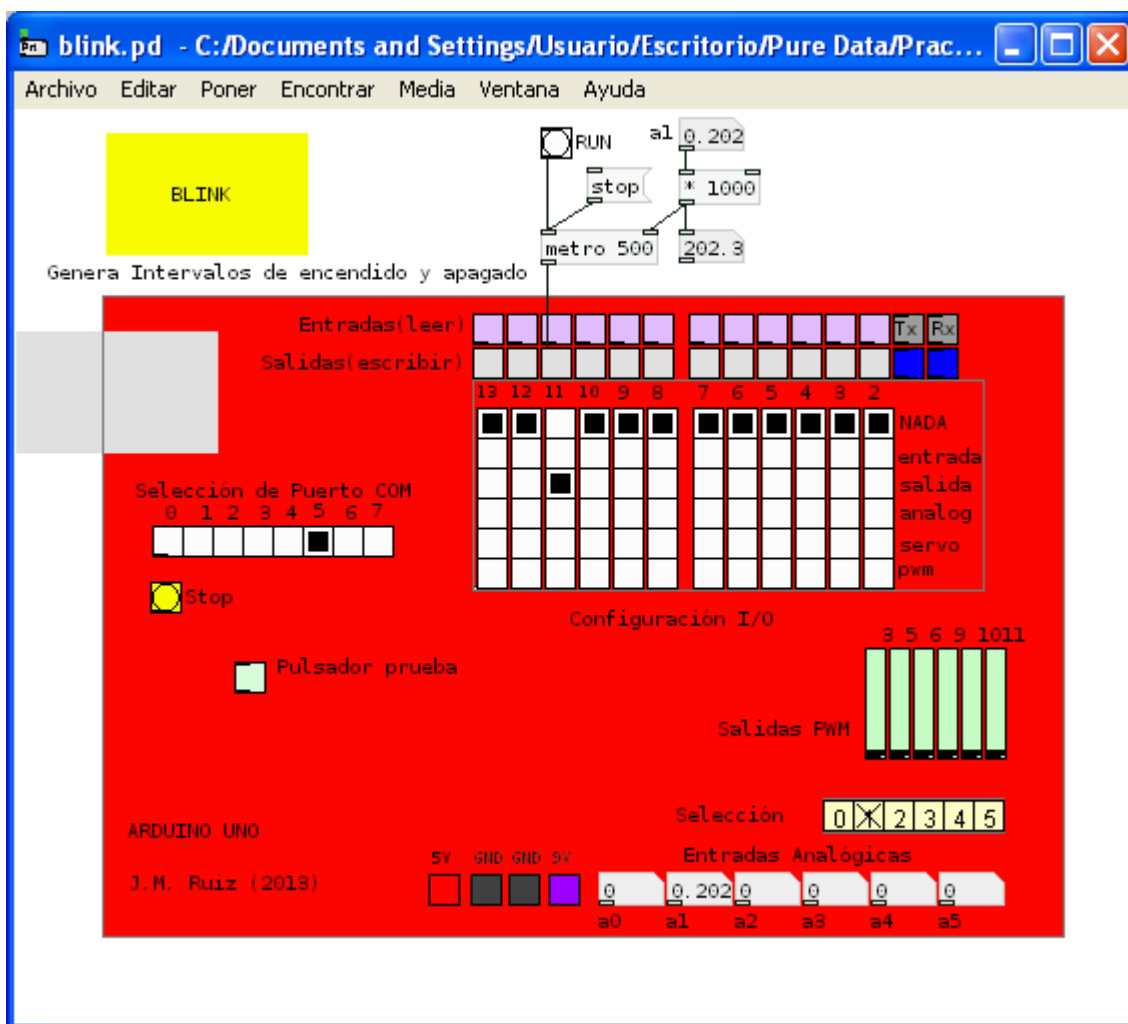


3.1.15. Blink

En este ejemplo primero del modelo de trabajo m2 hemos conectado a la salida PIN11 los bloque necesarios para que se envíe a esa salida una señal digital intermitente generada con la ayuda de un pulsador **Bang** y un bloque **metro** al que hemos colocado un tiempo que variaremos en función de la señal recibida del canal analógico **A1** (señal **a1**) que multiplicamos por 1000 para escalarlo desde (0 a 1) a (0 a 1024).

De la misma forma se ha colocado un **mensaje stop** para detener el envío de impulsos.

En este ejemplo hemos adoptado el método de cablearlo directo sobre el modelo de **Arduino m2**

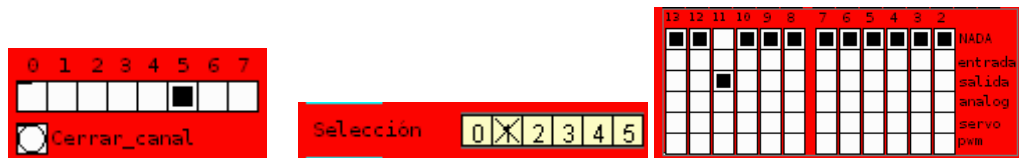


Fichero: *m21-blink.pd*

Como los pines de **Arduino** por defecto están en modo salida no hubiese echo falta poner el PIN11 como salida.

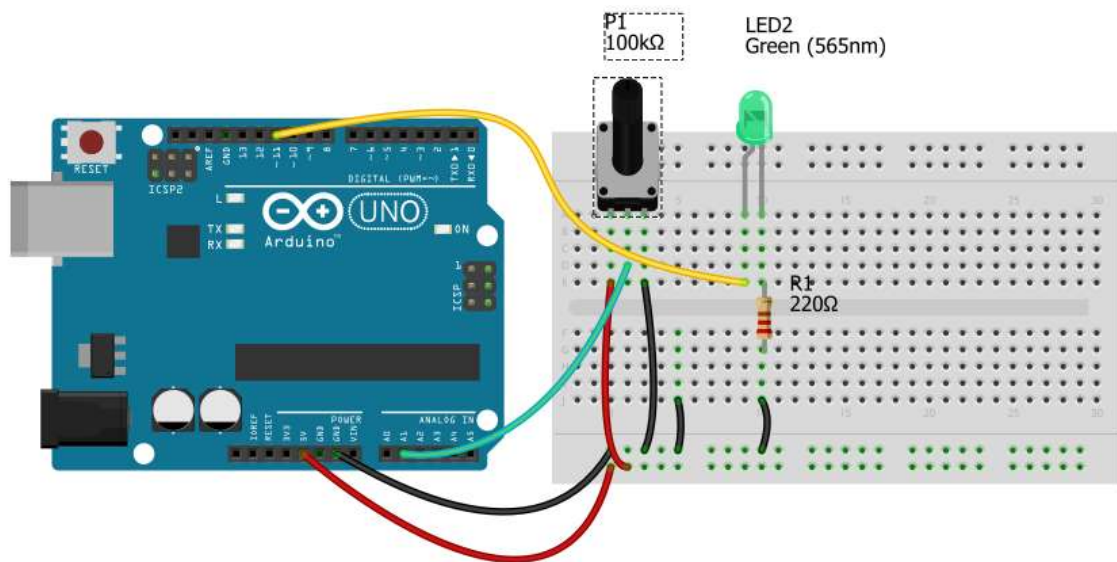
Configuraciones:

