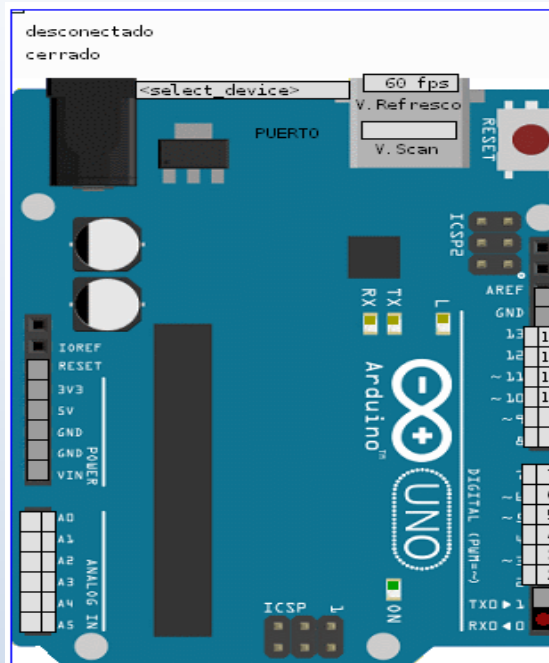


Anexo I
a
Nº7

Arduino + Pure Data

ANEXO I

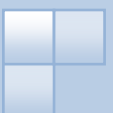
Desarrollos de la librería Pduino



Arduino_Gui

José Manuel Ruiz Gutiérrez

Serie: Herramientas Gráficas para la Programación de
Arduino



Ver. 1.0

INDICE

0. Introducción

1. Librería Pduino

- 1.1. Patch *arduino_help*
- 1.2. Subpatches *arduino_help*
 - 1.2.1. Patch *pd PIN_MODES*
 - 1.2.2. Patch *pd PIN_PROPIERTIES_INFO*
 - 1.2.3. Patch *pd OUPUT_DIGITAL*
 - 1.2.4. Patch *pd OUPUT_PWM*
 - 1.2.5. Patch *pd OUPUT_SERVO*
 - 1.2.6. Patch *pd INPUT_DIGITAL*
 - 1.2.7. Patch *pd INPUT_ANALOG*
 - 1.2.8. Path *pd REFERENCE*

2. Librería *arduino_gui*

- 2.1. Patch *arduino-gui*
 - 2.1.1. Nombramiento de variables:
- 2.2. Patch *arduino-gui-help*

3. Empezamos

- 3.1. Escribir una señal digital
- 3.2. Lectura de una entrada digital desde Arduino a Pure data
- 3.3. Blink
- 3.4. Blink Doble
- 3.5. Contador Básico
- 3.6. Lectura de señales analógicas
- 3.7. Lectura de señales analógicas 2
- 3.8. Activación de salidas en modo SERVO
- 3.9. Salidas PWM
- 3.10. Semáforo



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

0. Introducción

Arduino_Gui es una aplicación basada en **Pduino** que desarrolla algunas utilidades para facilitar la utilización de la Plataforma Open Hardware Arduino unida a Pure Data.

Los autores de esta aplicación son:

Roman Haefeli reduzent@gmail.com

Georg Holzmann grh@mur.at

Hans-Christoph Steiner hans@at.or.at

Gerda Strobl gerda.strobl@student.tugraz.at

Olsen Wolf sesselastronaut@googlemail.com

Esta es una traducción de la versión original con algunas modificaciones y ampliaciones. Se trata de la realización de un Pduino, mejorado. Todos los patches de Arduino-Gui se basan en el Pduino oficial (versión 0.5beta8) mantenido por [Hans-Christoph Steiner](#).

Para acceder a la librería Pduino desde la página oficial de Pure Data se puede hacer desde esta dirección:

<http://puredata.info/downloads/pduino>

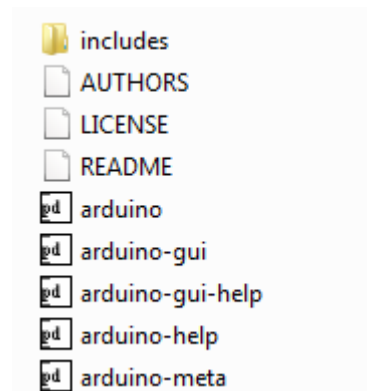
<http://puredata.info/downloads/pduino/releases>

El lugar de descarga de la aplicación Pduino versión 0.5beta8 es :

<https://github.com/EpicJefferson/Intro-to-puredata>

La librería utilizada como Firmware es Firmata 2.2 ubicada en: [Firmata-2.2.zip](#)

Para utilizar este patch de Pure Data es necesario tener acceso a los patch *arduino* y *arduino-help* que son los originales de Pduino. En la siguiente imagen se muestran los contenidos mínimos de la carpeta de trabajo:



Los patch y subpatch que se van a describir han sido traducidos de la versión original y en todo momento este trabajo se acogerá a las características de ser un software GNU.

La carpeta **includes** contiene patch que son utilizados por la aplicación **arduino-gui**.

Describimos el contenido de los principales ficheros:

arduino: Patch que contiene el elemento básico de comunicación con la tarjeta Arduino.

arduino-help Contiene el fichero de ayuda de objeto **arduino** de la librería.

arduino-gui contiene el patch que se convierte en el nuevo objeto principal de trabajo

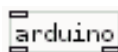
arduino-gui-help contiene el fichero de ayuda del objeto **arduino-gui**

arduino-meta Contiene los datos identificativos del autor de la librería Pduino

1. Librería Pduino

Esta librería se creó para comunicar **Arduino** con **Pure Data** y sobre ella se han realizado diversas modificaciones y aplicaciones de las que vamos a explicar dos de ellas. Antes de nada veamos de una manera básica la funcionalidad de esta librería **Pduino**.

El objeto que incluye la librería **arduino** se invoca sencillamente con su nombre:



He traducido el fichero *arduino-help* que viene en la librería con el fin de explicar su funcionamiento. En la carpeta de trabajo de **Arduino-Gui** se incluyen los dos ficheros: *arduino.pd* y *arduino-help.pd* que son los que vienen con la aplicación **Pure Data**.

1.1. Patch *arduino-help*

<p>1) Instalación</p> <pre>pd INSTALL_NOTES pd IMPORTANT_NOTES <- leer por favor</pre>	<pre>::::_ [arduino] _:::: Interface Arduino con Pure Data</pre>
<p>2) Conexión al puerto serie</p> <pre>devices <- lista de puertos -> chequea en la salida de PD Console! 0 <- poner el numero del puerto serie listados mediante el mensaje-comando 'device' open \$1 <- abrir el puerto serie de la tarjeta Arduino close <- cerrar el puerto serie version <- recoge la versión del protocolo firmware <- recoge la especificación del firmware send \$0-arduino</pre>	<pre>receive \$0-arduino arduino 1 <- Este es el Objeto Arduino send \$0-arduino-info send \$0-arduino-out</pre>
<p>3) Configura PIN, enviando mensajes</p> <pre>pd PIN_MODES <- configura el modo de trabajo del PIN pd PIN_PROPERTIES_INFO</pre> <p>Pure Data -> Arduino Arduino -> Pure Data</p> <pre>pd OUTPUT_DIGITAL pd INPUT_DIGITAL pd OUTPUT_PWM pd INPUT_ANALOG pd OUTPUT_SERVO</pre>	<p>EJEMPLO</p> <p>Enciende y apaga el LED del PIN13</p> <p>comment</p> <pre>pinMode 13 output <- configura pin 13 como salida digital 13 \$1 <- activa y desactiva el PIN 13 send \$0-arduino</pre>
<p>4) Información suplementaria</p> <pre>pd REFERENCE pd PERSISTENT_DEVICENAMES_IN_LINUX</pre>	<p>2012, Roman Haefeli Olsen Wolf 2006, Georg Holzmann Gerda Strobl Traducido y adaptado J.M.Ruiz 2013</p>

En este patch se recoge una importante ayuda sobre la utilización de la librería **Pduino** con las utilidades más importantes que nos permiten el dialogo y la gestión de las

comunicaciones con el “objeto” **arduino** que es en definitiva el núcleo de la comunicación. Recordemos que el firmware que posibilita esa conexión es **Firmdata**.

En el protocolo de la conexión con el puerto se utilizan una serie de comandos que se envían en forma de “mensajes” haciendo uso de los correspondientes bloques de función “Mensaje” de **Pure Data**.

Estos comandos son:

- “device” Que interroga sobre los puertos presentes en el sistema.
- “open \$1” Abre el numero d puerto indicado en la variable \$1
- “close” Cierra el puerto que tengamos abierto
- “versión” Interroga sobre la versión del protocolo de comunicación
- “firmware” Interroga sobre la versión de firmware instalada en el tarjeta **Arduino**

1.2. Subpatches de arduino-help

1.2.1. Patch *pd PIN_MODES*

PIN MODO

La mayoría de los pines de la placa Arduino soportan diferentes modos. Siempre tendremos que decirle a ARDUINO en qué modo va a utilizar un determinado pin. Esto se logra con el comando 'pinMode':

```
pinMode <pin#> <mode>
```

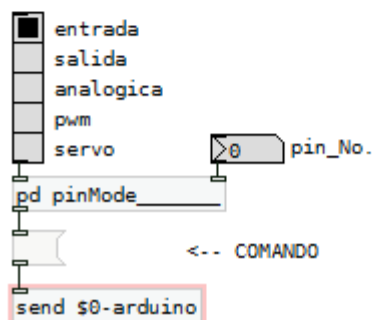
NOTA:

Cada PIN admite sólo un subconjunto de los modos disponibles.

LIST OF AVAILABLE PIN MODES

pinMode 11 input	==	pinMode 11 0	entrada digital
pinMode 11 output	==	pinMode 11 1	salida digital
pinMode 16 analog	==	pinMode 16 2	entrada analógica
pinMode 11 pwm	==	pinMode 11 3	salida modulacion anchura pulsos PWM
pinMode 11 servo	==	pinMode 11 4	servo (salida)

MODULO CONFIGURACIÓN DE PIN



Este subpatch nos muestra la manera de configurar el modo de trabajo de los pines de Arduino. La sintaxis del comando enviado al objeto arduino es muy sencilla:

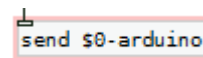
pinMode X Y X representa el numero de PIN e Y el modo de trabajo

- 0 Entrada Digital
- 1 Salida Digital
- 2 Entrada analógica
- 3 Salida PWM
- 4 Salida Servo

Es posible también utilizar las palabras: **input**, **output**, **analog**, **pwm** y **servo**

No olvidemos que el mensaje se envía al objeto **Arduino** mediante el objeto

send \$0 arduino



1.2.2. Patch *pd PIN_PROPIERTIES_INFO*

MAPA DE NUMERACION DE PIN

PUERTO<->MAPA PIN

port 0 : pins 2 - 7
port 1 : pins 8 - 15
port 2 : pins 16 - 19

PINEADO PARA
Arduino, NG, Duemilanove, Diecimila, UNO

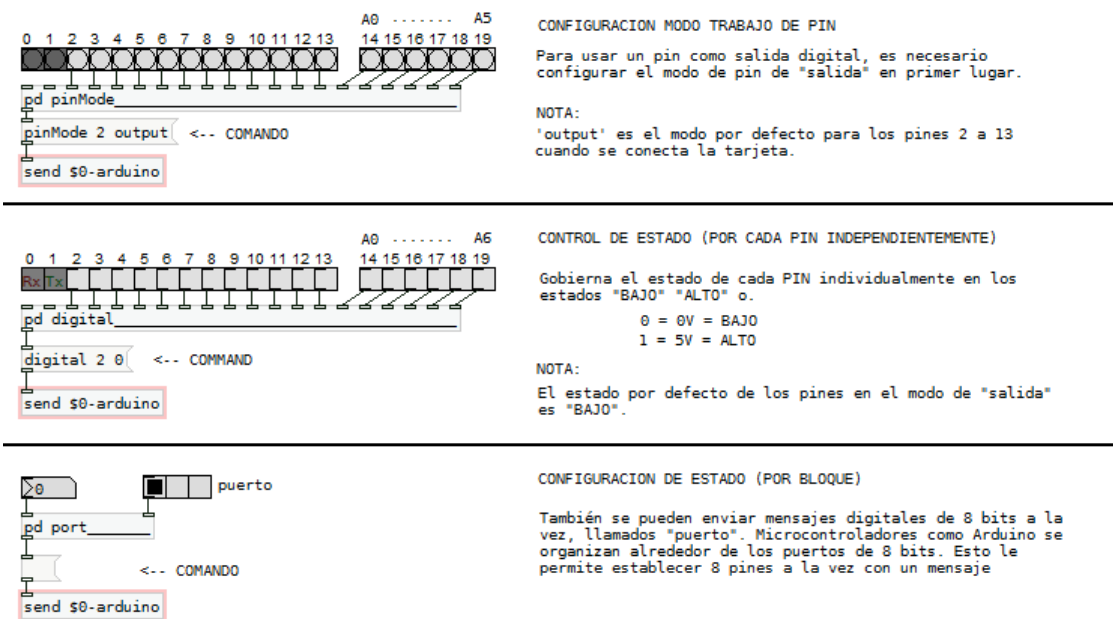
red: _Firmata_v2.2
white: _Firmata_v2.3

TABLA SOPORTE DE MODO PIN(FIRMATA 2.2 AND 2.3)

PIN	INPUT	OUTPUT	ANALOG	PWM	SERVO
2	X	X			X
3	X	X		X	X
4	X	X			X
5	X	X		X	X
6	X	X		X	X
7	X	X			X
8	X	X			X
9	X	X		X	X
10	X	X		X	X
11	X	X		X	X
12	X	X			X
13	X	X			X
14	X	X	X		ONLY 2.3
15	X	X	X		ONLY 2.3
16	X	X	X		
17	X	X	X		
18	X	X	X		
19	X	X	X		
20	X	X	X		ONLY 2.2
21	X	X	X		ONLY 2.2

En este patch se muestra información sobre los pines disponibles y las posibilidades de trabajo de cada uno de ellos.

