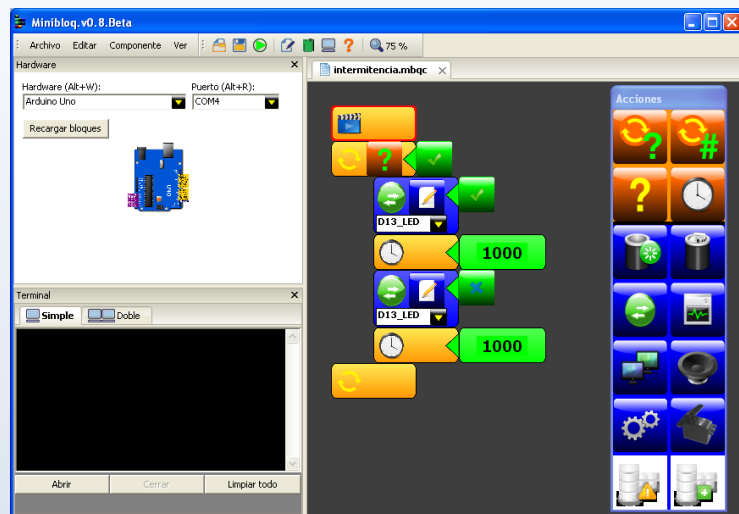


5

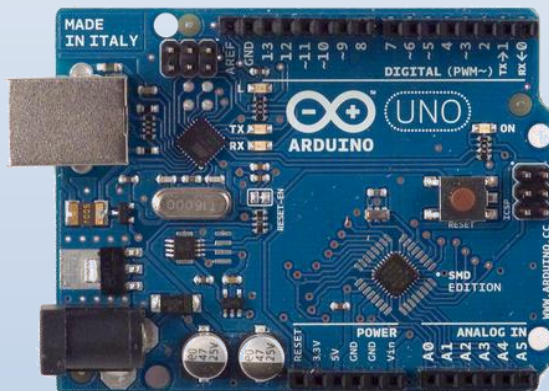
Minibloq + Arduino

Utilización del Entorno de Programación Minibloq para programar la Tarjeta Arduino



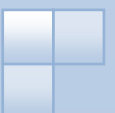
Ver. 1.0

+



José Manuel Ruiz Gutiérrez

Serie: Herramientas Gráficas para la programación de Arduino



Índice

1. Objetivo de este trabajo.
2. Una Introducción a Minibloq
3. Salida Intermitente
4. Salida intermitente con control de frecuencia mediante un canal de entrada analógica
5. Gobierno de una salida mediante un pulsador. Método 1.
6. Gobierno de una salida mediante un pulsador. Método 2.
7. Semáforo.
8. Contador sencillo.
9. Contador adelante/atrás.
10. Contador puesta a “cero”.
11. Generador de impulsos.
12. Función lógica.
13. Movimiento cíclico +180° -180° de u servo.
14. Control de posición simple de un servo (180°)
15. Control simple de un motor de cc.
16. Control de velocidad de un motor de cc.
17. Aceleración de un motor de cc.
18. Lectura de un canal analógico de entrada.
19. Simulador de un Termóstato.
20. Traspaso de un valor analógico de entrada a una salida analógica
21. Termóstato con leds y sensor LM35
22. Generación de notas musicales
23. ANEXO

Enero de 2012 Versión de Documento: V1.0

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

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

1. Objetivo de este trabajo.

Con esta *entrega número 5* de la colección “*Herramientas Gráficas para la Programación de Arduino*” abordaremos un tipo de herramientas que permiten la completa programación de la Tarjeta Arduino, depositando sobre ella el código compilado de la aplicación, es decir hablamos de herramientas del tipo IDE Arduino pero en este caso en modo gráfico.

El lenguaje de programación de Arduino es una variante muy sencilla del lenguaje C de tal modo que resulta fácil en principio elaborar sencillos ejemplos con los que programar la tarjeta, pero aun siendo fácil de programar desde su software base es muy interesante disponer de herramientas graficas. En este caso la elaboración de los programas se realiza mediante un interfaz gráfico que cuenta con unas librerías de funciones embebidas en unos bloques gráficos que se pueden ir ensamblando en una especie de “WorkFlow” que representará el “algoritmo” de la aplicación.

En este trabajo se aborda una herramienta que en mi opinión tiene muchas posibilidades de éxito en la comunidad de Arduino dado que también es un software libre.

En este trabajo aporto una colección de ejemplos que permitirán al lector comprender las posibilidades de esta herramienta y le animarán a continuar facilitándole el conocimiento de Arduino, una de las plataformas Open Hardware más interesantes y difundidas en el mundo.

Poner en la comunidad internacional Arduino este trabajo es para mí una satisfacción porque con ello creo aportar un “pequeño grano de arena” al conocimiento y a su pública y libre difusión a través de herramientas públicas y gratuitas.

Minibloq está basado completamente en software de código abierto. Es además software basado en componentes. Por lo tanto, está compuesto por diferentes paquetes. Cada uno de estos paquetes puede incluir su propio archivo de licencia. Agradezco sinceramente a *Julián U. da Silva Gillig* autor principal de Minibloq

2. Una introducción a Minibloq

Descripción General

Minibloq es un entorno de programación gráfica para Arduino TM, Multiplo, dispositivos físicos informáticos y robots. Una de sus principales objetivos es llevar la computación física y las plataformas robóticas a la escuela primaria, los niños y principiantes.