- Como siempre seleccionamos el puerto



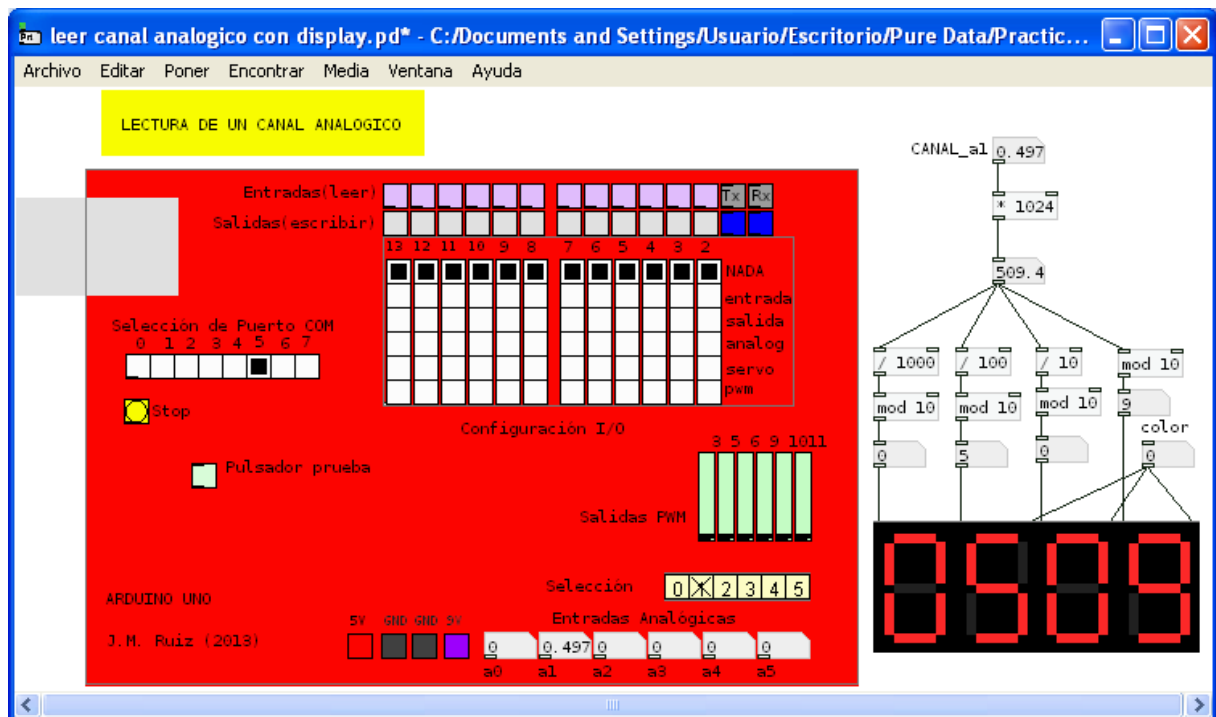
- Seleccionamos para leer los canales **a1** **Activar Entradas analógicas y la configuración de E/S digitales**

Esquema de montaje:

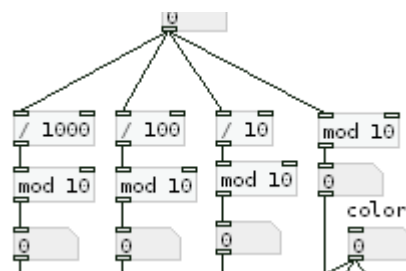


Lectura de un canal analógico y visualización en display

En este ejemplo deseamos leer el canal a1 CANAL_a1 0.497, lo escalamos al valor 0 a 1024 mediante un bloque de producto $\times 1024$ y después los mostramos en cuatro display independientes. La visualización se ha realizado en displays independientes y para ello hemos tenido que realizar la descomposición del número en millares, centenas, decenas y unidades



Fichero: *m22-leer canal analógico con display.pd*

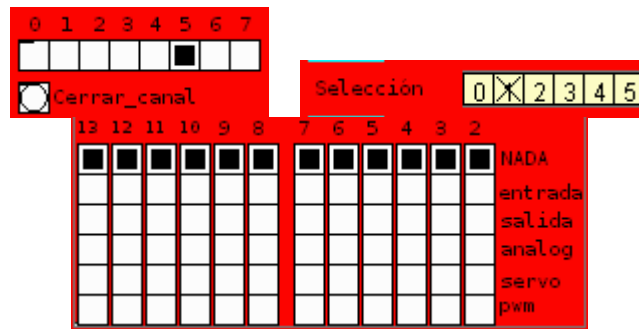


Este es el algoritmo para descomponer un numero y mostrarlo en cada display de manera autónoma

Los display están una librería que se encuentra entre los ficheros complementarios de este manual.

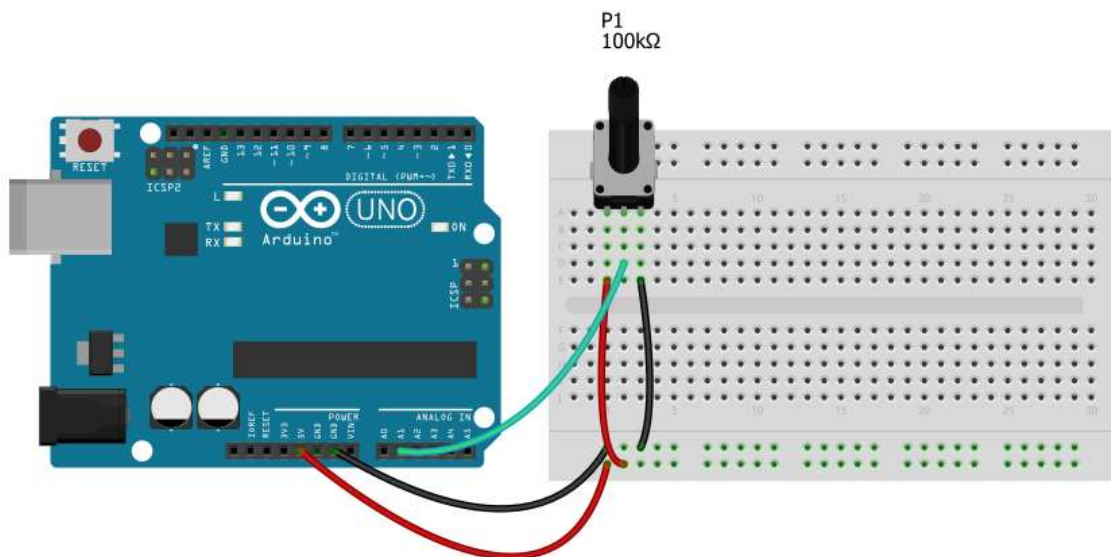
Configuraciones:

- Como siempre seleccionamos el puerto



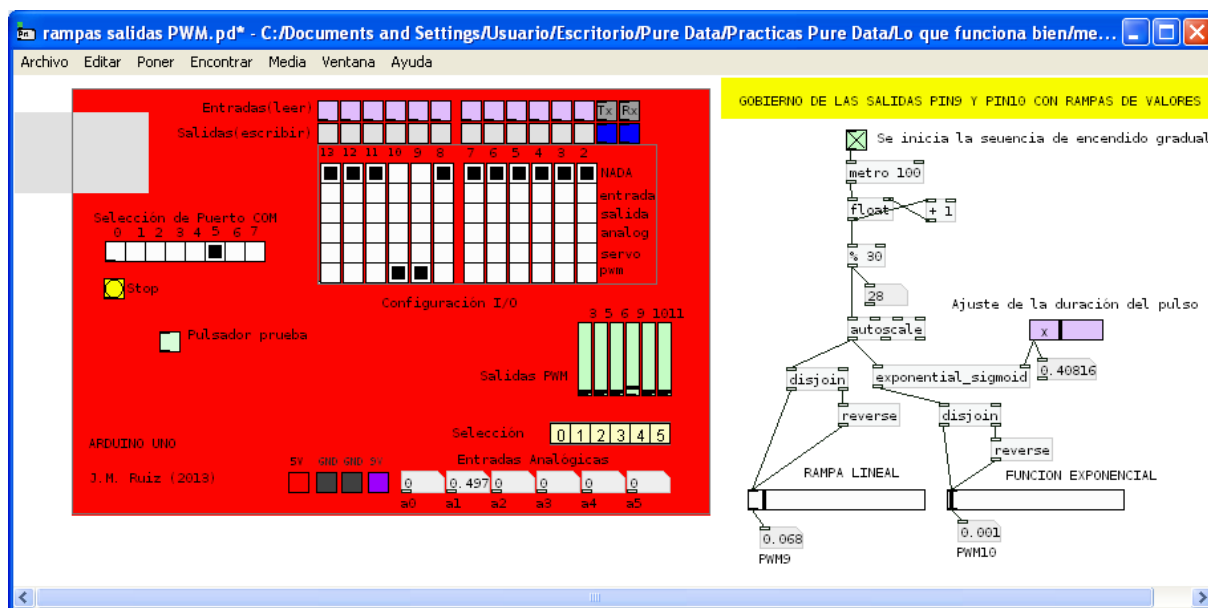
- Seleccionamos para leer los canales **a1** **Activar Entradas analógicas** y la configuración de **E/S digitales**