1.2.3. Patch ***pd OUTPUT_DIGITAL***



Una vez configurados los pines, aquellos que se utilicen como salidas digitales podrán ser gobernados desde **Pure Data** enviando al objeto **arduino** mensajes con la siguiente sintaxis:

digital X Y X es el nombre del PIN e Y es el valor al que queremos llevar su salida (0 o 1)

Para realizar este envío se puede recurrir a un objeto llamado **pd digital**



en el que se encapsula un subpatch al que invocamos para gobernar cualquiera de los pines que estén configurados como salidas (OUTPUT). No olvidemos que por defecto Arduino presenta sus pines digitales como salidas.

También disponemos en este subpatch de otros dos objetos encapsulados muy interesantes:

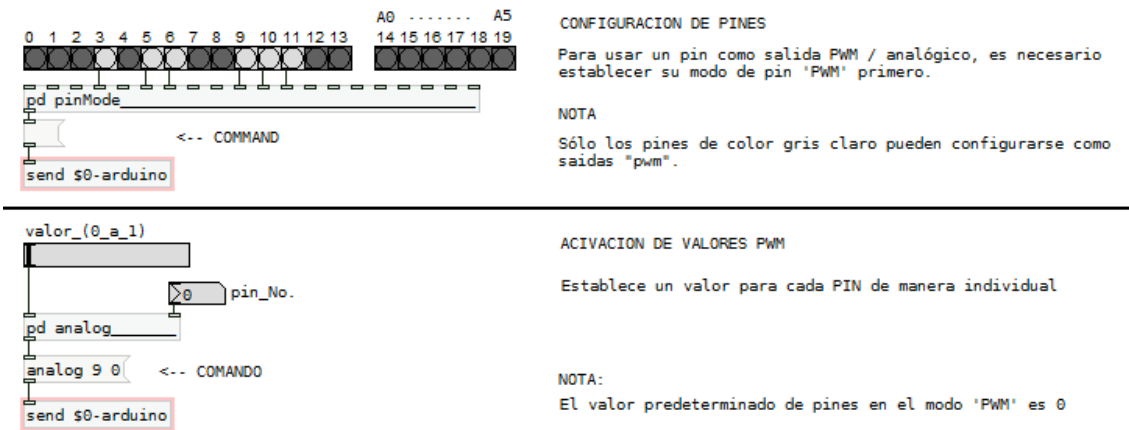
pd pinMode Permite la configuración de los pines

pd port Permite la configuración del Puerto de comunicaciones

No olvidemos que el mensaje se envía al objeto **Arduino** mediante el objeto

send \$0 arduino

1.2.4. Patch *pd OUPUT_PWM*



En este subpatch aparecen dos objetos encapsulados que se encargan de la gestión de las señales de salida PWM que como sabemos se pueden implementar en los pines digitales PIN 3,5,6,9,10,11

El objeto **pd pinMode** ya lo hemos comentado, se utiliza para configurar los pines en modo PWM.

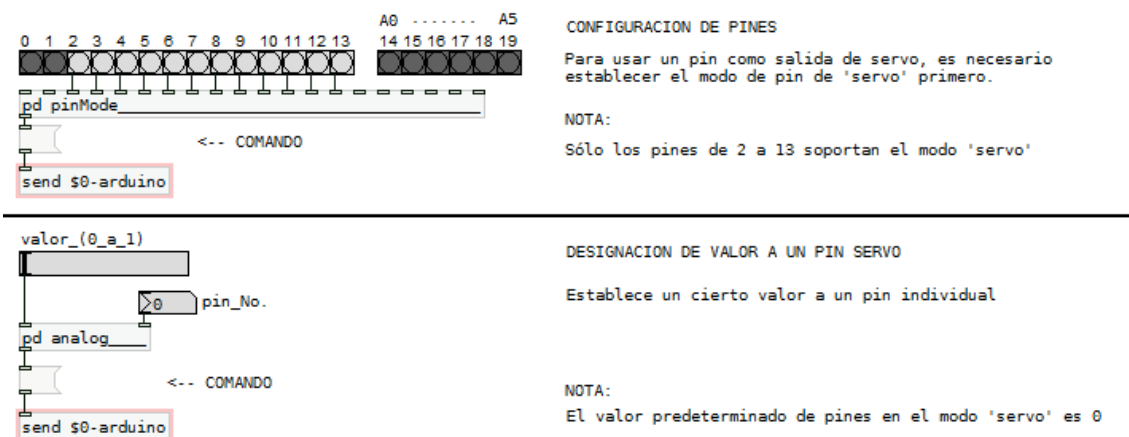
El objeto **pd analog** sirve para enviar al objeto arduino el mensaje que pone el valor analógico en el pin correspondiente (previamente configurado PWM):

analog X Y X representa el número de PIN e Y el valor (entre 0 y 1 que se convierte luego en 0 a 255)

No olvidemos que el mensaje se envía al objeto Arduino mediante el objeto

send \$0 arduino

1.2.5. Patch *pd OUPUT_SERVO*



Para controlar un servo a través de una de las salidas digitales de Arduino se puede configurar cualquier Pin (del 2 al 13) como salida servo.

Para gobernar la salida digital servo se utiliza el objeto, ya explicado anteriormente **pd analog** al que le suministramos los parámetros X (número de PIN) e Y valor de salida (de 0 a 1)

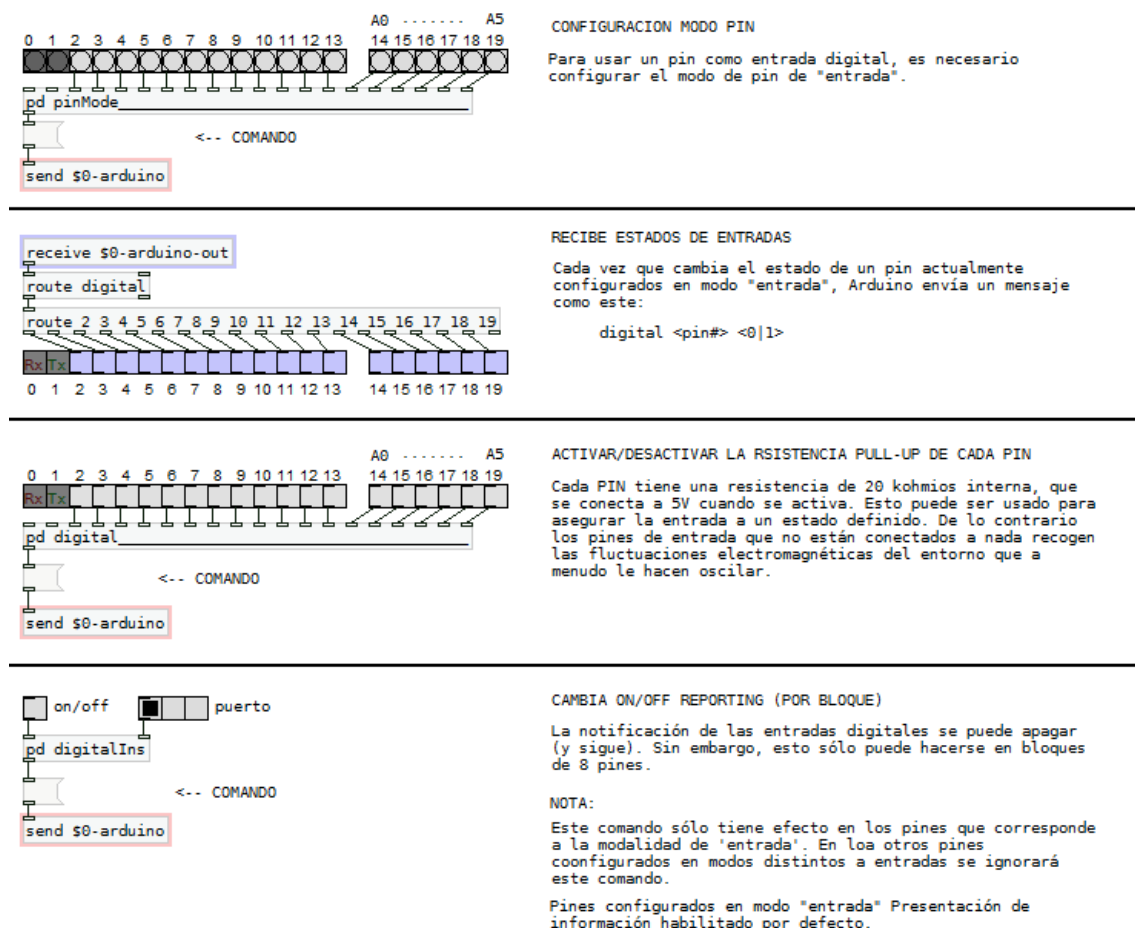
No olvidemos que el mensaje se envía al objeto Arduino mediante el objeto

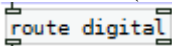
send \$0 arduino 

1.2.6. Patch **pd INPUT_DIGITAL**

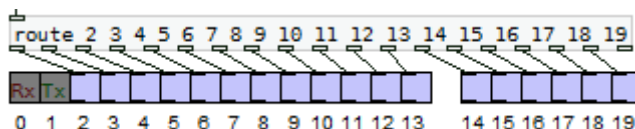
Aquí vemos el procedimiento para leer el estado de las entradas digitales.

Recibimos del objeto Arduino, mediante el bloque “receive...” en forma de paquete (telegrama) el estado de todos los pines digitales.



Para recoger el valor de un PIN de entrada (INPUT) de Arduino recibiremos ese valor mediante el objeto **route digital**  unido al objeto **route 2,3,4...**

En realidad lo que se hace aquí es recoger un “telegrama” del objeto arduino mediante el objeto **receive \$0-arduino-out** que se lleva a los objetos route para extraer cada uno de los valores de los PIN (2 al 19) aunque realmente en a practica solo serán utilizados los PIN 2 a 13



1.2.7. Patch *pd INPUT_ANALOG*

Este es el procedimiento para manejar las entradas analógicas de Arduino

CONFIGURACION MODO PIN

Para usar un pin como una entrada analógica, es necesario establecer el modo de pin de 'analógico'.

NOTA:
Sólo los PINES 14-19 (resp. A0-A5) compatible con el modo 'analógico'.

El modo por defecto de esos contactos es 'analógico', pero con la presentación de informes apagado.

RECIBIR VALORES ANALOGICOS

Un pin configurado en el modo 'analógico' y activado mide el voltaje en el pin y reporta el valor a un ritmo regular con el siguiente mensaje:

analog <pin#> <0...1>

NOTA:
La numeración de los pines es distinta de la numeración para establecer el modo de pin. La numeración utilizada para informar se corresponde con las etiquetas impresas en la tarjeta.

CAMBIAR ON/OFF REPORTING

La lectura de las entradas analógicas se puede activar o desactivar.

NOTA:
Este comando sólo tiene efecto en los pines que corresponde a la modalidad 'analógico'. En los otros grupos de Pins se ignorará este comando.

COMMENT

CONFIGURACION DEL INTERVALO DE MUESTREO DE VALOR

Se puede ajustar la velocidad a la que los pines en modo 'analógico' informan de su valor. Cuanto más corto sea el tiempo de intervalo, mayor será la tasa.

NOTA:
El valor predeterminado es 20 ms. El intervalo de tiempo mínimo es de 10ms (eguals 100Hz).

1.2.8. Path *pd REFERENCE*

En este patch se recoge una lista de los comandos que acepta el objeto **arduino** para las operaciones de entrada y salida de datos así como las solicitudes de información del driver y firmware o el propio manejo de puerto USB.

LISTA DE COMANDOS

<code>open <device#></code>	- conecta a Arduino mediante un numero de puerto
<code>devicename <devicename></code>	- conecata a arduino mediante un nombre de puerto
<code>close</code>	- desconecta Arduino
<code>version</code>	- devuelve la versión de protocolo
<code>firmware</code>	- devuelve la version de Firmware
<code>pinMode <pin#> <mode></code>	- configura el modo de trabajo del PIN
<code>digital <pin#> <0 1></code>	- gobierna el estado de lass salidas digitales
<code>digital <pin#> <0 1></code>	- cambia on/off las resistencias pull-up para cada entrada digital
<code>port <port#> <0...255></code>	- configura el estado de 8 salidas digitales a la vez
<code>analog <pin#> <0...1></code>	- cambia el valor de una salida pwm o servo
<code>digitalIns <port#> <0 1></code>	- cambia on/off reporting de 8 salidas digitales a la vez
<code>analogIns <apin#> <0 1></code>	- cambia on/off reporting de un pin analogico de entrada
<code>samplingInterval <10...16384></code>	- configura el intervalo de escaneo de los valores en los pines analógicos en ms

LISTA DE MENSAJES QUE PODEMOS RECIBIR DE ARDUINO

<code>version <MAJOR> <MINOR></code>	- version del protocolo
<code>firmware <spec></code>	- especificacion completa del firmware
<code>digital <pin#> <0 1></code>	- estado actual de lad entradas digitales
<code>analog <apin#> <0...1></code>	- estado actual de las entradas analógicas

Aquí quedan descritos los comandos que permite el objeto Arduino de Pure Data con los que podemos “conversar” con nuestra tarjeta Arduino.

2. Librería Arduino-Gui

2.1. Patch **arduino-gui**

Pasamos a describir el objeto “**arduino-gui**” que esta basado en el objeto arduino de Pure Data.