Características

Minibloq está en desarrollo. Estas son las características implementadas en la última versión de trabajo

- **Fácil:** Sólo unos pocos clics y su primer programa se está ejecutando.
- **En tiempo real generador de código:** Se crea el código, mientras que usted está agregando bloques o modificar los valores de parámetros, que muestra el código en una ventana de sintaxis de colores. De esta manera, *Minibloq facilita la transición a la programación basada en texto.*
- **Tiempo real, la comprobación de errores.**
- **Drag & drop básico con giro automático.**
- **Interfaz avanzada:** zoom, cortar y pegar, ventanas acoplables, y el teclado de navegación son sólo algunas de las características de la interfaz gráfica de usuario de Minibloq. Y hay más ...
- **Terminal incorporado:** Hay una terminal integrado que le permite enviar y recibir datos a la junta a través de puertos serie / USB.
- **Todo-en-uno-listo-para-uso-solución:** Se trata de "baterías incluidas software". El paquete incluye todo lo posible para empezar a trabajar.
- **Portable:** No requiere la instalación (a excepción de los controladores necesarios para tarjetas específicas, como Arduino TM). Se puede ejecutar desde un pen drive también. Ah, y se ejecuta por completo fuera de línea, todos ellos en su propio ordenador. Más información: Usted puede tener copias paralelas de Minibloq, incluso con la configuración de diferentes que se ejecutan en el mismo equipo.
- **Rápido:** Es una aplicación nativa, compilado con C ++ (GCC), con wxWidgets. Por esta razón, Minibloq es adecuado para ordenadores de gama baja y

notebooks. Y, también incluye los núcleos precompilados, se construye y se descargan los programas muy rápido.

- **Modular y ampliable:** El usuario puede crear nuevos bloques es propia.

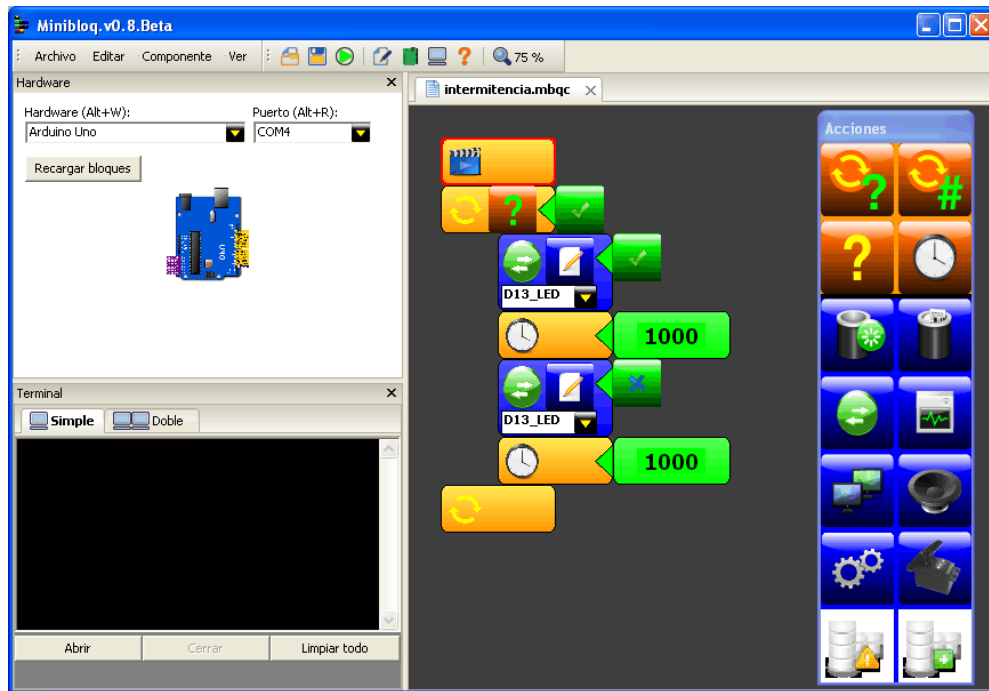
Próximas características

- **Libre y con las fuentes:** El programa estará disponible de forma gratuita, en su versión completa. Sin cargos para funciones avanzadas o similares. Sólo tienes que descargar y empezar a usarlo. Además, el código fuente completo también estará disponible. La licencia se llama RMPL (RobotGroup-Multiplo-pacifista-License). Esta licencia es básicamente una licencia MIT, pero con una restricción que prohíbe los proyectos militares.
- **Fácil integración con hardware nuevo:** Añadido soporte para nuevas plataformas y placas será sencillo. Esto puede no ser una característica para los principiantes, pero no será tan difícil de todos modos. Compiladores y lenguajes diferentes, incluso se podría añadir.
- **Internacionalización:** La primera versión estará disponible en Inglés y Español, pero el usuario y la comunidad pueden contribuir con nuevas traducciones, ya que esto sólo implica editar un archivo de texto.


Como utilizarlo:

Descargar la herramienta en <http://blog.minibloq.org/>

Una vez instalado el software bastará con iniciar el programa y aparecerá el entorno:

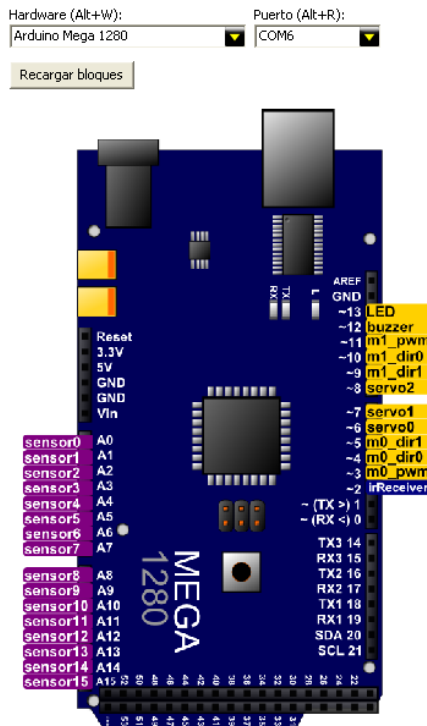


Proceso a seguir:

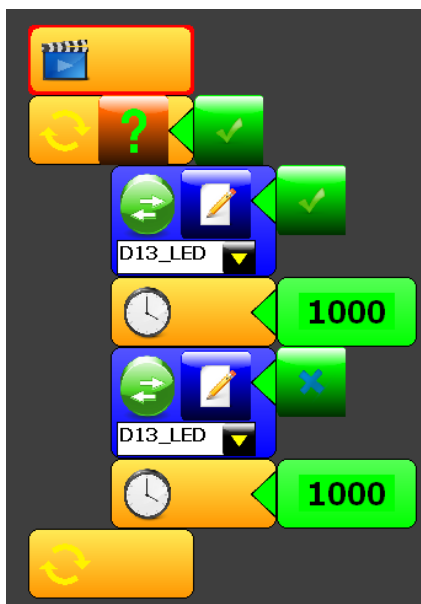
1. Seleccionamos la tarjeta de entre el grupo de tarjetas  con las que se comunica Minibloq.