Esquema de montaje:



3.1.16. Rampas salidas PWM

Este ejemplo utiliza un patch realizado ya en el foro de PD. Básicamente de lo que se trata es de generar dos señales distintas, una en forma de rampa lineal y la otra de rampa exponencial. Mediante un bloque de tipo **metro** generamos un contador que se encarga de alimentar al resto de bloques. Dejamos la explicación del funcionamiento para otro momento



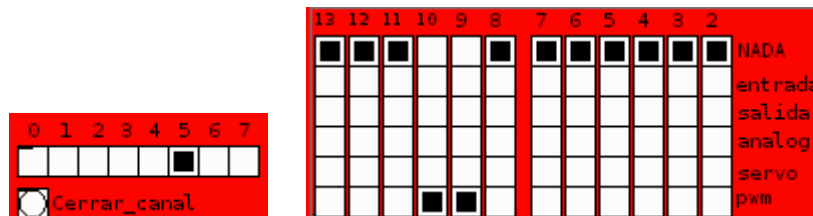
Fichero: *m23-rampas salidas.pd*

Las salidas de señal se llevan a dos **Vslider** que la muestran y en los que hemos configurado sus propiedades para el envío del valor las variables **PWM9** y **PWM10**.

Dejamos para el lector que realice las modificaciones oportunas para que el disparo de las señales se realice a través de una entrada digital.

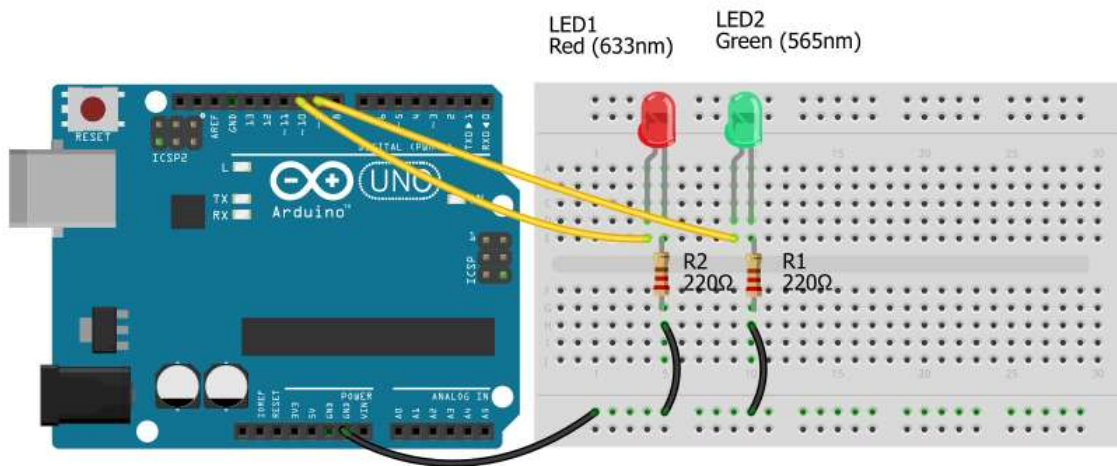
Configuraciones:

- Como siempre seleccionamos el puerto



- Seleccionamos para leer los pines PIN9 y PIN10 como salidas PWM

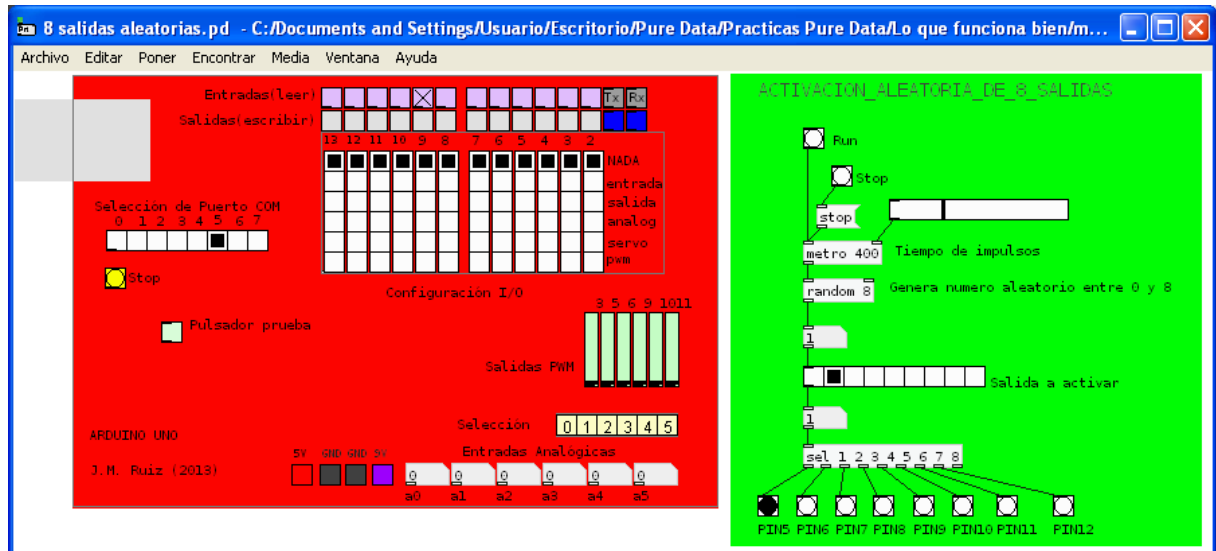
Esquema de montaje:



8 Salidas aleatorias

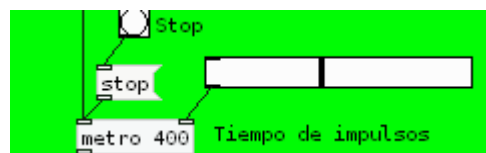
Esta aplicación consiste en activar las salidas digitales de **Arduino** de manera aleatoria. Se van a utilizar de salida los pines siguientes:

PIN5,PIN6,PIN7,PIN8,PIN9,PIN10,PIN11,PIN12



Fichero: m24-8 *salidas aleatorias.pd*

El funcionamiento es muy sencillo se genera una señal variable (impulso) **metro 400** controlada por el bloque metro que actúa sobre el bloque **random 8** que se encarga de generar un numero aleatorio entre 0 y 8.

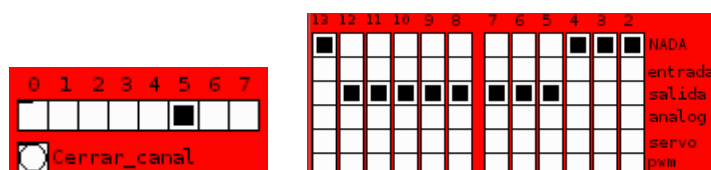


Se ha colocado un **Vslider** para controlar la velocidad de los impulsos generados por el bloque **metro**.