He realizado algunas modificaciones y/o ampliaciones sobre el trabajo desarrollado por sus autores con el fin de facilitar, por un lado la identificación gráfica de la tarjeta Arduino y por otro he etiquetado las E/S analógicas/digitales con códigos que permitan el acceso fácil a estos valores. Estas etiquetas de las variables son:

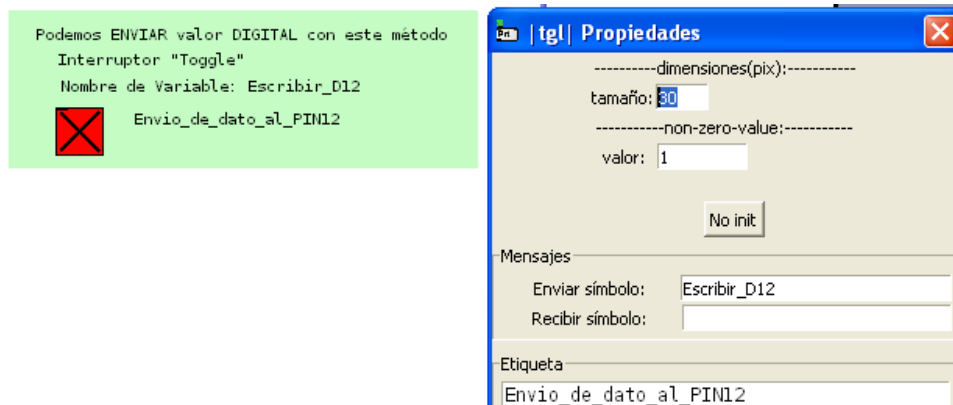
2.1.1. Nombramiento de variables:

Escritura en PIN de un valor digital (salida Digital)

Escribir_Dx En donde **x** representa el número de PIN (2 al 13)

Ejemplo: Para escribir un valor en el PIN 12 deberíamos utilizar la variable **Escribir_D12**

A los efectos de Pure Data se trataría de utilizar un objeto (Button) que en su parámetro “Enviar Símbolo” tuviese indicado el valor de esa variable:

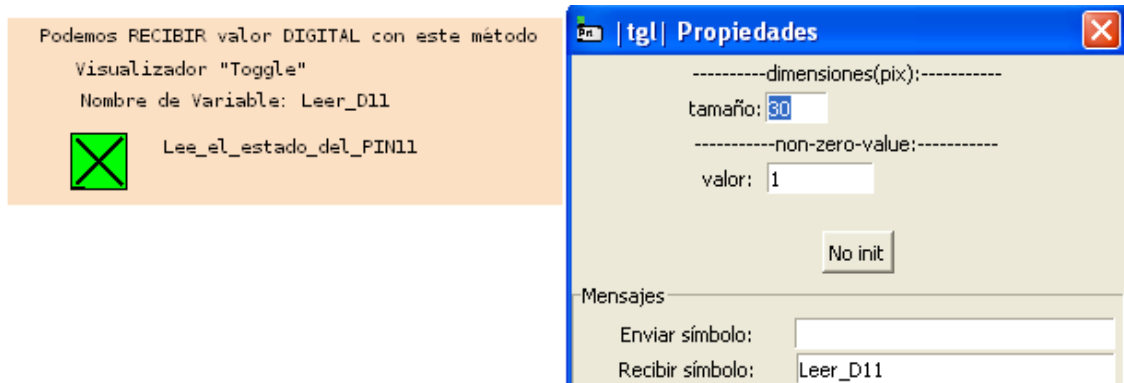


Lectura del valor de un PIN de valor digital (entrada digital)

Leer_Dx En donde **x** representa el número de PIN (2 al 13)

Ejemplo: Para leer un valor recogido del PIN 12 deberíamos utilizar la variable **Leer_D12**

A los efectos de Pure Data se trataría de utilizar un objeto (Button) que en su parámetro “Recibir Símbolo” tuviese indicado el valor de esa variable. En este caso el objeto Button actuaría sencillamente como un receptor de valor.

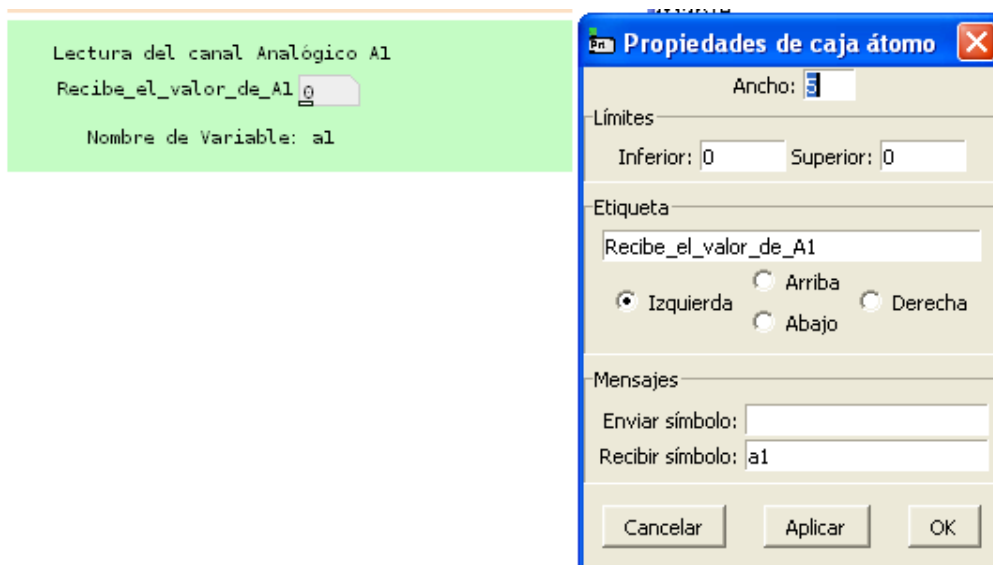


Lectura del valor de un PIN de valor ANALÓGICO (entrada analógica)

ax En donde **x** representa el número de PIN (0 al 5)

Ejemplo: Para leer un valor recogido del PIN 1 deberíamos utilizar la variable **a1**

A los efectos de Pure Data se trataría de utilizar un objeto (Número) que en su parámetro “Recibir Símbolo” tuviese indicado el valor de esa variable.

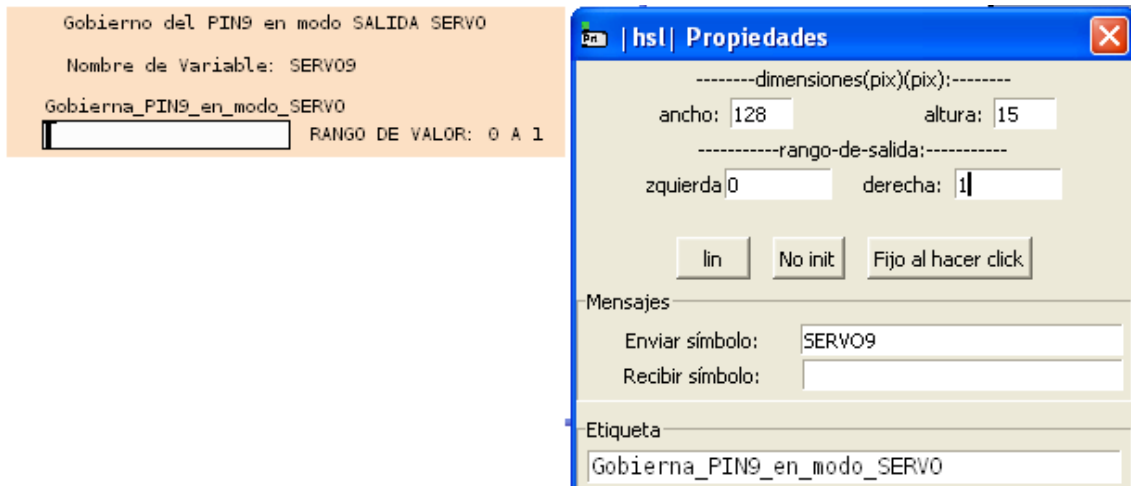


Controlar un PIN como salida SERVO (salida servo)

SERVOx En donde **x** representa el número de PIN (2 al 13)

Ejemplo: Para gobernar el PIN 9 deberíamos utilizar la variable **SERVO9**

A los efectos de Pure Data se trataría de utilizar un objeto (hslider) que en su parámetro “Enviar Símbolo” tuviese indicado el valor de esa variable. Su rango debe ser de 0 a 1

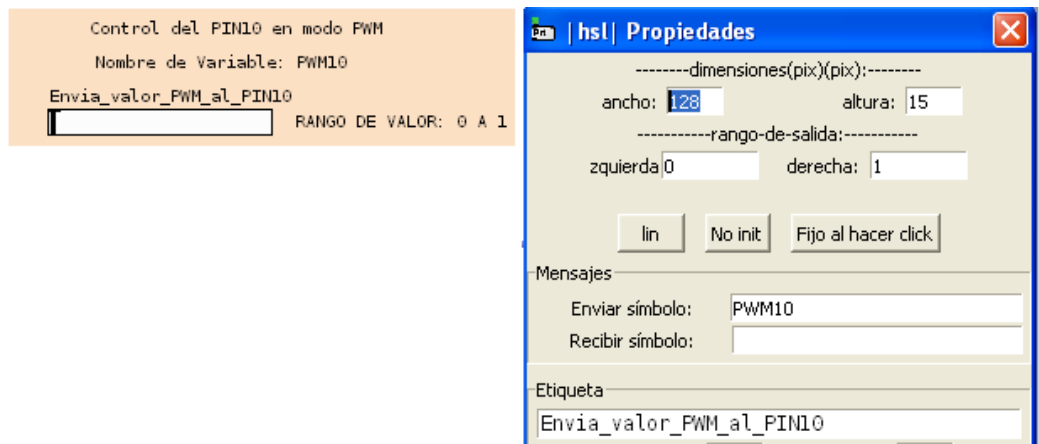


Controlar un PIN como salida PWM (salida analógica)

PWM_x En donde **x** representa el número de PIN (3,5,6,9,10,11)

Ejemplo: Para gobernar el PIN 12 deberíamos utilizar la variable **PWM12**

A los efectos de Pure Data se trataría de utilizar un objeto (hslider) que en su parámetro “Enviar Símbolo” tuviese indicado el valor de esa variable. Su rango debe ser de 0 a 1

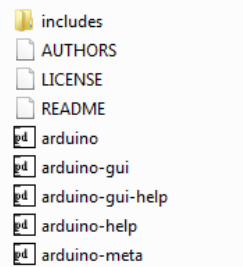


Este patch es el que insertaremos en nuestros ejercicios con Pure Data y Arduino, en realidad es el match principal de esta librería Arduino-Gui.

Consta de una imagen de Arduino que nos permite ubicar cada uno de los pines de la tarjeta. Se han colocado sobre esta imagen una serie de controles (tipo menú desplegable) que nos permitirán seleccionar el modo de funcionamiento de cada PIN y también unos cuadros de selección que permiten habilitar entradas analógicas de la tarjeta.

Cuando nosotros colocamos el objeto **arduino-gui** sobre un nuevo match quedaría de la siguiente manera:

No debemos olvidar nunca que en la carpeta en la que estamos trabajando con Pure Data deben estar los ficheros que hemos comentado al principio, de lo contrario no se encontrara el patch **arduino-gui**. Podemos resolverlo indicándole el patch en el que se encuentran estos fichero pero es recomendable trabaja en el la misma carpeta.



En el ejemplo *ag0-valores.pd* encontramos los procedimientos para el tratamiento de valores (lectura y escritura)

LECTURA Y ESCRITURA DE VALORES MEDIANTE SUS ETIQUETAS

Podemos ENVIAR valor DIGITAL con este método
Interruptor "Toggle"
Nombre de Variable: Escribir_D12

Envio_de_dato_al_PIN12

Podemos RECIBIR valor DIGITAL con este método
Visualizador "Toggle"
Nombre de Variable: Leer_D11

Lee_el_estado_del_PIN11

Lectura del canal Analógico A1
Recibe_el_valor_de_A1 0.299

Nombre de Variable: a1

Gobierno del PIN9 en modo SALIDA SERVO
Nombre de Variable: SERV09

Gobierna_PIN9_en_modos_SERVO

RANGO DE VALOR: 0 A 1

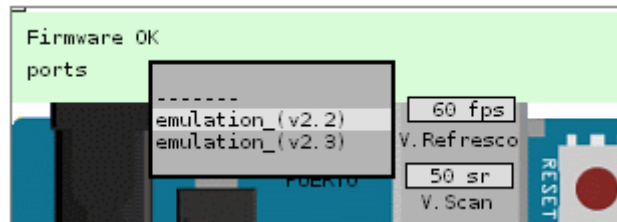
Control del PIN10 en modo PWM
Nombre de Variable: PWM10

Envia_valor_PWM_al_PIN10

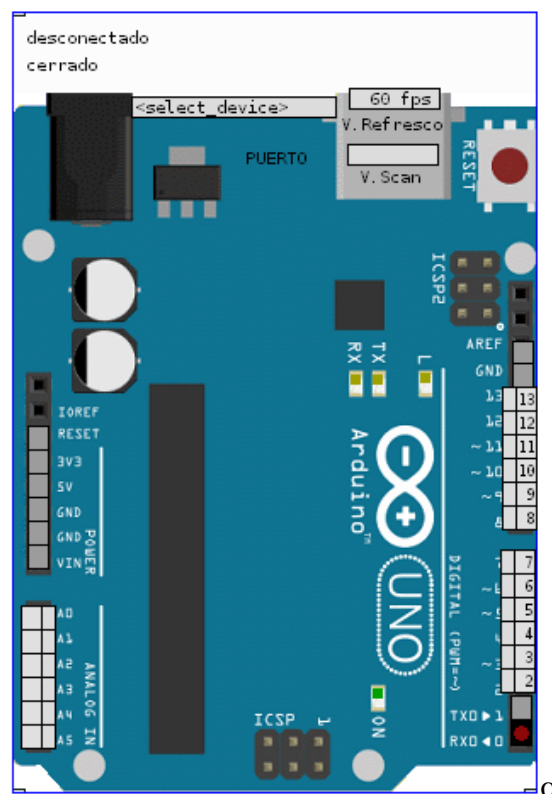
RANGO DE VALOR: 0 A 1

Para establecer la comunicación con Arduino debemos seleccionar mediante el ratón sobre la ventana *<select device>* en donde aparecerán los puertos por los que podemos comunicarnos con Arduino.

Arduino-Gui puede funcionar en modo directo: conectado con arduino a través del puerto USB o en modo emulación..