- Se conecta la tarjeta al puerto USB y dejando unos segundos para que la detecte el software se selecciona el puerto en el que se ha conectado en el lugar correspondiente del entorno.



- Ya estamos en disposición de empezar a programar arrastrando bloques y colocándolos en el área correspondiente. Cada bloque deberá programarse con sus parámetros correspondientes.




4. Si lo deseamos podemos abrir la ventana  de código y ver cómo se va escribiendo el código a medida que colocamos bloques en nuestra aplicación.

```
main.pde x
#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

void setup()
{
  initBoard();

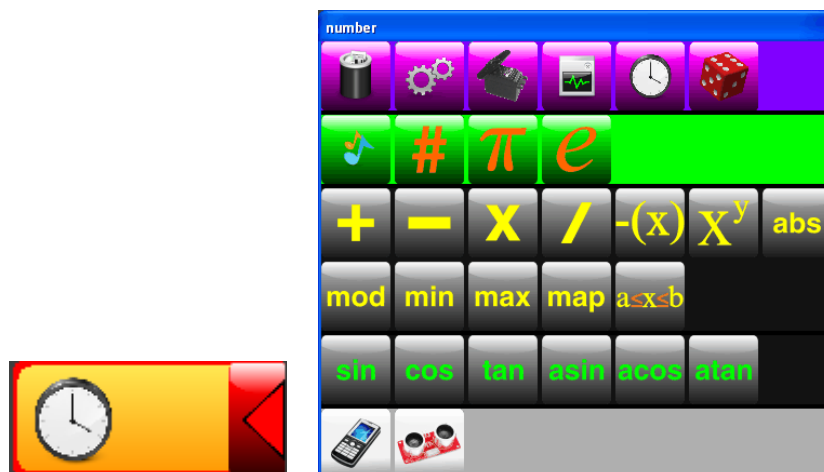
  while(true)
  {
    digitalWrite(D13_LED, true);
    delay(1000);
    digitalWrite(D13_LED, false);
    delay(1000);
  }
}

void loop()
{
}
```

5. Una vez escrito el programa se “Ejecuta”  enviándose a la tarjeta correspondiente.

Cada componente tiene unos parámetros que son configurables

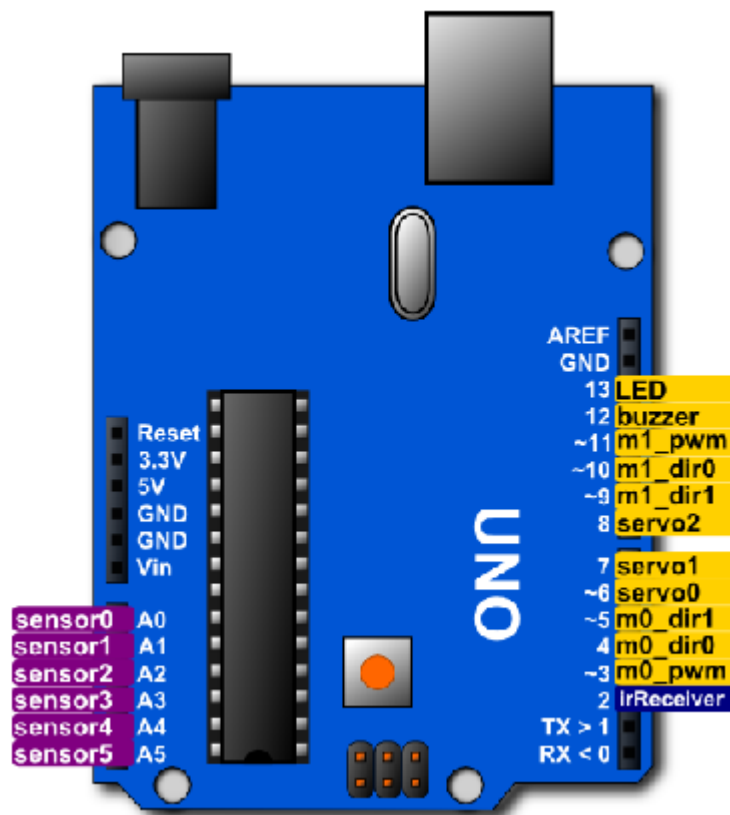
Por ejemplo en la figura se muestran el menú de configuración de un bloque de temporización



Los bloques disponibles para programar Arduino son:



Al final de este manual se ofrece un ANEXO en el que están comentado todos los bloques de Minibloq.












Arduino Uno visto desde Minibloq.

3. Salida intermitente

Nuestro primer ejercicio será el encendido y apagado de un diodo led conectado en la salida PIN 13 de la Tarjeta Arduino.

El programa lo que debe hacer es activar la salida PIN 13 durante 0,2 seg. Y desactivarla durante 1 seg. En un bucle continuo.

1. Recurrimos al bloque “**mientras que**”  que se activará siempre manteniendo la condición como **true** .
2. Dentro de este bucle mediante el bloque “**IOPin (setter)**”  con la condición true (encender)  activa la salida **D13_LED** que es el **PIN 13** de la Tarjeta Arduino.
3. Seguidamente realizamos la temporización (tiempo salida activa) mediante el bloque “**retraso**”  durante 200 ms es decir 0,2 seg.
4. El siguiente paso es desactivar la salida **PIN 13** y lo hacemos de nuevo con el bloque “**IOPin (setter)**”  solo que esta vez le ponemos la constante falsa  (apagada).
5. Finalmente se coloca de nuevo un de “**retraso**”  bloque de temporización (tiempo de salida desactivada) durante 1000 ms. Es decir 1 seg.