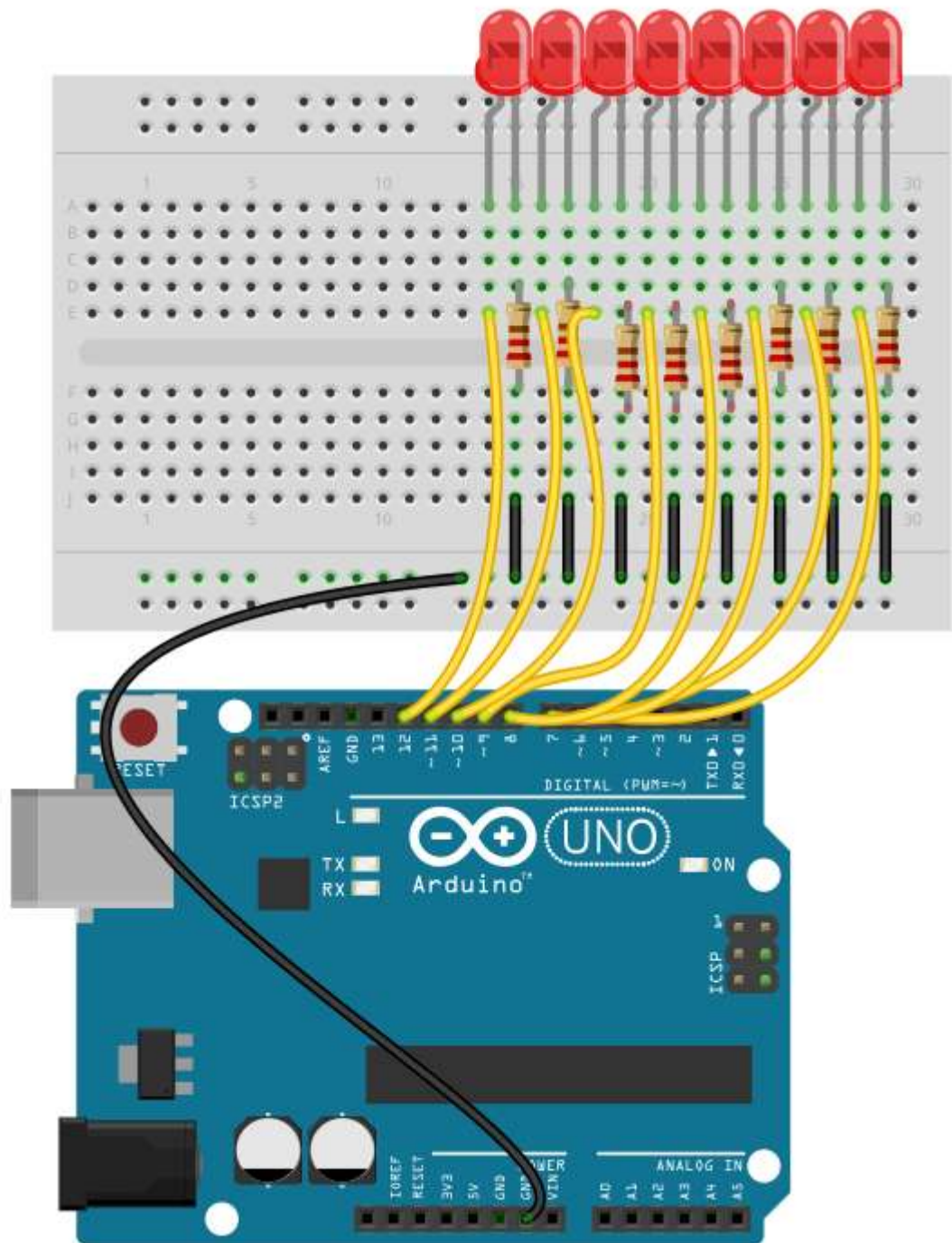
El número generado se lleva a un bloque **sel 1,2,3,4,5,6,7,8** que lo que hace es activar la salida en función de la entrada

Configuraciones:

- Como siempre seleccionamos el puerto y los pines de salida

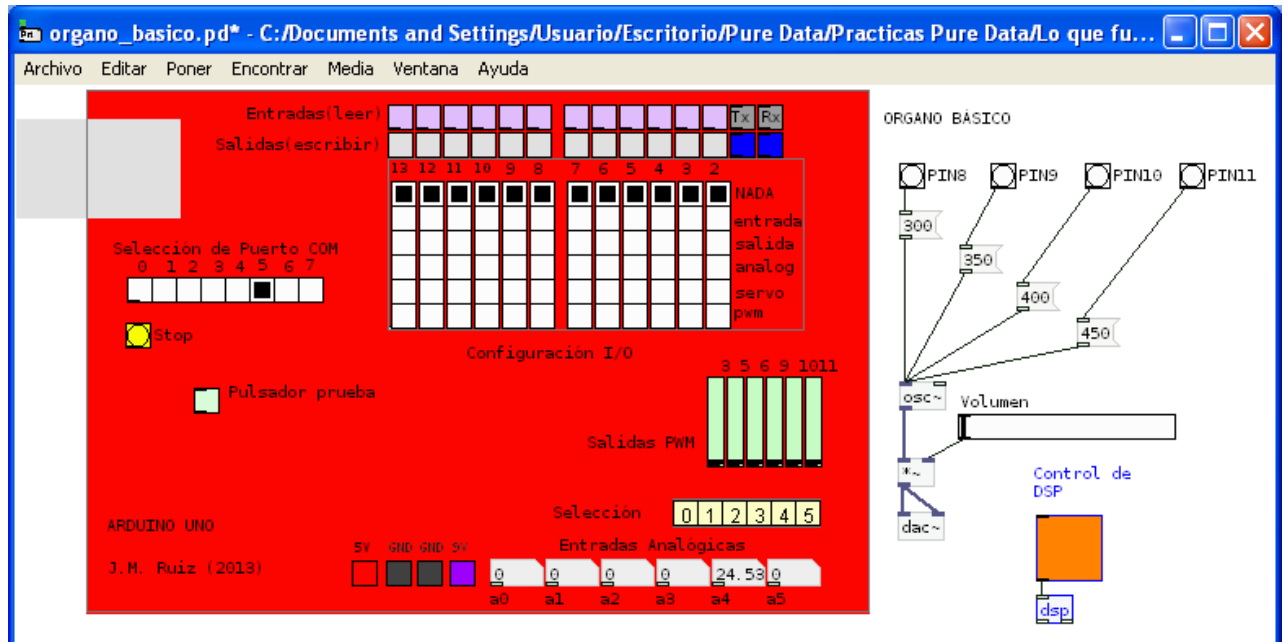


Esquema de montaje:



3.1.17. Órgano básico

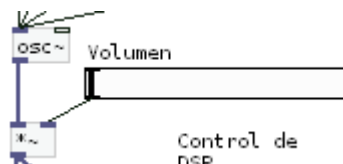
En este caso se trata de generar hasta cuatro tonos distintos de valores **300**, **350**, **400** y **450 Hz**. Mediante la ayuda de los pines PIN8, PIN9, PIN10, PIN11 que se configuraran como entradas digitales



Fichero *m25-organo_basico.pd*

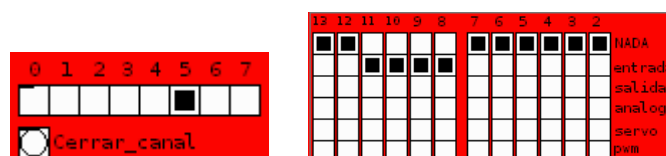
Se ha colocado el bloque **dsp** que activa y desactiva el sistema DSP de PD para poder escuchar en nuestro altavoz los sonidos.