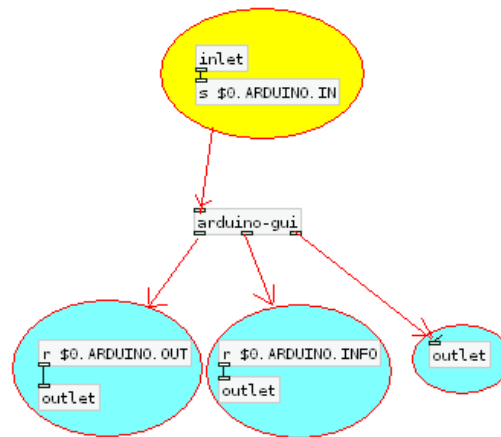
Para comenzar a realizar una aplicación lo que debemos hacer es cargar el objeto que contiene la librería **arduino-gui**, es lo hacemos simplemente con **Poner** → **Objeto** del menú de Pure Data, y escribiendo dentro de la caja que aparece justamente el nombre “**arduino-gui**”. A continuación aparecerá el objeto tal como se ve en la figura siguiente.



Lo que hacemos a continuación es colocar los distintos objetos con los que vamos a trabajar, realizando el enlazado ente ellos y poniendo los parámetros que correspondan a cada uno (etiquetas, valores, tamaños colores, etc..)

El objeto que colocamos en nuestra área de trabajo posee también un terminal tipo **inlet** (entrada) y tres terminales **outlets** (salida).

En la figura siguiente vemos un detalle de estos terminales



A través del **inlet s \$0 ARDUINO.IN** podemos enviar mensajes al bloque que no son otra cosa que comandos para configuración de las E/S, gobierno de salidas digitales y analógicas, petición de información sobre firmware, puertos, etc..

Los outlets del bloque nos envían datos provenientes de Arduino

`r $0. ARDUINO.OUT`

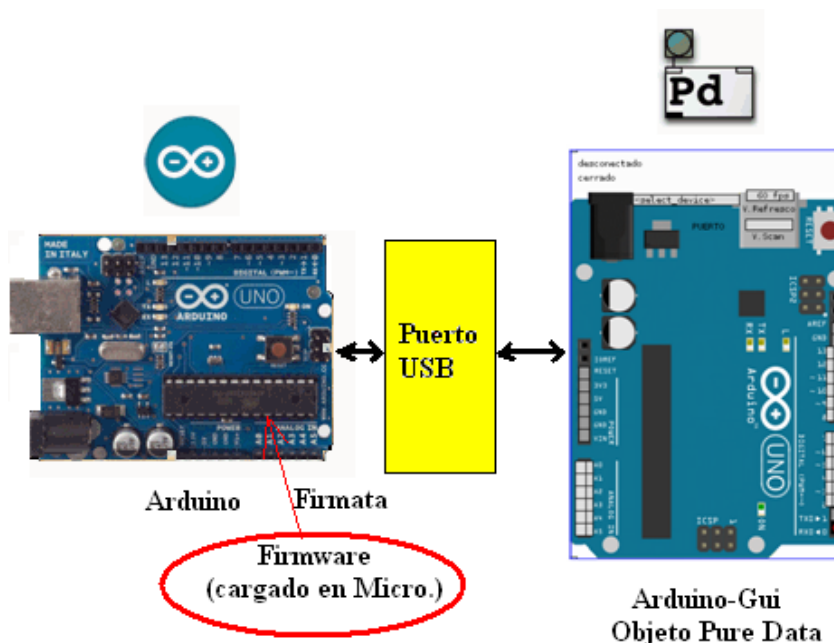
Se recogen datos de los pines digitales y analógicos

`r $0. ARDUINO.INFO`

Se recoge información sobre firmware, puertos, etc..

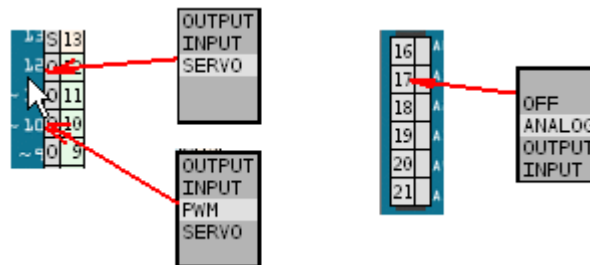
`outlet`

Se recoge información sobre los comandos que se envían



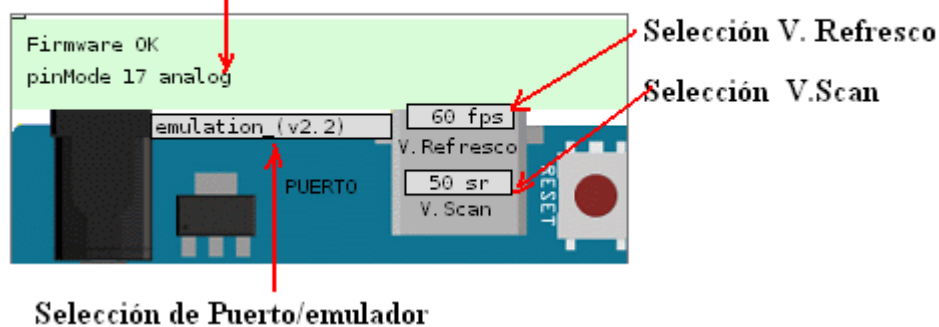
Este esquema nos muestra la arquitectura de nuestro sistema. En la tarjeta Arduino debemos tener guardado el firmware Firmata que se encargara de la comunicación a través del puerto USB con el objeto Arduino-Gui de Pure Data.

Para seleccionar la forma de trabajo de los pines bastará con pulsar sobre el pequeño cuadrado de pin y del menú que aparezca se podrá seleccionar la opción correspondiente al tipo de dato que vamos a designar en el:



En la figura siguiente se encuentran los lugares en donde modificar o visualizar la información propia de la comunicación con Arduino a través del USB

Indicación de Estado Instrucción recibida



2.2. Patch **arduino-gui-help**

ARDUINO - GUI

[arduino-gui] es una versión gráfica del común [arduino]. Se trata de un shell con algunas mejoras GUI que faciliten el envío de comandos a la placa Arduino. Estos comandos también se pueden grabar en un cuadro de mensaje.

Modulos soportados:

- * UNO
- * Duemilanove
- * Diecimila
- * NG

Firmware Soportado:

- * StandardFirmata >= v2.2

Requiere:

- * Pd[-extended] >= 0.43

<- selecciona un puerto
<- configura velocidad de refresco
<- configura refresco de entradas analógicas
<- configurar el modo de trabajo de los PIN

EMULATION MODE

Seleccione 'emulación' en el selector de dispositivo para emular una placa Arduino sin tener que enviar los comandos a un dispositivo físico.

Para ver todas las referencias `arduino`

<- recoge los comandos arduino en una caja de texto

print ARDUINO.OUT
print ARDUINO.INFO
digital 13 0

Copyright, 2012, Roman Haefeli
Olsen Wolf
Traducido y adaptado por J.M. Ruiz 2013

En este patch se recoge información de ayuda que explica el funcionamiento del objeto **arduino-gui**.

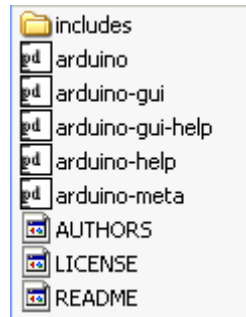
El objeto **arduino-gui** esta basado en el objeto **arduino** creado por (C) Copyright 2006-2012 Free Software Foundation. A su vez esta versión que presento fue realizada por Copyright, 2012, Roman Haefeli y Olsen Wolf

3. Empezamos.

Para trabajar necesitaremos tener instalado el software:

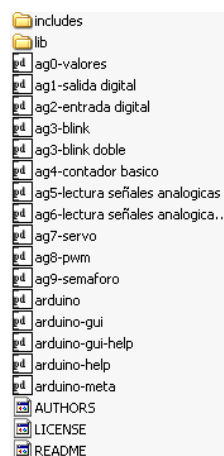
- **IDE Arduino 1.0.5** . <http://arduino.cc/en/Main/Software>
- **Pure Data Pd-Extended** <http://puredata.info/downloads>

Una carpeta de trabajo que contendrá una estructura de ficheros del tipo:



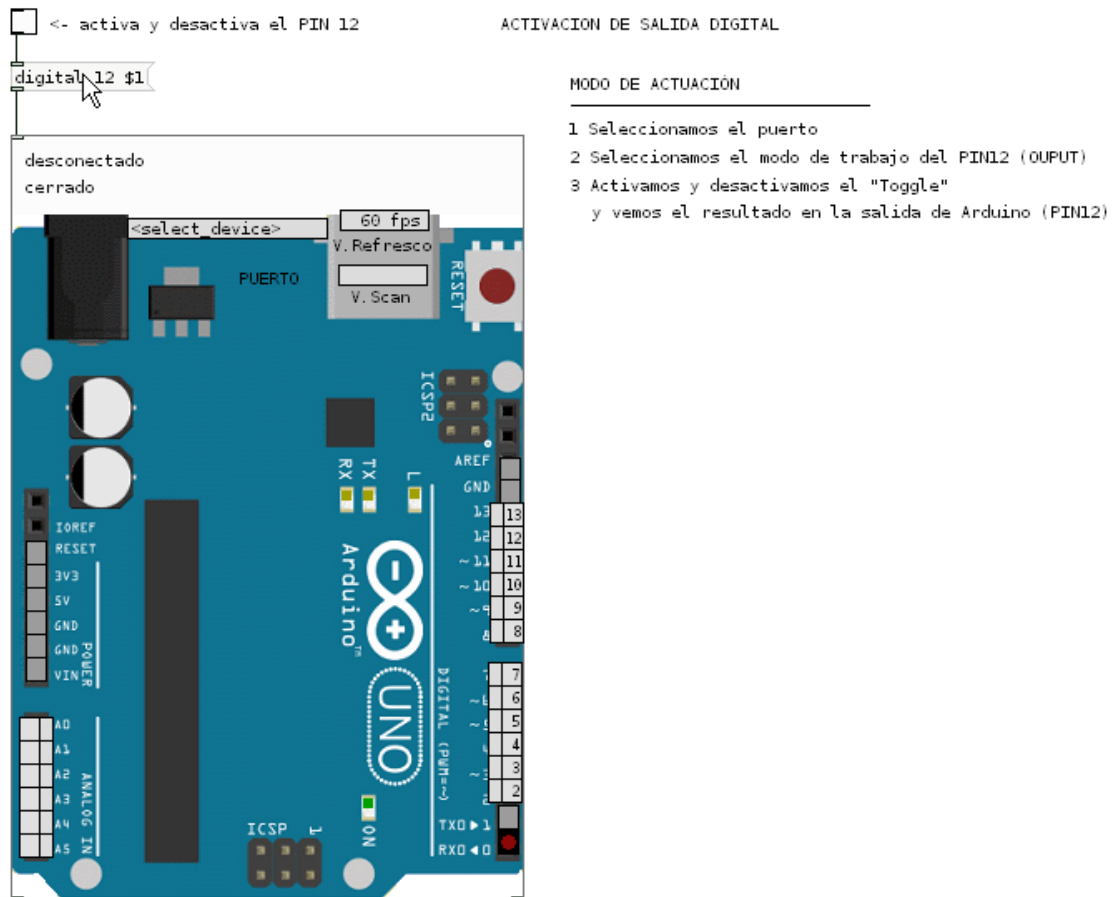
Los ficheros que se encuentran en el raíz de la carpeta, para su ejecución necesitan tener la carpeta **includes** en a que figuran subpatch, imágenes y otros ficheros necesarios. Los trabajos que realicemos los tenemos que poner en el raíz de esta carpeta de trabajo con el fin d que cuando sea invocada la librería **arduino-gui.pd** esta se encuentre en su lugar. Podríamos colocarla en otra carpeta distinta pero al invocarla desde Pure Data tendríamos que indicar el path en e que se encuentra.

Este manual va acompañado de un fichero comprimido que al descomprimirse lo hace en una carpeta que se llama **arduino-gui** en el que figura esta estructura comentada además de los ficheros que pongo como ejemplo en este manual y una carpeta llamada **lib** en la que se encuentran una serie de librerías que pueden ser usadas en las prácticas que realice usted.



No debemos olvidarnos antes de nada de cargar en la tarjeta el firmware **Firmata** haciendo uso del **IDE Arduino**. Si lo desea este firmware figura en los ejemplos que trae el **IDE Arduino Ejemplos-> Firmata-> StandardFirmata**

3.1. Escribir en una salida digital



En este primer ejemplo se trata de gobernar una salida digital que estableceremos en el PIN12. Para ello haremos uso de dos únicos objetos Pure Data: Objeto **Toggle** y objeto **Mensaje**

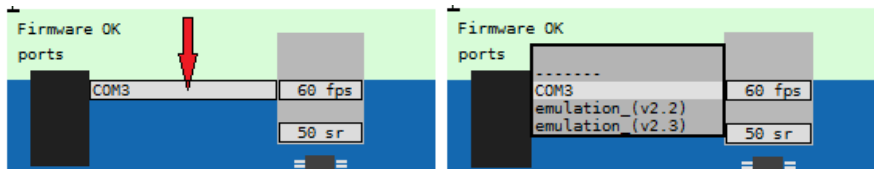
Toggle vine a ser un interruptor que al pulsar sobre el envía un “1” lógico -> activa la salida **PIN12**

En el objeto **Mensaje** escribimos *digital 12 \$1* lo cual significa que cuando por su entrada llegue el valor “1” enviará al bloque **arduino-gui** la orden de activar la salida **PIN12 \$1** es el nombre de una variable que es la que toma e valor que entra proveniente del **Toggle**

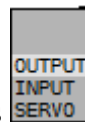
Una vez realizado el montaje y unidos los bloques pasamos al modo de ejecución CTRL+E o deseleccionando en el menú Editar-> Modo Edición.

A continuación nos colocamos sobre el espacio *<select device>* y manteniendo pulsada la tecla izquierda del ratón seleccionamos el COM por el que nos comunicaremos con Arduino.

Seleccionamos el puerto.