En la siguiente figura vemos el algoritmo completo y junto a el se muestra el código generado que será compilado y descargado a la aplicación cuando pulsemos el botón “**ejecutar**” .

```

#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

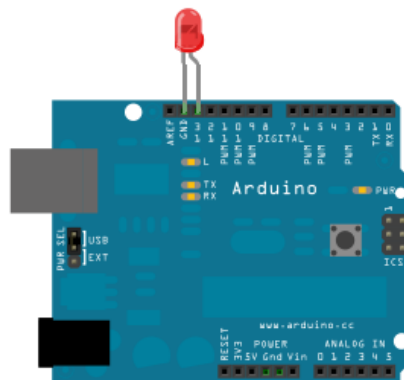
void setup()
{
  initBoard();

  while(true)
  {
    DigitalWrite(D13_LED, true);
    delay(200);
    DigitalWrite(D13_LED, false);
    delay(1000);
  }
}

void loop()
{
}

```

El montaje de este primer ejemplo es muy sencillo y se muestra en la siguiente imagen.



Made with Fritzing.org

4. Salida intermitente con control de frecuencia mediante un canal de entrada analógica.

En este ejemplo vamos a controlar la frecuencia de apagado del encendido/apagado de la salida **PIN 13** con un valor que recogemos de una de las entradas analógicas de Arduino Analog0

Debemos definir una variable a la que llamaremos “**frecuencia**” que será justo el valor de los tiempos **Te**=tiempo encendido y **Ta**=tiempo apagado. En este caso ambos vamos a hacerlos iguales, es decir:




$$\mathbf{Te=Ta=frecuencia}$$



A la hora de asignar el valor a la frecuencia debemos escalarlo, es decir lo multiplicamos, en este caso por 10 dado que el valor que se puede leer en el canal Analog0 es de 0 a 1024.


$$\mathbf{frecuencia=valor_Analog0 * 10}$$



El programa se muestra en la figura siguiente y consiste en lo siguiente:

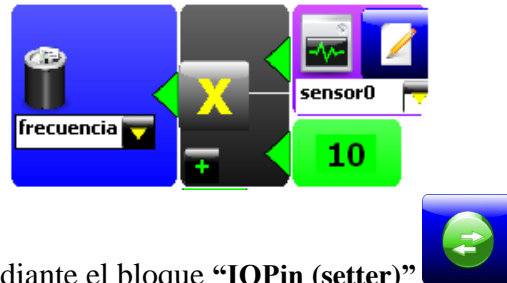
1. En primer lugar forzamos el valor de la variable frecuencia a cero para ello



nos servimos del bloque “**variable**”  en el que escribimos el nombre de la variable y como argumento de entrada le ponemos el valor “**número**”  “**cero**” 

2. Seguidamente, colocamos el bloque “**mientras que**”  que se activará siempre manteniendo la condición como **true** .


3. Dentro del bucle la primera operación es asignar a la variable frecuencia el valor del canal Analógico 0 “**sensor0**” multiplicado por 10. Esto lo conseguimos con el bloque “**variable**”  al que le ponemos de entrada el

resultado de una operación matemática “x”  en el que el primer dato es el canal  **sensor0** y el segundo el valor numérico **10**





4. Se activa la salida PIN 13 mediante el bloque “IOPin (setter)”  con la condición true (encender) 




5. Seguidamente realizamos la temporización (tiempo salida activa) mediante el bloque “retaso”  durante el tiempo que en esta caso asociamos a la variable “frecuencia”.



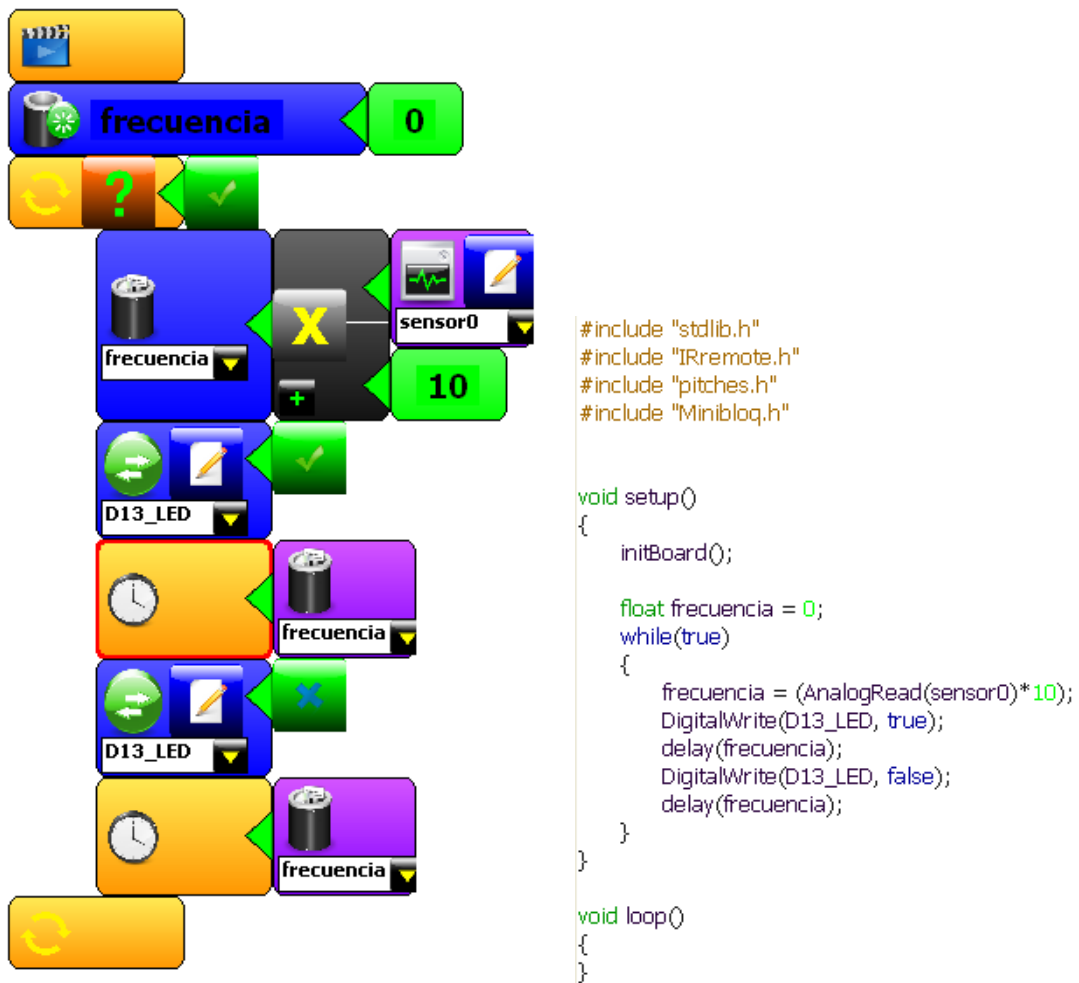
6. El siguiente paso es desactivar la salida PIN 13 y lo hacemos de nuevo con el bloque “IOPin”  solo que esta vez le ponemos la constante falsa  (apagada).