Los sonidos los genera un bloque oscilador **osc~** que entrega su salida a un bloque producto **~*** que mediante el bloque **dac** envía señal de sonido. El volumen se controla con Vslider conectado al bloque de producto

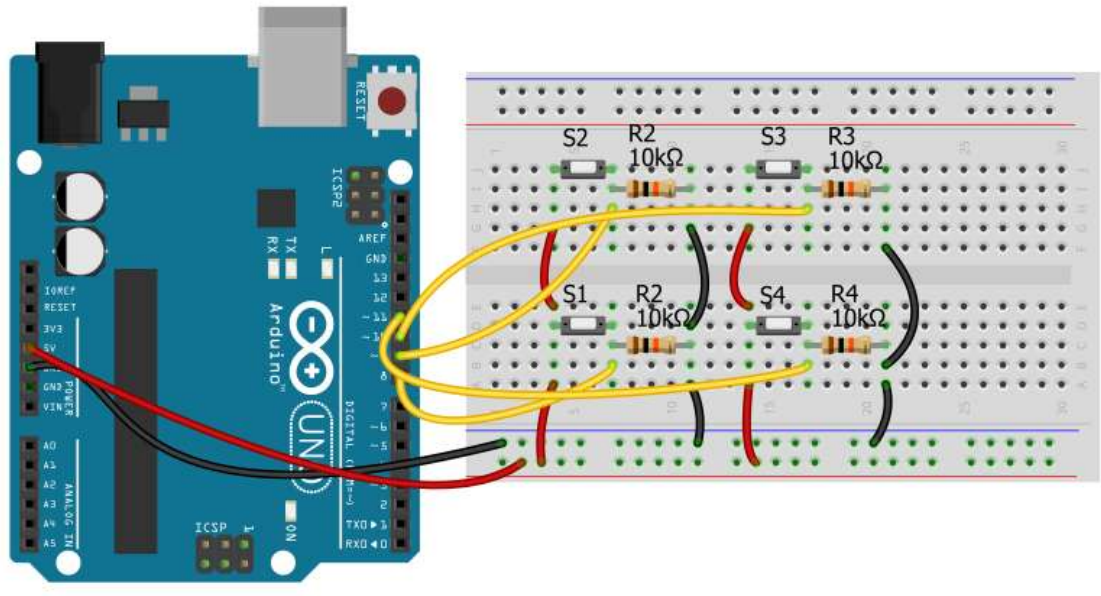


Configuraciones:

- Como siempre seleccionamos el puerto y los pines de salida

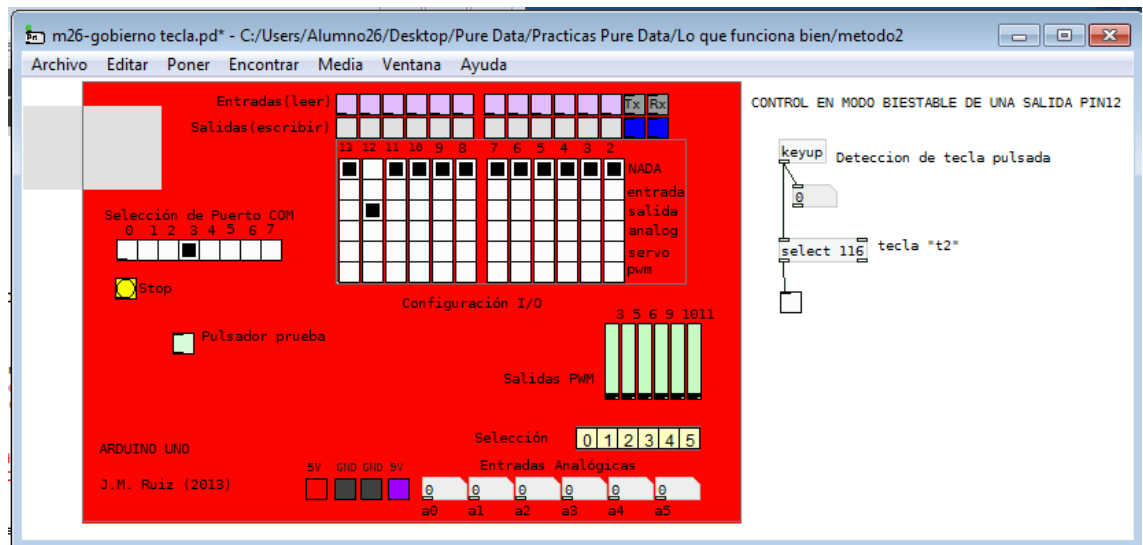


Esquema de montaje:



3.1.18. Control de una salida mediante una tecla en modo biestable

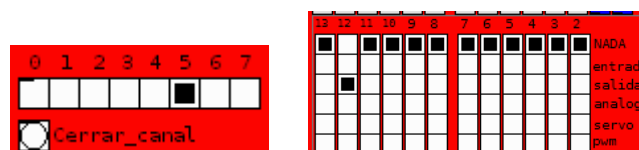
Con este ejemplo solo quiero demostrar que se puede controlar una salida de arduino pulsando una tecla. La función de Pure data a la que recurrimos es **Keyup** que recoge la tecla pulsada (código decimal del ASCII correspondiente). El valor lo entrega a un bloque de tipo **Select** que configurado con el valor "116" que equivalente a la letra minúscula "t" activa y desactiva la salida **PIN12** en modo biestable.



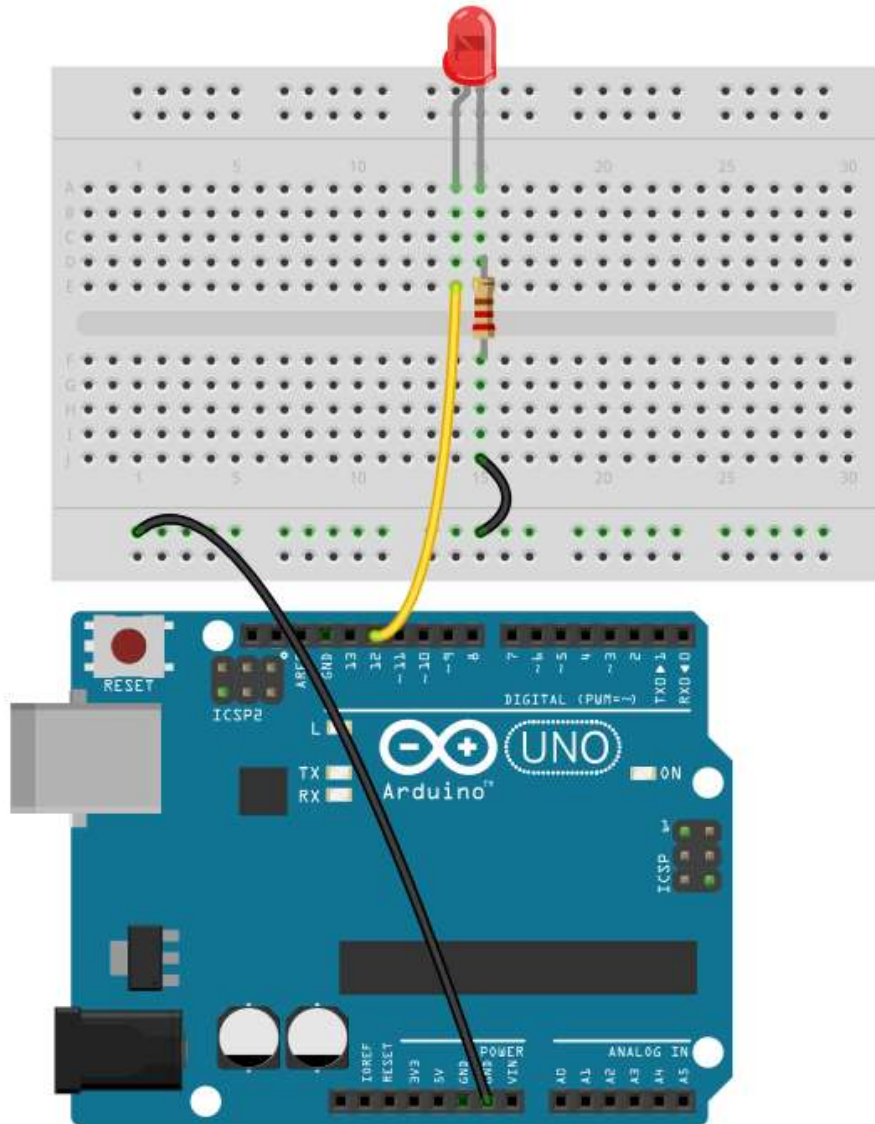
Fichero: *m26-gobierno tecla.pd*

Configuraciones:

- Como siempre seleccionamos el puerto y los pines de salida



Esquema de montaje:




4. Librerías Gráficas para Arduino

A continuación voy a mostrar unas librerías que he realizado para poder utilizar con **Arduino** facilitando la interacción con la Tarjeta así como algunas funciones interesantes para el usuario. Muchos de los módulos de estas librerías han sido tomados del Foro de **PD** y de algunos materiales libres encontrados en la Web.



Las librerías son ficheros **patch** de **PD** que son invocados y añadidos a nuestra aplicación facilitándonos el trabajo y dando importantes potencialidades a nuestras aplicaciones.

¿Cómo se invoca un fichero de librería? Para hacerlo basta con seleccionar **Poner-> Objeto** y cuando aparezca una caja con el cursor intermitente escribiremos el


nombre de  la librería. Es IMPORTANTE que el fichero con el que trabajamos este en una carpeta en la que también este el fichero de librería que invocamos. Si deseamos tener las librerías en una carpeta dentro de la carpeta de trabajo lo podemos hacer, en ese caso podremos el nombre de la carpeta de las librerías el carácter / y el

nombre de la librería .

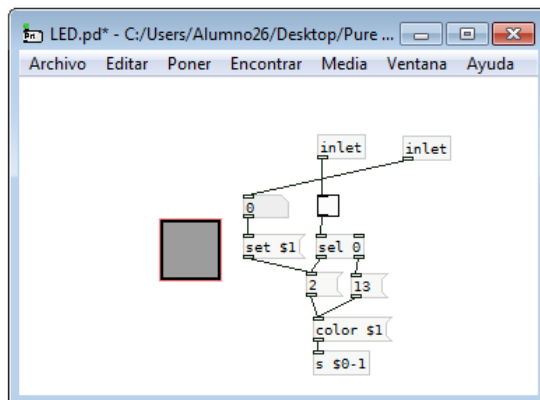
Pulsando fuera de la caja, una vez escrito el nombre, aparece la parte gráfica de la

librería  en este caso la librería tiene dos *inlets* (entradas) .

Si nos colocamos sobre la librería con el ratón y pulsamos el botón derecho nos

aparece un menú contextual en el que podemos elegir  Con la opción **Propiedades** podemos modificar algunas de las propiedades del “lienzo” que contiene el objeto de la librería y con la opción **Abrir** podemos abrir el fichero de la librería, en nuestro caso abriríamos *LED.pd* que es el fichero de la librería. Con **Ayuda** se abriría un fichero de ayuda si es que lo hubiese para mostrar ayuda sobre la librería (este fichero debería llamarse *LED-Help.pd*)

Este es el contenido de la librería LED una vez que selecciono **Abrir** es el siguiente

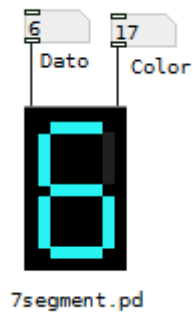


Listado de librerías que se incluyen en esta versión del manual Arduino+Pure Data

Display de 7 segmentos.

Se trata de un sencillo display al que le entregamos un numero de 0 a 9 y nos lo muestra encendiendo los segmentos correspondientes. Los *inlets* son: Dato en el que ponemos el dato y cColor en el que ponemos un numero con el que cambiamos el color de los segmentos. El número será de 0 a 29 para la paleta básica de colores.

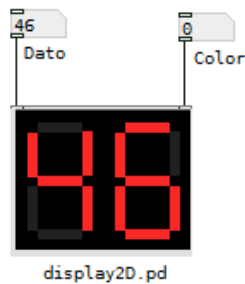
Display 1 Dígito



Display doble.

Se trata de un display formado por dos dígitos al que podremos poner un número de entrada de 0 a 99. Tiene igualmente un *inlet* de entrada para cambiar el color.

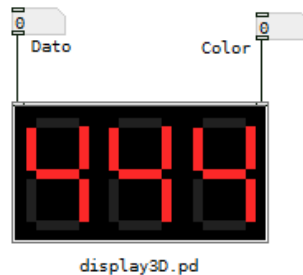
Display 2 Dígitos



Display triple.

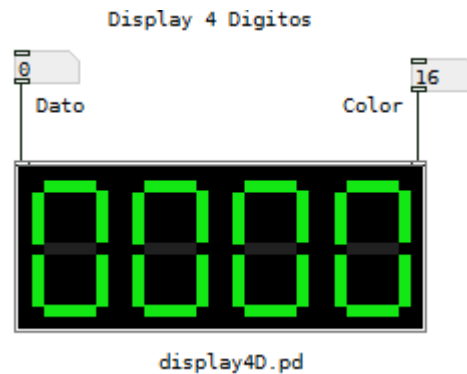
Se trata de un display formado por tres dígitos al que podremos poner un número de entrada de 0 a 999. Tiene igualmente un *inlet* de entrada para cambiar el color.

Display 3 Dígitos



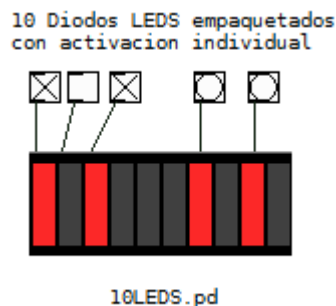
Display cuádruple.

Se trata de un display formado por tres dígitos al que podremos poner un número de entrada de 0 a 9999. Tiene igualmente un *inlet* de entrada para cambiar el color.



Barra de 10 LEDs independientes.

Esta librería incluye hasta 10 LED con excitación independiente que se pueden gobernar bien con un objeto *Bang* o con un *Toggle*.

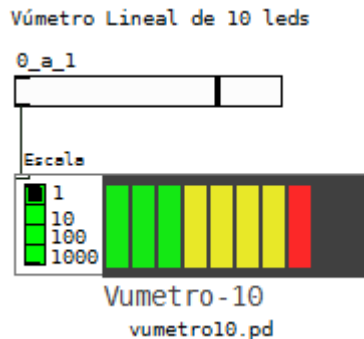


LED sencillo

Se trata de un simple, LED con dos entradas una para encendido y apagado y la otra para el color del estado de apagado (fondo)



Vúmetro de 10 LEDs



Este objeto permite el encendido gradual de hasta 10 leds de colores en función de la entrada. Dispone de un selector de escala para poder leer valores de:

- 1: de 0 a 1
- 10: de 0 a 10
- 100: de 0 a 100
- 1000: de 0 a 1000

Generador de Señal

Este objeto es muy interesante, permite dibujar un perfil de señal de entrada en una venta grafica que se convierte en un **array** de datos que podremos sacar luego por su **oulet** de salida.