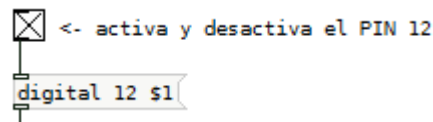
Después debemos seleccionar el modo de trabajo del PIN12 que es con el que vamos a actuar. En este caso no haría falta porque por defecto los pines digitales están configurados como salida (OUTPUT) pero se seleccionaría pulsando con el ratón sobre el pequeño cuadrado de la derecha junto al número de pin y allí se despliega un menú de tipos de configuración y seleccionamos OUTPUT



Selección de modo de trabajo de los pines digitales

Actuamos sobre el interruptor “**Toggle**” y observamos que la salida **PIN12** se pone

verde  si miramos en la entrada del objeto **arduino-gui** vemos el código que le

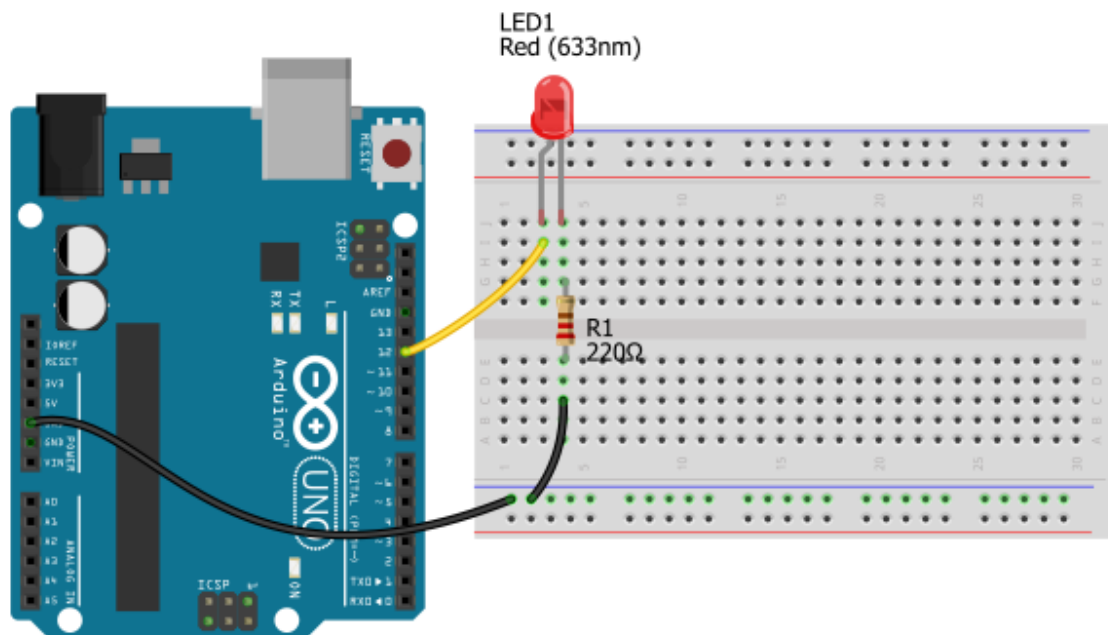


estamos enviando

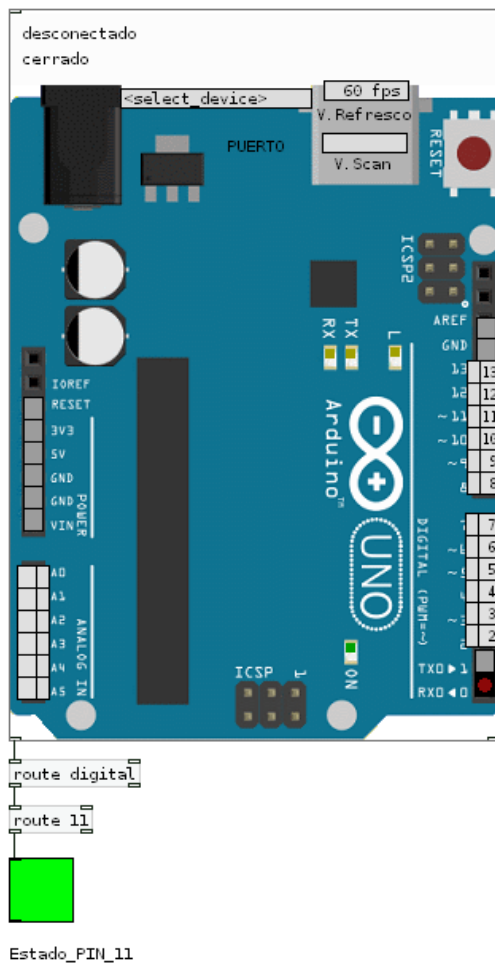
que será el mismo que veamos

en el “**outlet**” de salida del en donde mediante el objeto de mensaje vemos todo lo que recibe Arduino. El mensaje se envía mediante el bloque **Mensaje** de Pure data.

Esquema de montaje:



3.2. Lectura de una entrada digital desde Arduino a Pure data



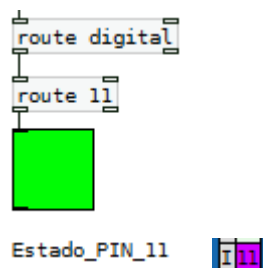
LECTURA DE UNA ENTRADA DIGITAL

MODO DE ACTUACIÓN

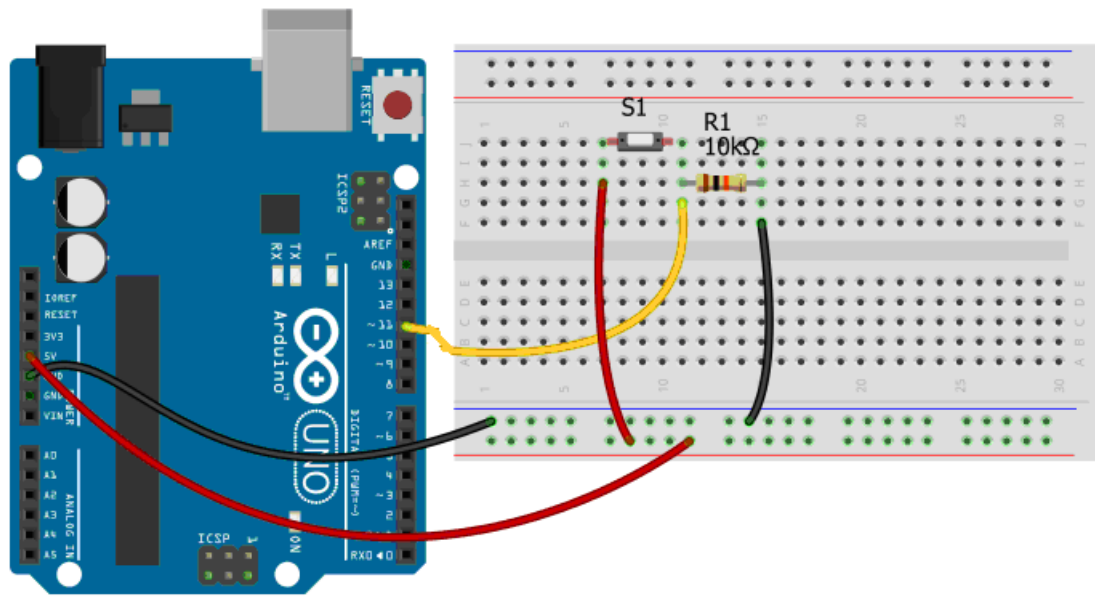
- 1 seleccionamos el puerto
- 2 Seleccionamos el modo de trabajo del PIN11
- 3 Activamos y desactivamos el pulsador de la entrada PIN11 y vemos el resultado en el indicador verde (Toggle)

Con este montaje probaremos como leer una variable de entrada de tipo digital obtenida del PIN11 configurado como entrada (INPUT).

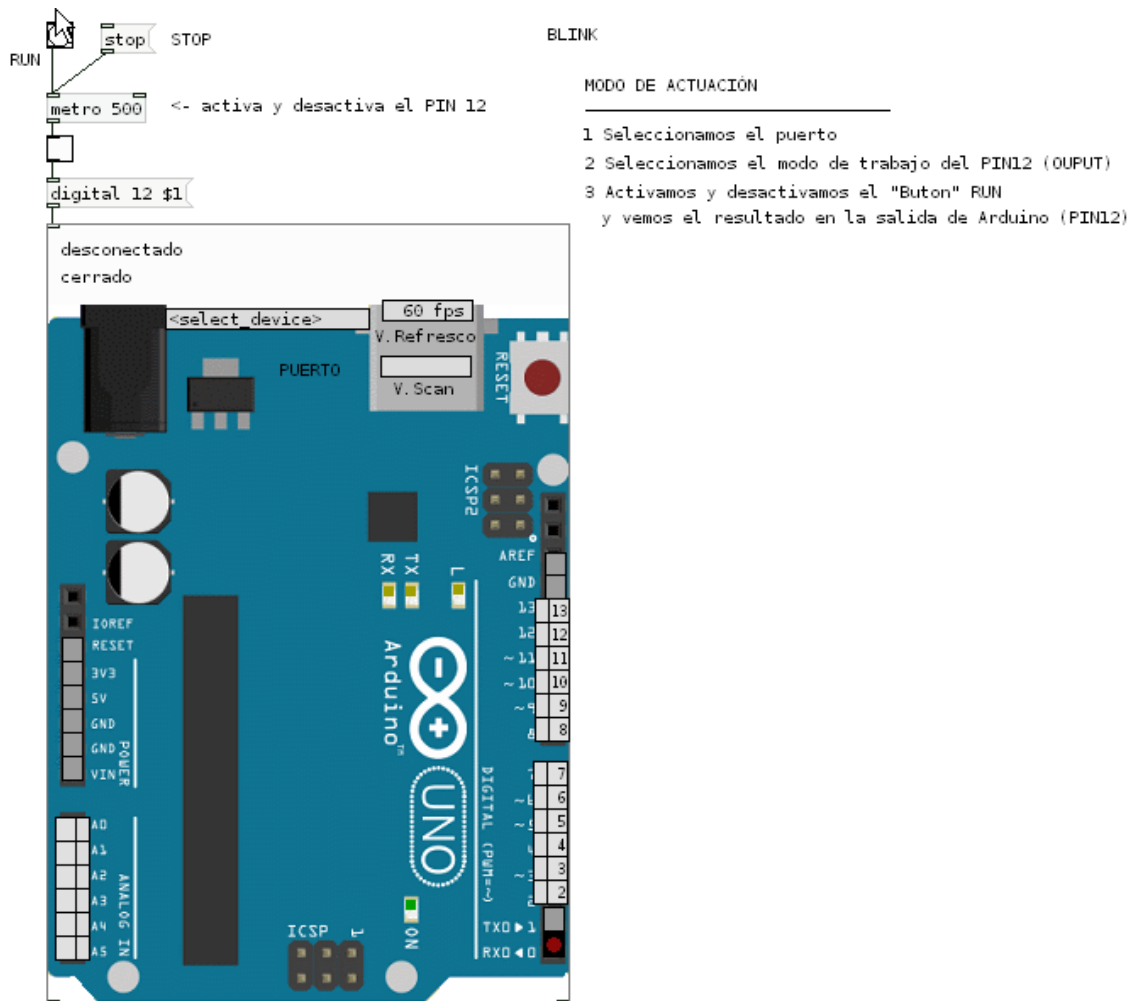
El valor lo obtenemos a través del **oulet** del objeto **arduino-gui** mediante el bloque **route digital** que lee el estado de los pines digitales para que luego el bloque **route 11** extraiga el estado de la variable del **PIN11** que se muestra con un objeto **Toggle** al que le ponemos el color verde y aumentamos de tamaño.



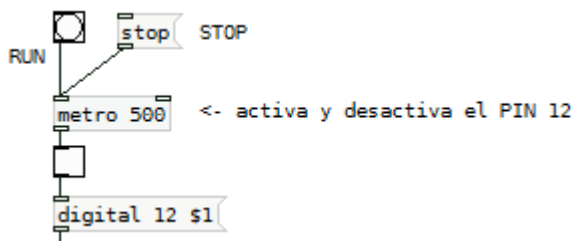
Esquema de montaje.