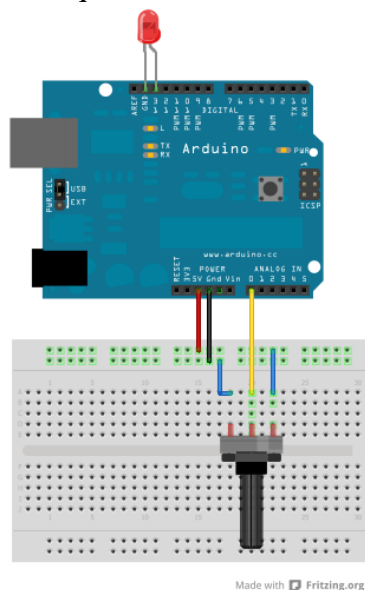
7. Finalmente se coloca de nuevo un de “retraso”  bloque de temporización (tiempo de salida desactivada) el tiempo que en esta caso asociamos a la variable “frecuencia”.



En la siguiente figura se muestra el algoritmo completo.





A continuación se muestra el esquema de conexión de los componentes



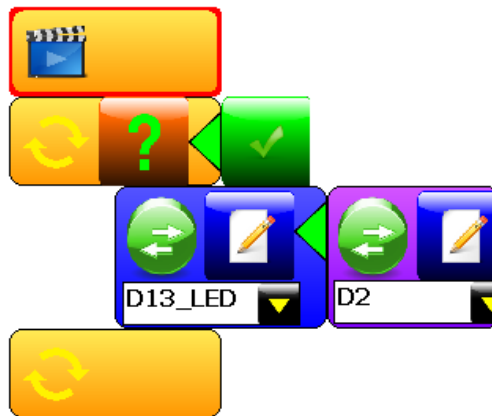
5. Gobierno de una salida mediante un pulsador. Metodo1

Queremos gobernar una salida digital **PIN 13** mediante el accionamiento de una entrada digital **PIN 2** a la que le hemos colocado un pulsador.

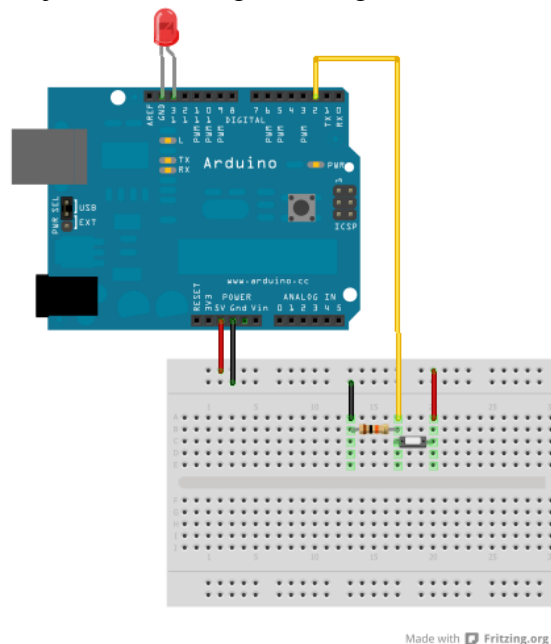
El programa es muy sencillo. Mediante el bloque “IOPin (setter)”  asignado al D13_LED equivalente al PIN 13, al que ,como entrada, le hemos fijado el valor que

entrega bloque de tipo “IOPin (captador)”  al que le hemos asignado el valor D2 que se corresponde ala entrada PIN 2 de Arduino

En la siguiente figura vemos el aspecto del programa.







El esquema del montaje es el de la siguiente figura.



6. Gobierno de una salida mediante un pulsador. Metodo2

En este segundo método gobernaremos igualmente una salida digital **PIN 13** mediante el accionamiento de una entrada digital **PIN 2** a la que le hemos colocado un pulsador.

El programa se montaría de la siguiente manera.

1. Se pondría como siempre el bucle “repetir”  siempre activo .
2. El programa en este caso incorpora una estructura condicional  en la que la condición es el estado de la entrada PIN 2 que se detectara mediante la funciones “**IOPin (captador)**” .
3. Si se cumple que la entrada PIN 2 vale “true” entonces se ejecuta la parte si del condicional mediante la instrucción “**IOPin (setter)**”



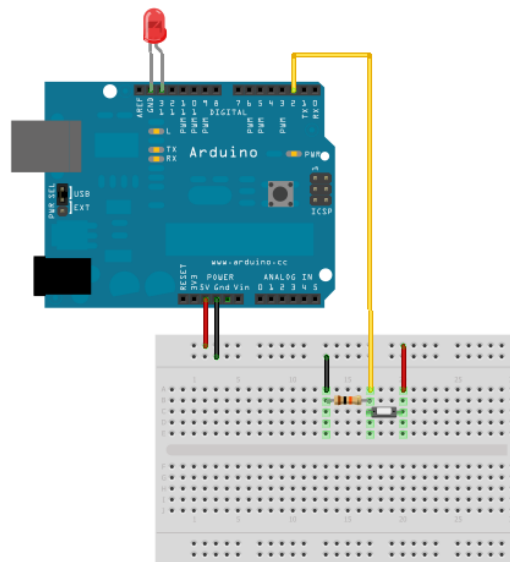
4. Si no se cumple la condición entonces la salida D13_LED se pone en “false” mediante la instrucción “**IOPin (setter)**”



En la siguiente figura vemos el aspecto del programa.