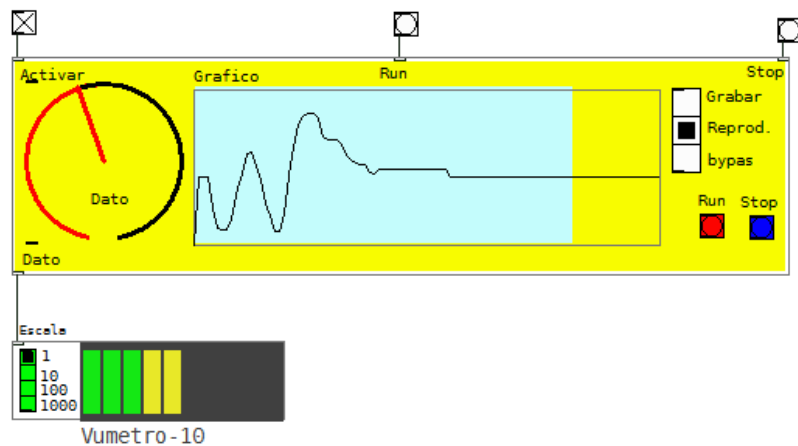
Tiene tres modos de trabajo:

Grabar: Editamos y grabamos el array de datos, simplemente deslizando el ratón sobre el area gráfica manteniendo pulsado el botón izquierdo.

Reproducir: Reproduce el array cargado previamente. Sacando los valores por el oulet de salida.

Bypass: detiene el envío de los datos a la salida aunque continúa con el barrido del array

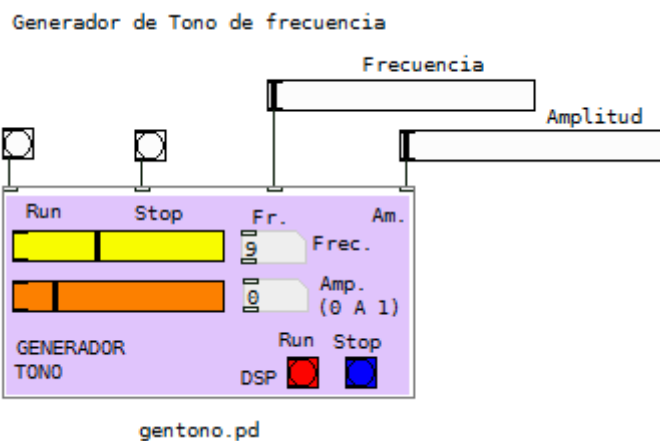
Para iniciar el funcionamiento se debe activar con una señal **Toggle Activar** en su *inlet* de entrada. La grabación y reproducción de los datos necesita la activación del sistema DSP de **PD** para ello se recurre a los botones **Run** y **Stop** del bloque que también se han puesto como *inlets* de entrada.



Fichero *genera_señal.pd*

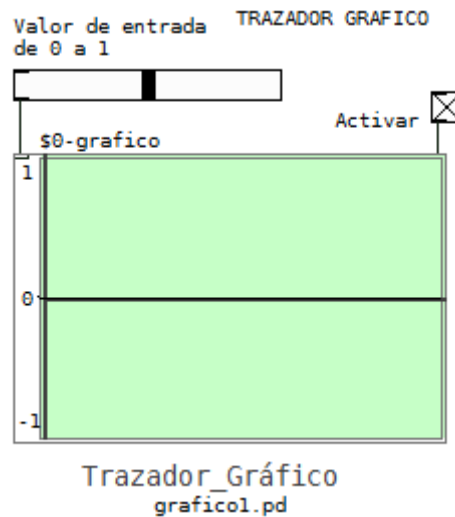
Generador de Tono de Frecuencia

Esta librería activa la función el sistema **DSP** de **PD** y generar un tono del que podemos variar la frecuencia y la amplitud. Posee dos entradas para estos valores y otras dos para arranque **Run** y parada **Stop** de la generación del tono. Es posible también manipular estas variables desde los objetos integrados en la propia imagen de la librería.



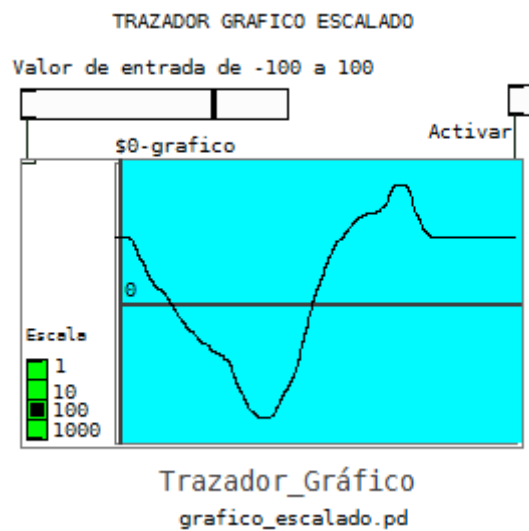
Trazador Gráfico

Con el Trazador Gráfico podemos registrar en un gráfico una señal que introducimos por la entrada correspondiente de un valor comprendido entre 0 y 1 siempre que tengamos activado el *inlet* Activar.



Trazador Gráfico escalado

Este objeto de librería es una versión más, como en la del anterior, en la que se ha facilitado el escalado de los rangos de datos de entrada hasta cuatro escalas. De esta manera podemos visualizar una amplia gama de datos



Botón UP/DOWN

Este objeto permite incrementar o decrementar un valor numérico simplemente poniendo el ratón sobre el botón Up p Down y manteniendo pulsada la tecla izquierda.

Boton UP/DOWN



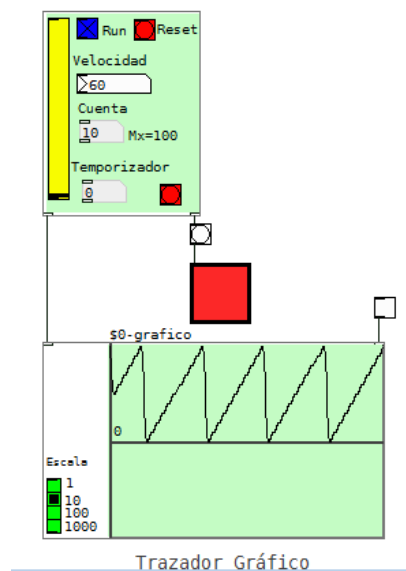
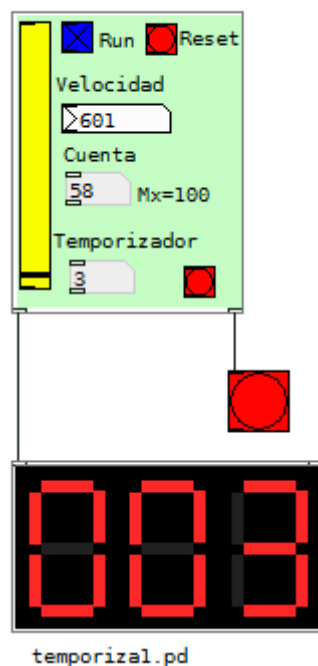
botonUP.pd

Temporizador Básico

El temporizador genera cuando llega a un valor determinado un impulso en su salida. En realidad es un generador de rampa en el que podemos variar los escalones y el tiempo entre escalón y escalón.