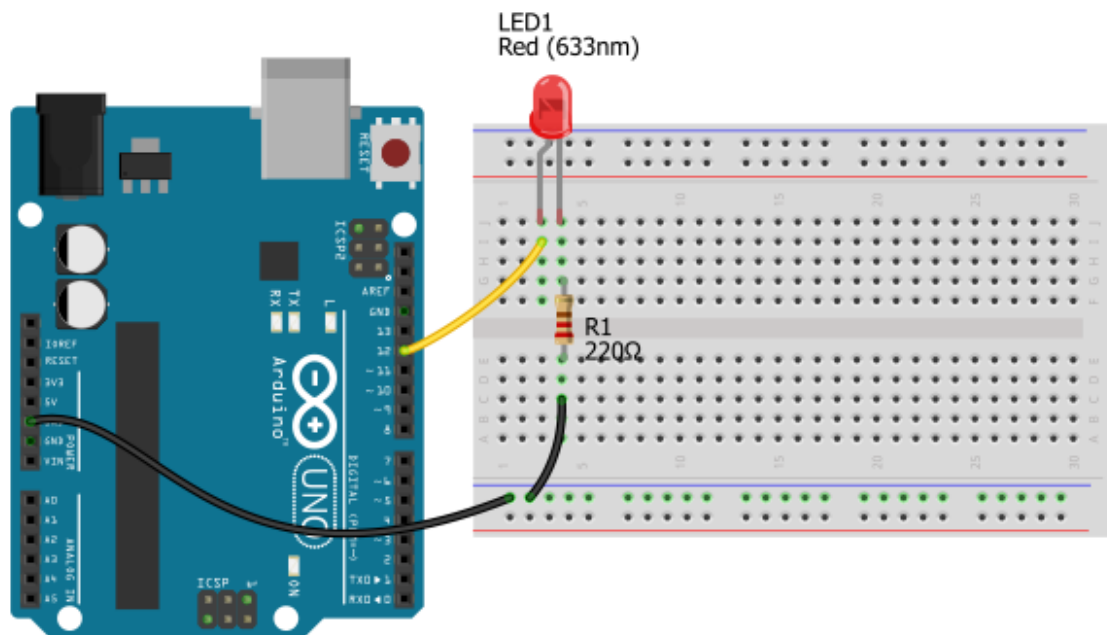


3.3. Blink

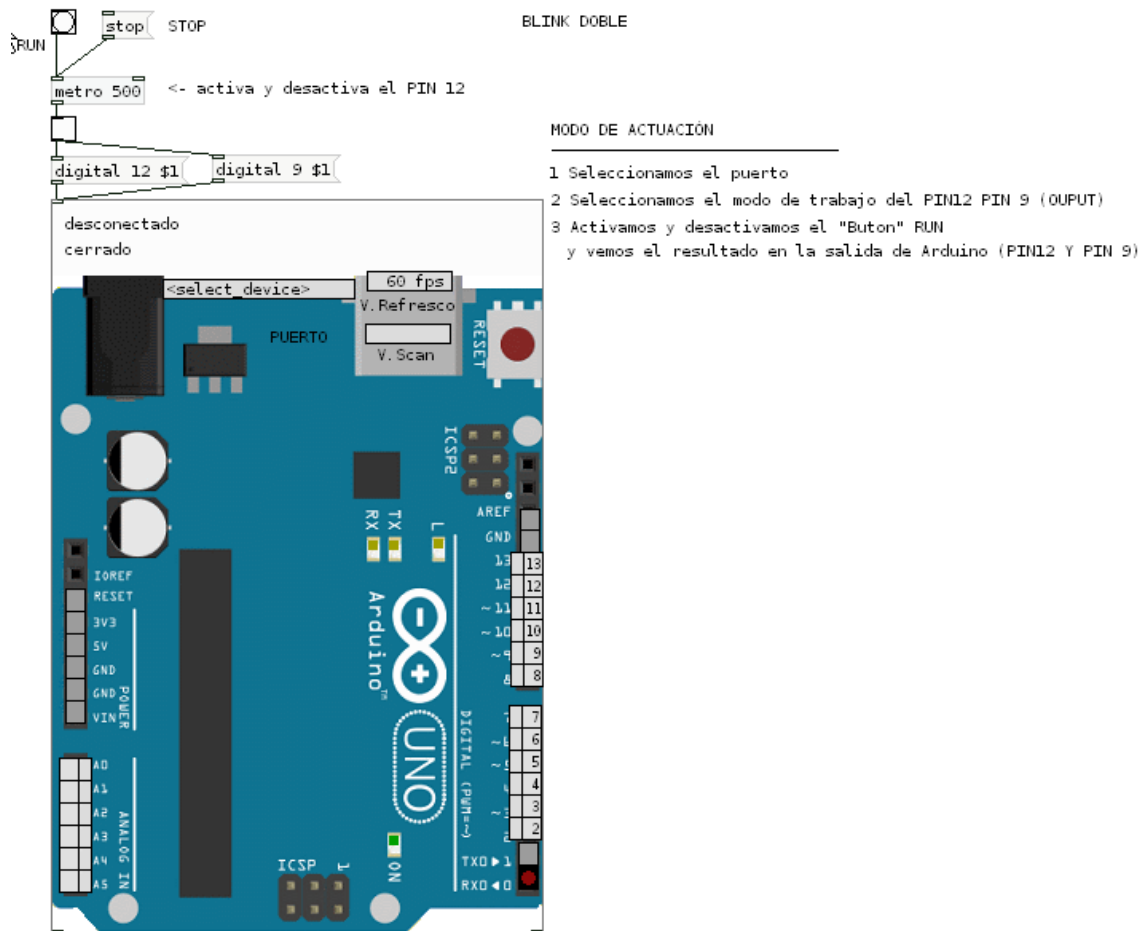


En este montaje hacemos uso del bloque **metro** que genera impulsos con la duración que le ponemos como parámetro (es posible darle un valor modificable también). Enviando el mensaje **stop** detenemos el envío de señal. El mensaje enviado es **digital 12 \$1** en donde la variable \$ es la que recogemos del bloque **metro** y la pasamos también por un bloque **Toggle** para tener información de su estado simplemente.

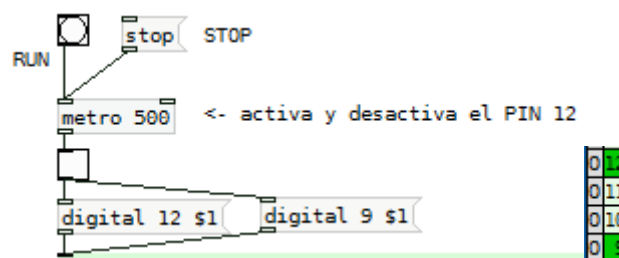


Esquema de montaje:

3.4. Blink Doble

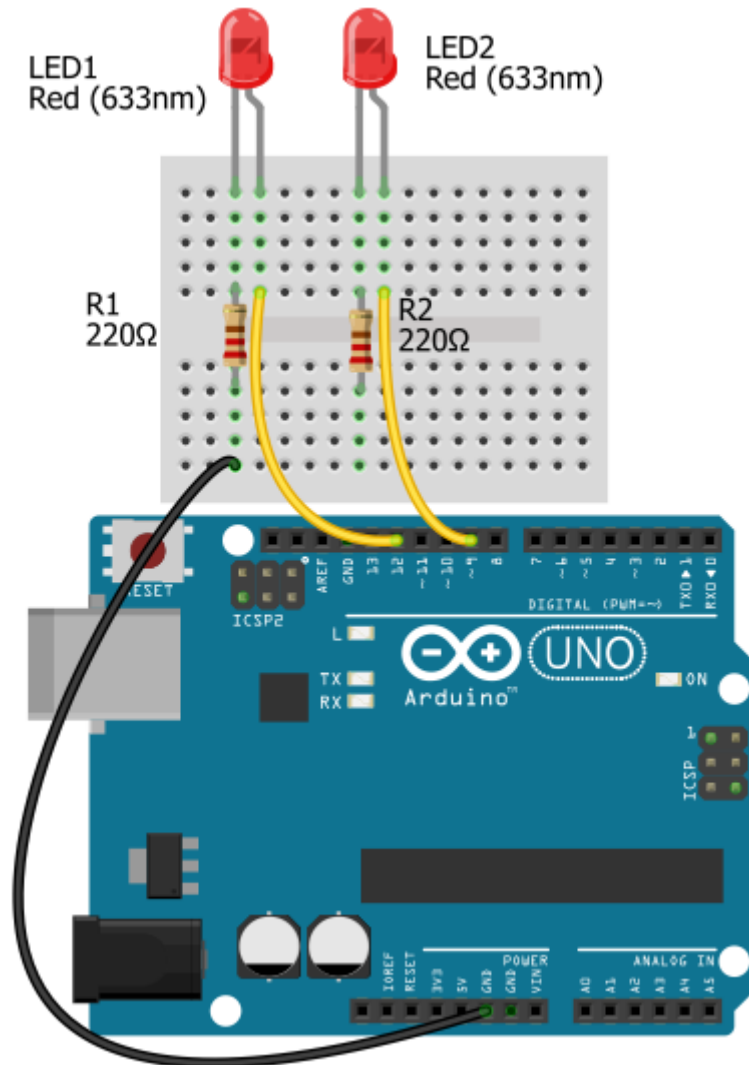


Este es una variación del montaje anterior en e que actuamos sobre dos salidas digitales PIN12 y PIN9. Vemos como en los indicadores de estado de la variable PIN se ponen de color verde cuando se activa la salida.

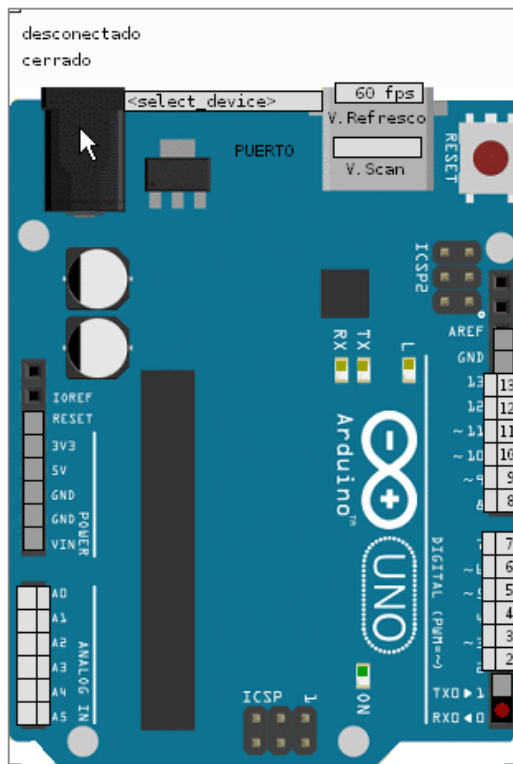


Se lleva la señal del bloque **metro** a dos objetos **Mensaje** que envían cada uno a un pin los valores de activación y desactivación de las salidas.

Esquema de montaje



3.5. Contador Básico



CONTADOR BASICO

MODO DE ACTUACIÓN

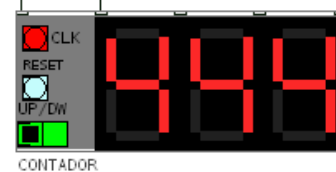
- 1 seleccionamos el puerto
- 2 Seleccionamos el modo de trabajo de las entradas PIN11 y PIN10
- 3 Activamos y desactivamos los pulsadores de cada entrada y comprobamos el funcionamiento

Imp. Entrada

☐ Leer_D11 <-- recoge estado de entrada PIN11

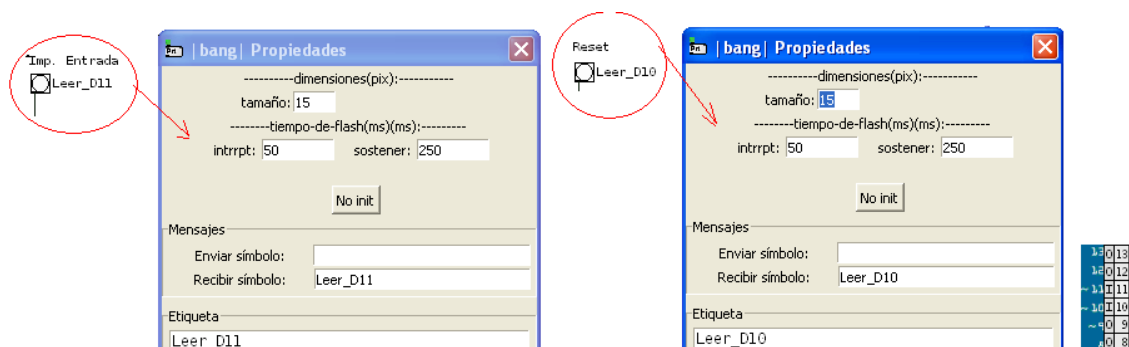
Reset

☐ Leer_D10 <-- recoge estado de entrada PIN10



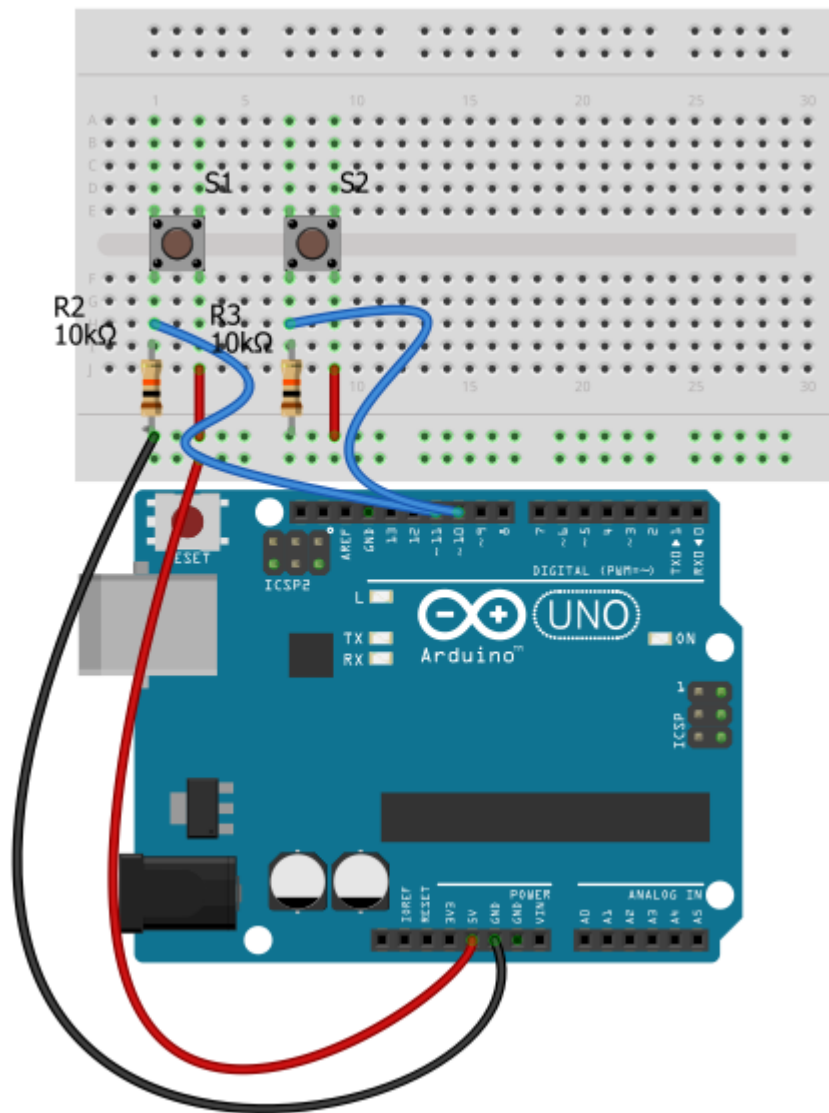
En este ejemplo utilizamos un bloque de la librería que se encuentra en la carpeta **lib** y que se denomina **contador**. Se trata de recoger los impulsos de entrada que proceden de un pulsador conectado en el **PIN 11** y contarlos. Esta vez en lugar de recoger el valor de la variable del **outlet** de salida del bloque **arduino_gui** lo que hacemos es simplemente recoger el valor de la variable **Leer_PN11** que como ya hemos explicado anteriormente contiene el valor del estado del **PIN 11**, esta vez configurado como entrada digital.