El esquema del montaje es el de la siguiente figura.

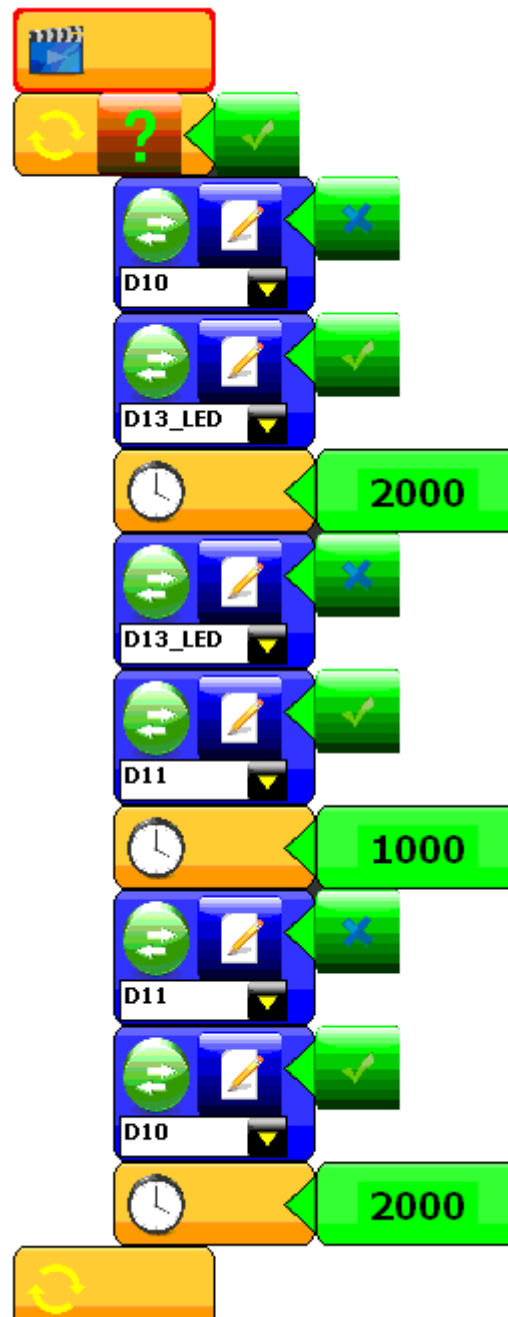


Made with  Fritzing.org

7. Semáforo

En este ejemplo vamos a realizar un semáforo. Utilizaremos las siguientes salidas

Lámpara Roja	PIN 13	Tr=Tiempo Rojo	2 seg.
Lámpara Ámbar	PIN 11	Ta=Tiempo Ambar	1seg
Lámpara Verde	PIN 10	Tv=Tiempo Verde	2 seg.



El programa es muy sencillo se trata de activar y desactivar en la secuencia apropiada las tres lámparas dejando entre cada secuencia los tiempos correspondientes.

Para ello se han utilizado funciones del tipo **“IOPin (setter)”** tanto para el encendido como para el apagado.



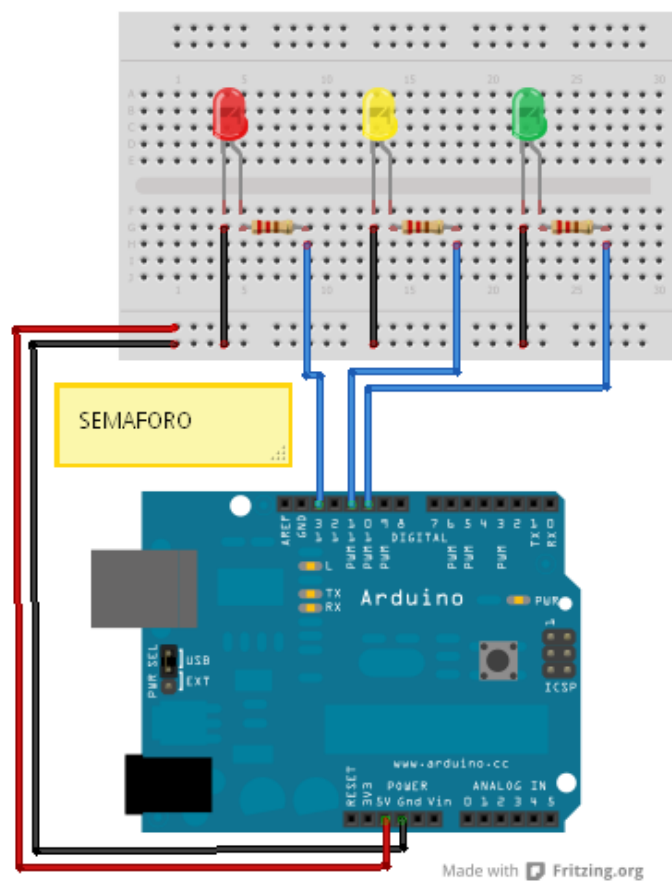
Apagado

Encendido

La temporizaciones se han realizado con bloques de función tipo **“retraso”**



En la siguiente imagen se muestra el circuito de montaje con protoboard.