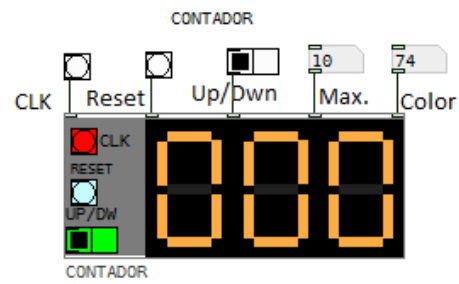
La variable **Cuenta** es el número de escalones que debe contar. La variable **Velocidad** es el tiempo entre escalón y escalón. Los botones **Run** y **Reset** son para arrancar y reiniciar la cuenta respectivamente. La variable Temporizador indica el estado de cuenta y es realmente la salida **Dato**. La salida *outlet* de la derecha es la señal de disparo del final de cuenta

TEMPORIZADOR BASICO



Contador

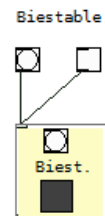
El contador como su nombre indica cuenta impulsos que le llegan por a su entrada CLK. Puede contar hacia adelante y hacia atrás, puede contar con limitación y valor máximo y se le puede cambiar el color de los dígitos. También se puede resetear



contador.pd

Biestable

Es una librería que implementa un biestable




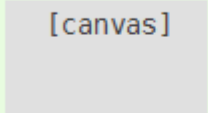
biestable.pd

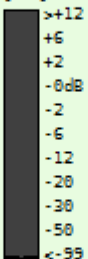
5. Otras Librerías Gráficas interesantes que incluye PD

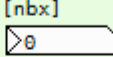
gui_objects


Pd comes with a standard, built-in set of GUI elements, which you can select from the "Put" menu.


 [bng]

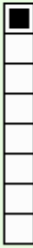
 [canvas]


 [vu]


 [nbx]

 [tgl]


 [vslider]

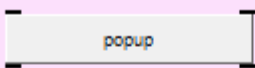
 [vradio]


 [hradio]

 [hslider]

There are also some OS-native GUI elements available

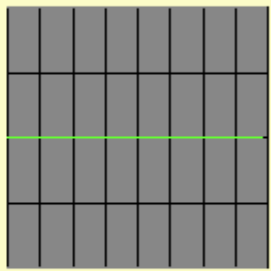
 [button]

 [popup]


 [ticker]

And various others:


[cyclone/Scope~]



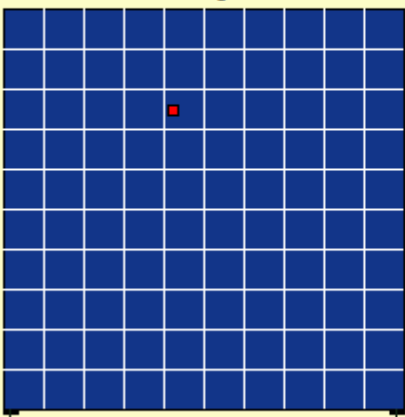
[gcanvas]



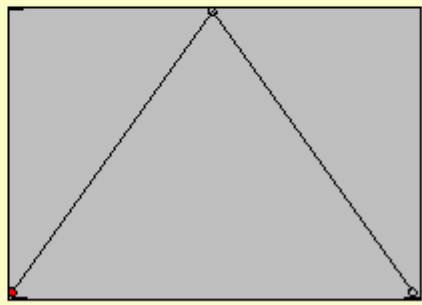
[knob]



[unauthorized/grid]



[envgen]



Bibliografía:

Página Personal de creador de PD Software [Miller Puckette](http://www.crca.ucsd.edu/~msp/index.htm)
<http://www.crca.ucsd.edu/~msp/index.htm>

Página Principal de PD: <http://puredata.info/>

Libro "Loadbang" de Johannes Kreidler traducción al español PD
http://www.filefront.com/16053499/Johannes-Kreidler_Tutorial-PD.zip/

Pure data FlossManuals. <http://en.flossmanuals.net/puredata/>

Generación Musical y Visual
http://creaciodigital.upf.edu/wikis/gmv/index.php/Main_Page#Generaci.C3.B3n_Musical_y_Visual

Documentación en línea de PD. Puckette, M. S
http://www-crca.ucsd.edu/~msp/Pd_documentation/index.htm

Theory and Techniques of Electronic Music. Puckette, M. S.
<http://www-crca.ucsd.edu/~msp/techniques.htm>

Página HTLM con documentación sobre PD:
<http://www.crca.ucsd.edu/~msp/software.html> .
http://crca.ucsd.edu/~msp/Pd_documentation/

Conexión **PD** y **Arduino** a través del puerto serie. nf.interactive

Arduino Playground: <http://playground.arduino.cc/Interfacing/PD>

Firmata: <http://arduino.cc/es/Reference/Firmata> http://firmata.org/wiki/Main_Page

Pduino: Hans-Christoph Steiner's <http://at.or.at/hans/pd/objects.html>

Arduino-PureData-MessageSystem: <http://kiilo.org/tiki/tiki-index.php?page=Arduino-PureData-MessageSystem>

“Pduino 0.5.beta8” que se puede descargar en [pduino 0.5.beta8](http://pduino.0.5.beta8)

Foro Pure Data: <http://puredata.hurlleur.com/index.php>

Workshops:

Museo de la Universidad de Alicante: <http://muaworkshops.d3cod3.org/>

PD + Arduino: <http://cargocollective.com/max-pd-tutorial/pd-arduino>

Physical Programming

http://www.openobject.org/physicalprogramming/Student_Projects

Paja: <http://mlab.taik.fi/paja/?p=118>

Physical Modelling for pd: <http://drpichon.free.fr/pmpd/>

Puredata sound tutorials: <http://obiwannabe.co.uk/html/sound-design/sound-design-all.html>

Paper Computing Tutorial

http://ocw.mit.edu/courses/media-arts-and-sciences/mas-714j-technologies-for-creative-learning-fall-2009/assignments/paper_kit/

Software utilizado en este trabajo

IDE Aduino Ver. 1.0.5 Que se puede descargar en <http://arduino.cc/en/Main/Software>

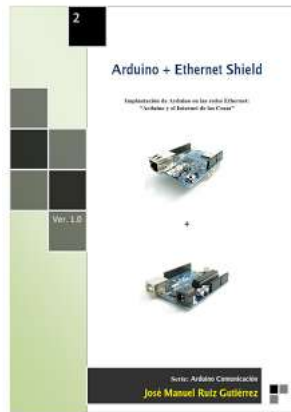
PD Ver 0.43-4 Extended. Que se puede descargar en <http://puredata.info/downloads>

La libreria Pduino. Que se puede descargar en <http://at.or.at/hans/pd/objects.html>

El firmware Firmata Que se puede descargar en http://firmata.org/wiki/Main_Page

Todas las prácticas se han realizado en un PC Compatible con Windows XP instalado y la tarjeta **Arduino Uno** Ver.3

Mis anteriores publicaciones sobre Arduino



Arduino + Ethernet Shield

Nº 2 Serie: Arduino Comunicación



Arduino+XBee

Nº 1 Serie: Arduino Comunicación

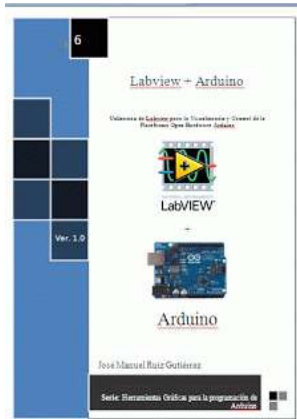


Manejo y Aplicaciones del Bus I2C de Arduino

Nº 1 Serie Monografías aplicaciones de Arduino

LabVIEW+ Arduino

Nº 6 Serie: Herramientas Gráficas para la programación de Arduino



Minibloq + Arduino

Nº 5 Serie: Herramientas Gráficas para la programación de Arduino



S4A (Scratch) + arduino

Nº 4 Serie: Herramientas Gráficas para la programación de Arduino





IDE Arduino + Ardubloq

Nº 3 Serie: Herramientas Gráficas para la programación de Arduino



Arduino + MyOpenLab

Nº 2 Serie: Herramientas Gráficas para la programación de Arduino



Herramientas de Programación Gráfica de Arduino

Nº 1 Serie: Herramientas Gráficas para la programación de Arduino



Nº 1 Serie Practicas de Arduino



Nº 1 Serie Traducciones de Textos



Nº 2 Serie Practicas de Arduino



Este trabajo está bajo licencia [Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by-nc-nd/3.0/)