Por la entrada **PIN10** leeremos el valor digital que pondrá a “cero” RESET el contador, igualmente esta entrada se recoge de forma diferida simplemente configurando el bloque **Bang** como recetor de señal.

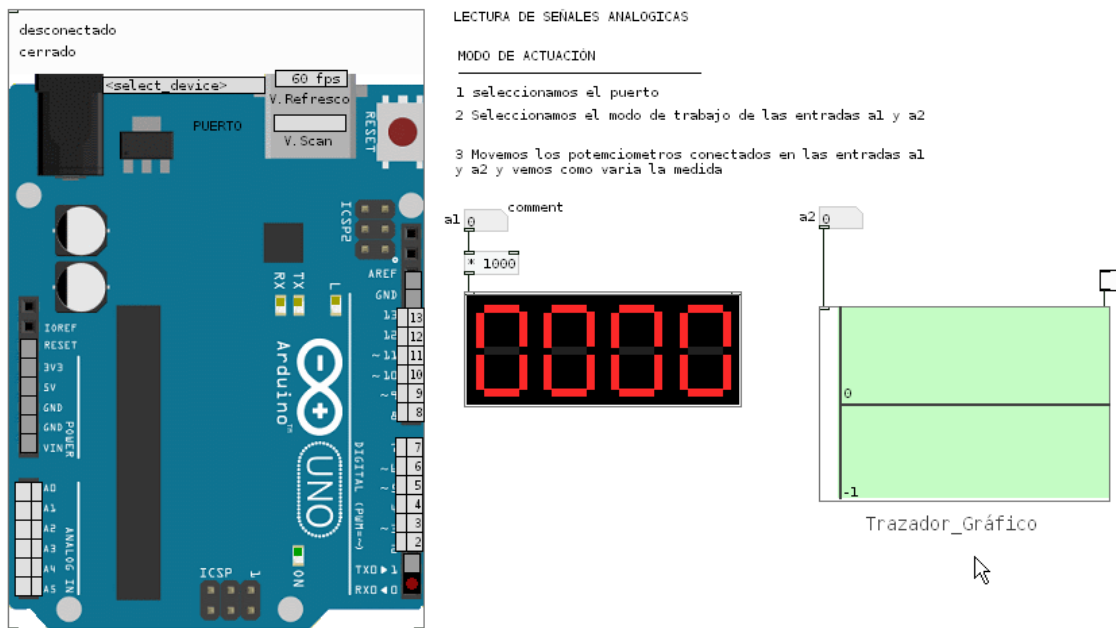


No debemos olvidar configurar los pines **PIN11** y **PIN 10** como entradas.

Esquema de montaje



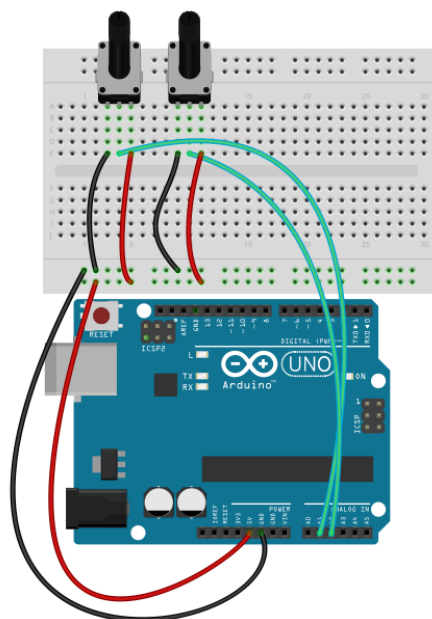
3.6. Lectura de señales analógicas.



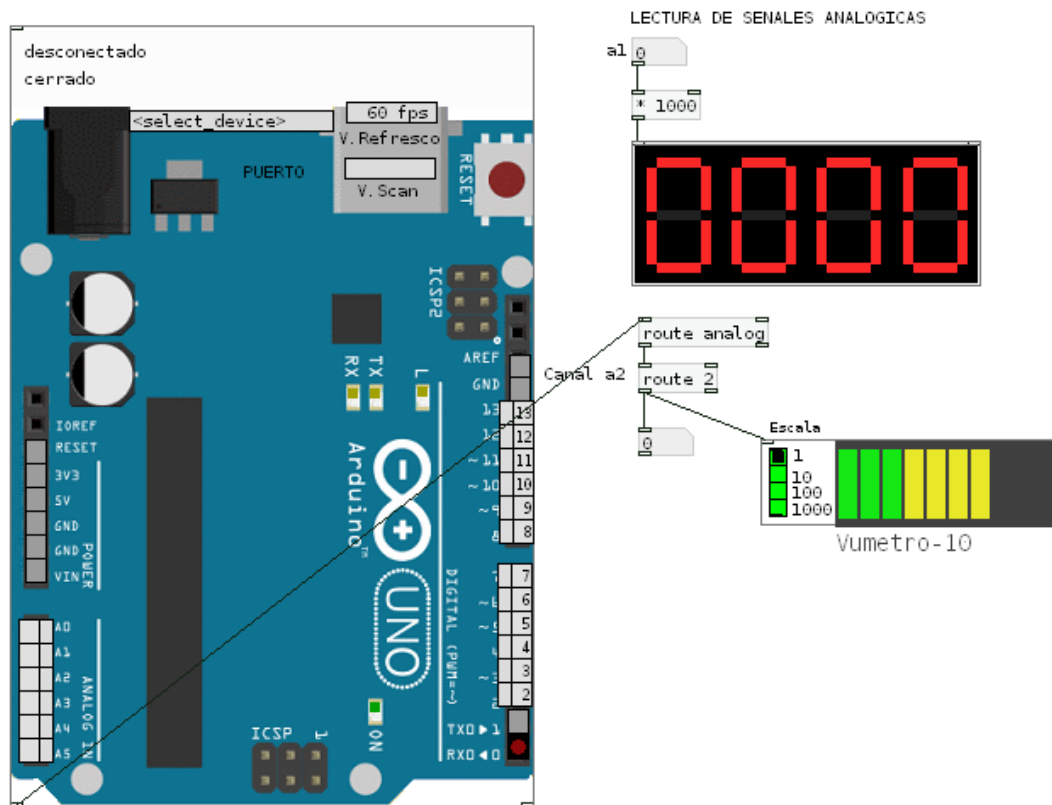
Con este ejemplo leeremos los valores de entrada de dos canales analógicos de Arduino (A1 y A2) entregados por el objeto **arduino_gui** en los valores de variable **a1** y **a2**. Se recogen estos valores y se llevan por un lado **a1** a un display tomado de la librería y por otro **a2** a un trazador grafico en el que se visualizara la señal leída.

En el caso de la señal a1 lo que hacemos es multiplicarla por 1000 ya que el objeto nos la entrega con un rango de 0 a 1.

Esquema de montaje.



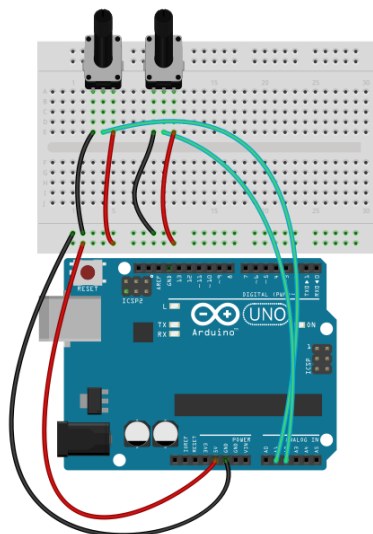
3.7. Lectura de señales analógicas 2



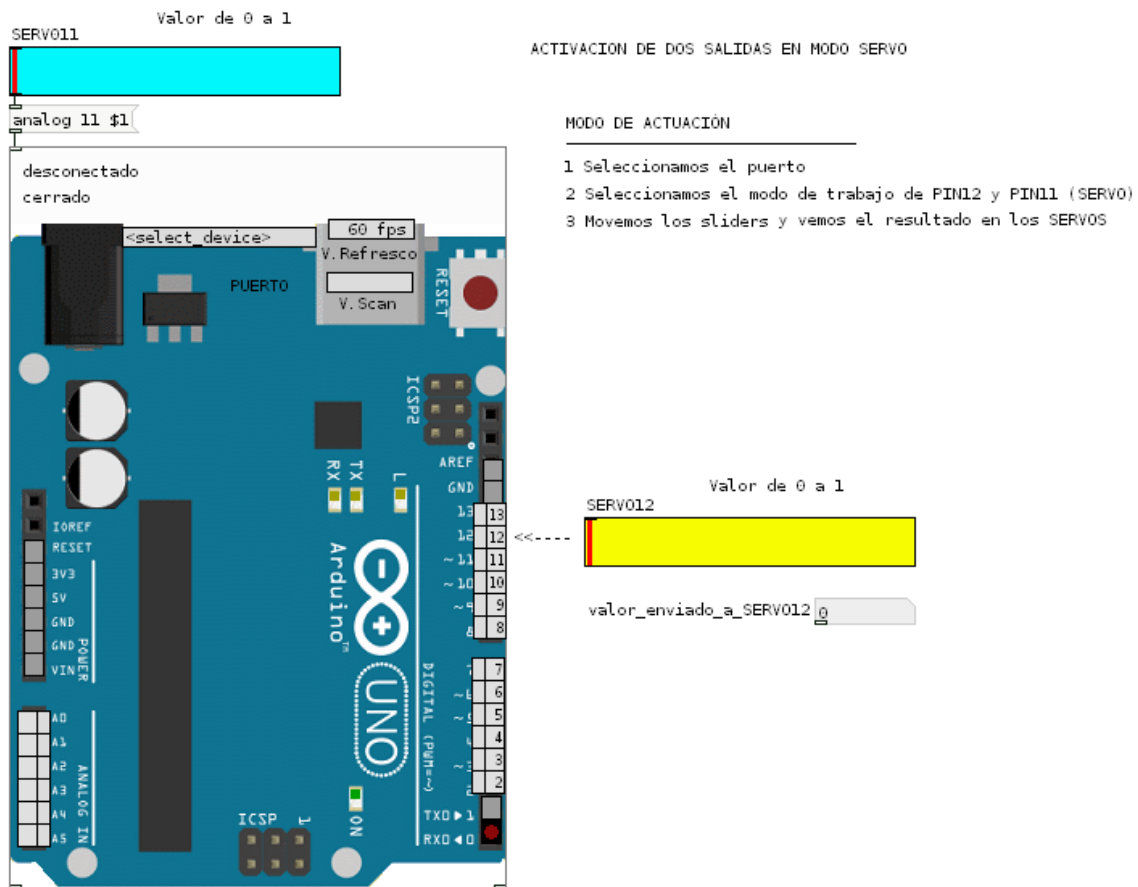
En este ejemplo se leen igualmente dos variables analógicas a1 y a2, la primera la sacamos directamente desde el **outlet** del objeto **arduino_gui** y en el caso del canal a2 lo hacemos a través del nombre de la variable invocado desde el objeto Numero en el que se ha puesto el parámetro: **Recibir Símbolo: a1**.

Se ha introducido el objeto de librería **vumetro10** que muestra a través de una escala de 10 barras leds un valor que puede ser escalado en 1,10,100 y 1000

Esquema de montaje.



3.8. Activación de salidas en modo SERVO

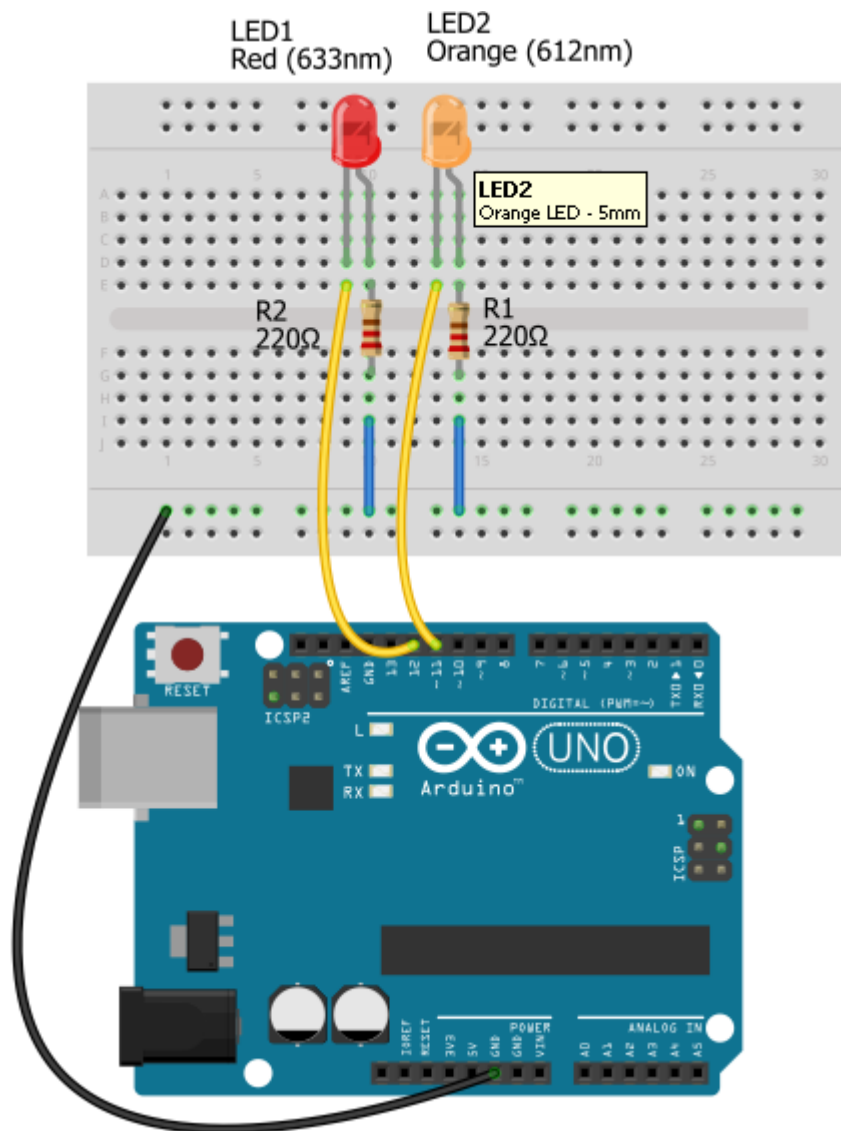


Este montaje se realiza para estudiar el comportamiento de las salidas SERVO. Para ello basta con definir en este caso los pines **PIN12** y **PIN11** como salidas SERVO y en el caso del gobierno del **PIN 11** inyectamos el mensaje correspondiente en el **inlet** del objeto **arduino_gui** mediante un objeto Mensaje en el que escribimos **analog 11 "\$1** para el **PIN 12** lo hacemos de manera indirecta poniendo el nombre de la variable en la configuración del **slider Enviar Símbolo: SERVO11**.