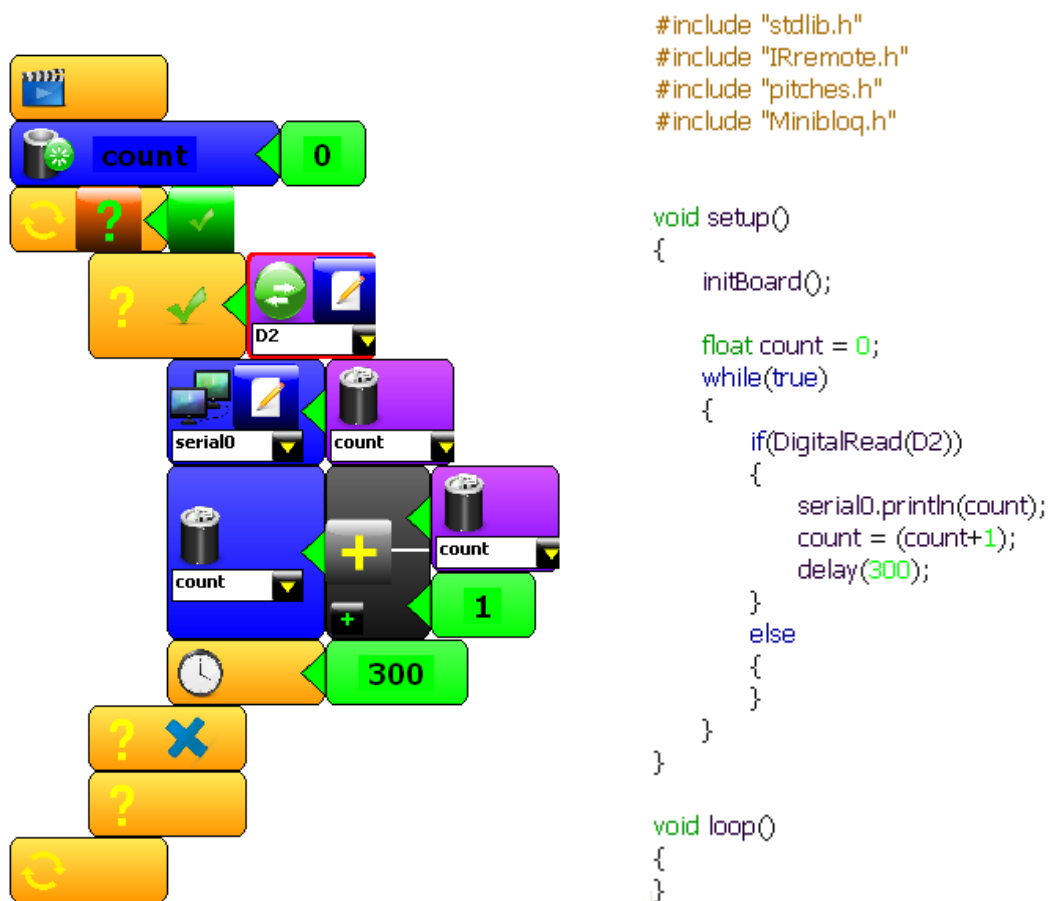


8. Contador Sencillo

Contar es una función muy útil en los sistemas, es por ello por lo que a continuación vamos a estudiar el siguiente ejemplo.

Se trata de contar los impulsos que van entrando por una de las entradas digitales **PIN 2**.

Para constatar que se está realizando la cuenta haremos uso del terminal de visualización de puerto que incorpora el software Minibloq, mediante este terminal veremos como se va incrementando el valor del contador a medida que se van recibiendo los impulsos de cuenta.



```

#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

void setup()
{
  initBoard();


  float count = 0;
  while(true)
  {
    if(DigitalRead(D2))
    {
      serial0.println(count);
      count = (count+1);
      delay(300);
    }
    else
    {
    }
  }
}

void loop()
{
}

```

En la figura anterior podemos observar como sería el algoritmo para realizar esta aplicación y el código generado.

1. En primer lugar definimos una variable que le llamaremos **count** mediante el

bloque de función “variable (crear)” 

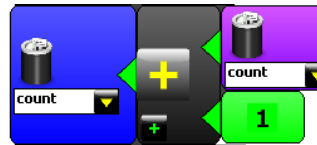
2. A continuación se implanta un bloque de repetición continua “mientras que”
3. Seguidamente se colocara un bloque condicional “Si” en el que la condición será que la entrada D2 sea “true”



4. Seguidamente pondremos un bloque el tipo “serie (setter)” mediante el cual enviamos al puerto USB de Arduino el valor que le consignemos, en este caso la variable “count”

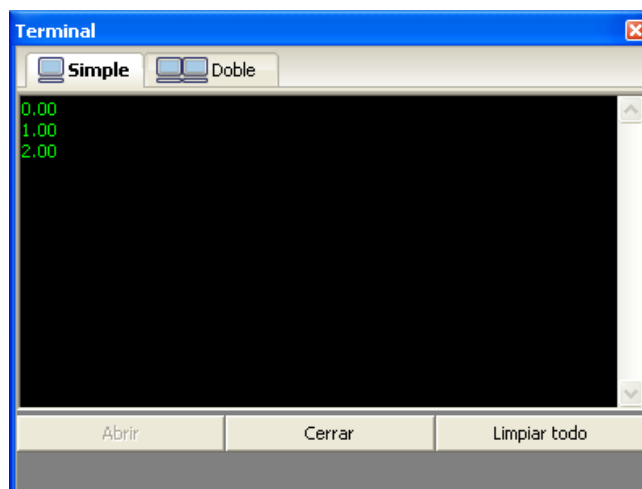


5. Lo siguiente es realizar el incremento de la variable “count”, mediante un bloque de función de tipo “variable (asignar)”