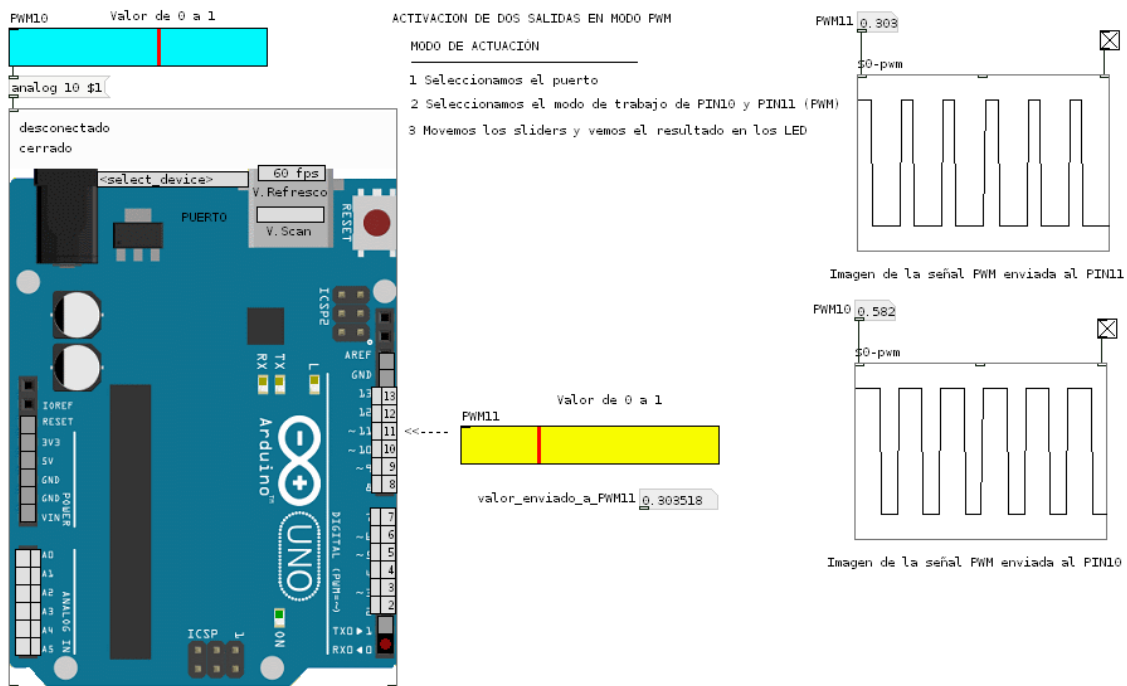
También hemos colocado un bloque **Número** para leer el valor que mandamos con el slider igualmente, en este caso, configuramos en el bloque Número con **Recibir Símbolo: SERVO11**.

Se han colocado dos objetos **pwm** de la librería que se encuentra en la carpeta **lib** y que muestran de manera gráfica el tipo de señal que estamos mandando por cada una de las salidas **PWM**.

Los objetos **Hslider** que se utilizan para generar el valor de a enviar a las salidas deben configurarse para que estos valores sean entre 0 y 1.

Esquema de montaje:

3.9. Salidas PWM

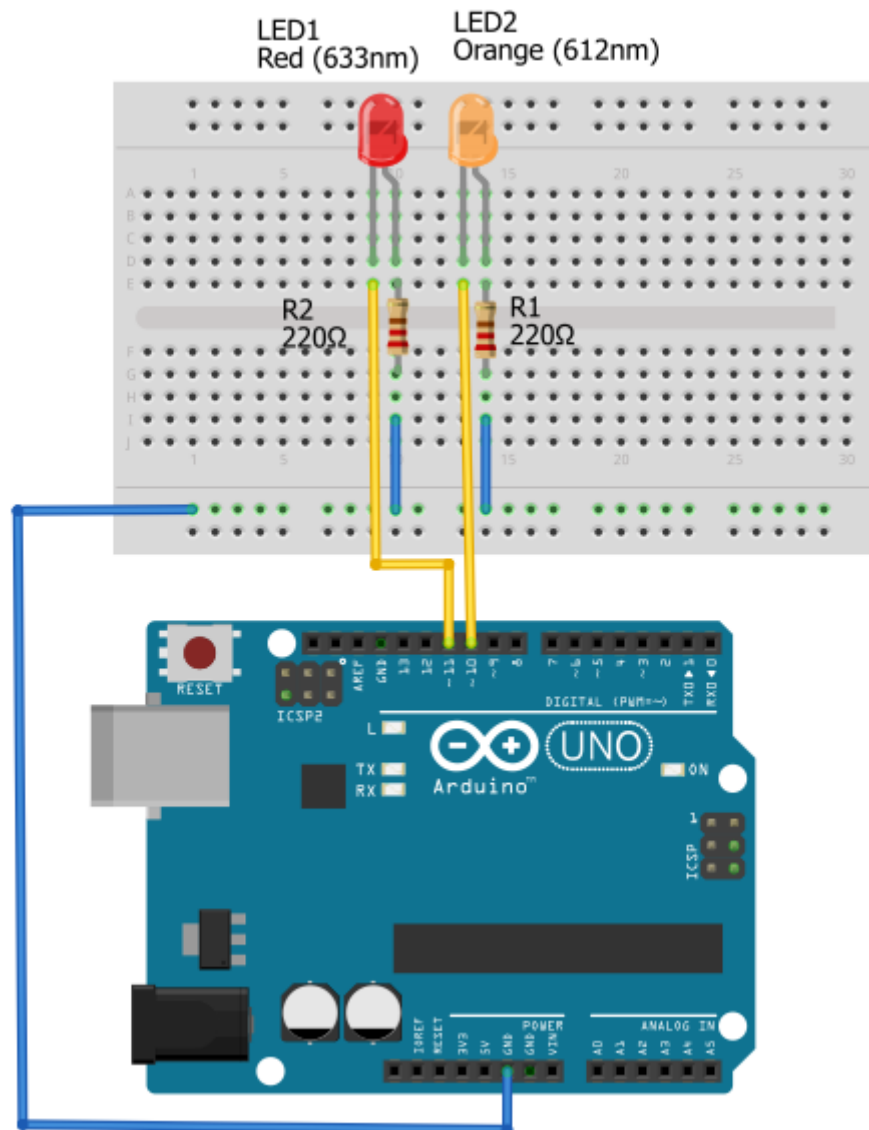


Este montaje se realiza para estudiar el comportamiento de las salidas PWM (salidas analógicas). Para ello basta con definir en este caso los pines **PIN10** y **PIN11** como salidas PWM y en el caso del gobierno del **PIN 10** inyectamos el mensaje correspondiente en el **inlet** del objeto **arduino_gui** mediante un objeto **Mensaje** en el que escribimos **analog 10 "\$1** y para el **PIN 11** lo hacemos de manera indirecta poniendo el nombre de la variable en la configuración del **slider Enviar Símbolo: PWM11** También hemos colocado un bloque **Número** para leer el valor que mandamos con el slider igualmente, en este caso, configuramos en el bloque **Número** con **Recibir Símbolo: PWM11**.

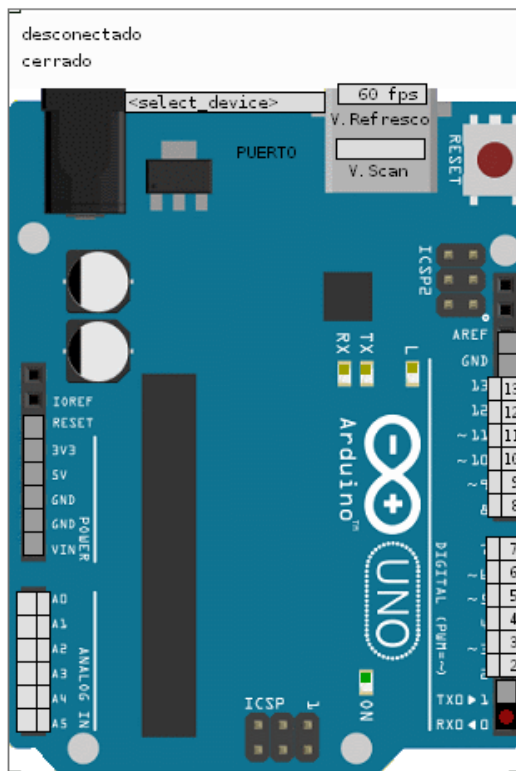
Se han colocado dos objetos **pwm** de la librería que se encuentra en la carpeta **lib** y que muestran de manera gráfica el tipo de señal que estamos mandando por cada una de las salidas **PWM**.

Los objetos **Hslider** que se utilizan para generar el valor de a enviar a las salidas deben configurarse para que estos valores sean entre 0 y 1.

Esquema de montaje



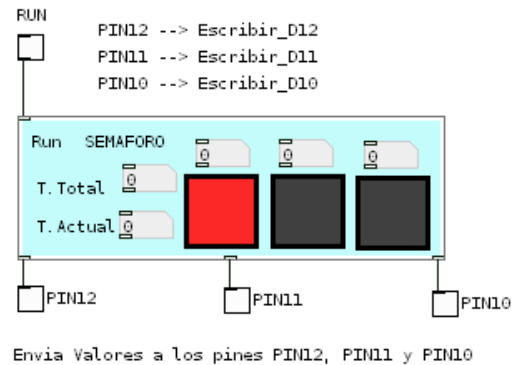
3.10. Semáforo



SEMAFORO

MODO DE ACTUACIÓN

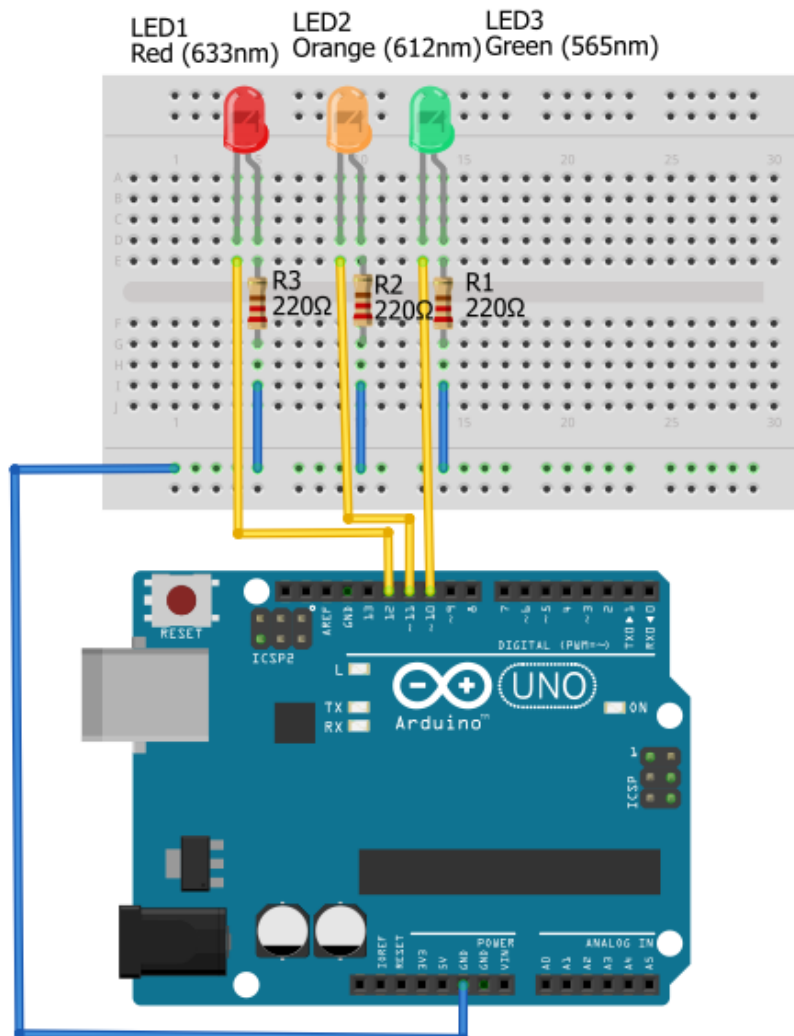
- 1 Seleccionamos el puerto
- 2 Poner los tiempos de encendido de cada salida
- 3 Pulsar el boton Run



Este es un ejemplo que permite la simulación de un semáforo. Se han seleccionado los pines **PIN12** **PIN 11** y **PIN 10** como salidas para las lámparas rojo, ámbar y verde. Se utiliza el bloque **semáforo** de la librería en la que ya implemente el control de las lámparas. La manera de actuar sería muy sencilla. Primero programamos el tiempo de encendido de cada lámpara en las cajas de dato que existen sobre los indicadores de lámpara y seguidamente activamos el **Toggle RUN**.

Las salidas del bloque semáforo son tres y se llevan directamente a tres objetos **Toggle** en los que ponemos en cada uno en su ventana de configuración los valores **Enviar Símbolo: Escribir_D12, Escribir_D11 y Escribir_D10**

Es muy sencillo aprovechar la potencialidad de Pure Data para desarrollar y/o crear nuevas librerías de objetos con las que poder realizar una amplia gama de simulaciones con Arduino así como prototipos de sistemas de control sobre los que poder estudiar la herramienta.

Esquema de montaje:

José Manuel Ruiz Gutiérrez Sep 2013
j.m.r.gutierrez@gmail.com



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