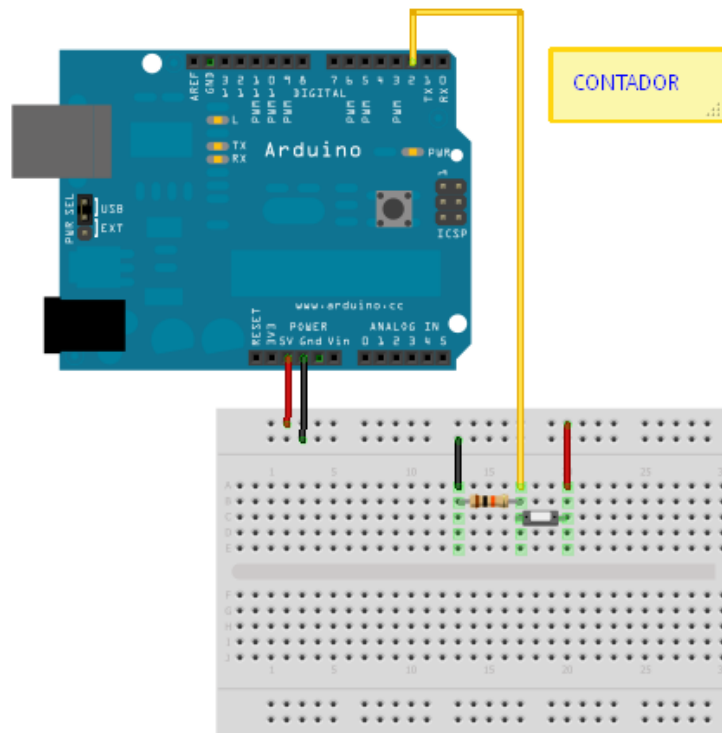


6. Finalmente se ha colocado un retardo con el fin de que el refresco del valor de la variable a mostrar sea cada cierto tiempo que permita leer fácilmente.

Vemos en la siguiente imagen el aspecto de la ventana de visualización de valores recibidos en el puerto.



Este es el esquema de montaje en tarjeta protoboard de la aplicación.



Made with  Fritzing.org

9. Contador adelante/atrás

Esta es una variante del ejercicio anterior en la que deseamos poder contar hacia adelante o hacia atrás haciendo uso de dos entradas digitales **Digital2** y **Digital3** correspondientes a los pines **PIN 2** y **PIN 3** de la tarjeta Arduino respectivamente.

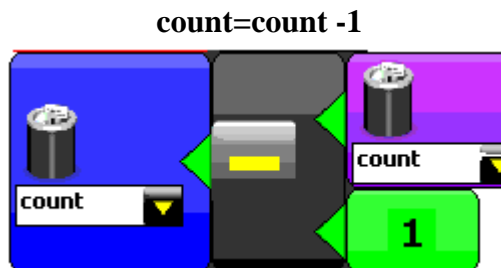
De la misma manera que hemos hecho en el ejemplo anterior definimos la variable **count** que almacenará el valor de contador.

Esta vez dispondremos de dos bucles tipo “si” uno para cada una de las dos operaciones “**contar**” y “**descontar**”

1. Descontar:



Para el bucle descontar testamos el estado de la variable de entrada digital **Digital3 PIN 3** y si se cumple que esta activada incrementamos el contador mediante un bloque de función de tipo “**variable (asignar)**” en el que la entrada es **count -1**:



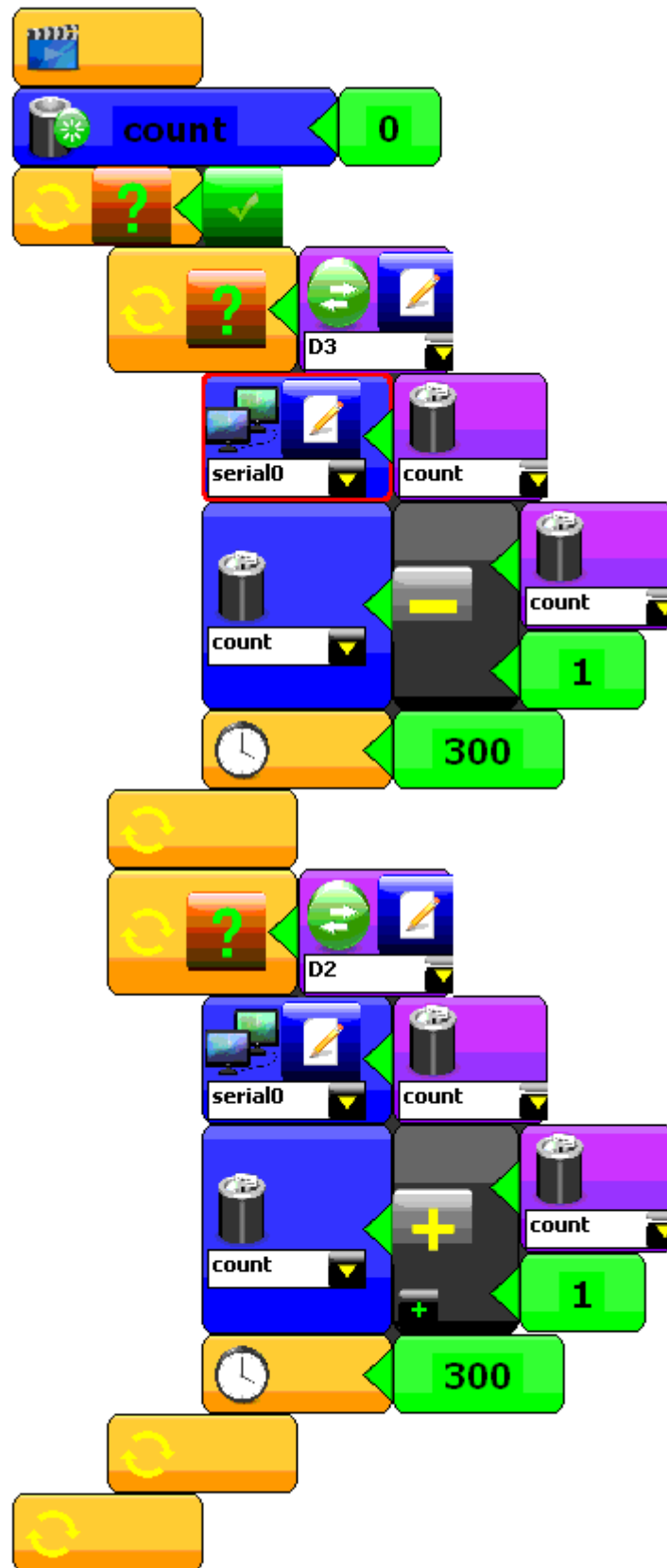
2. No olvidemos que se utilizara también el bloque tipo “**serie (setter)**” mediante el cual enviamos al puerto USB de Arduino el valor que le consignemos, en este caso la variable “**count**”



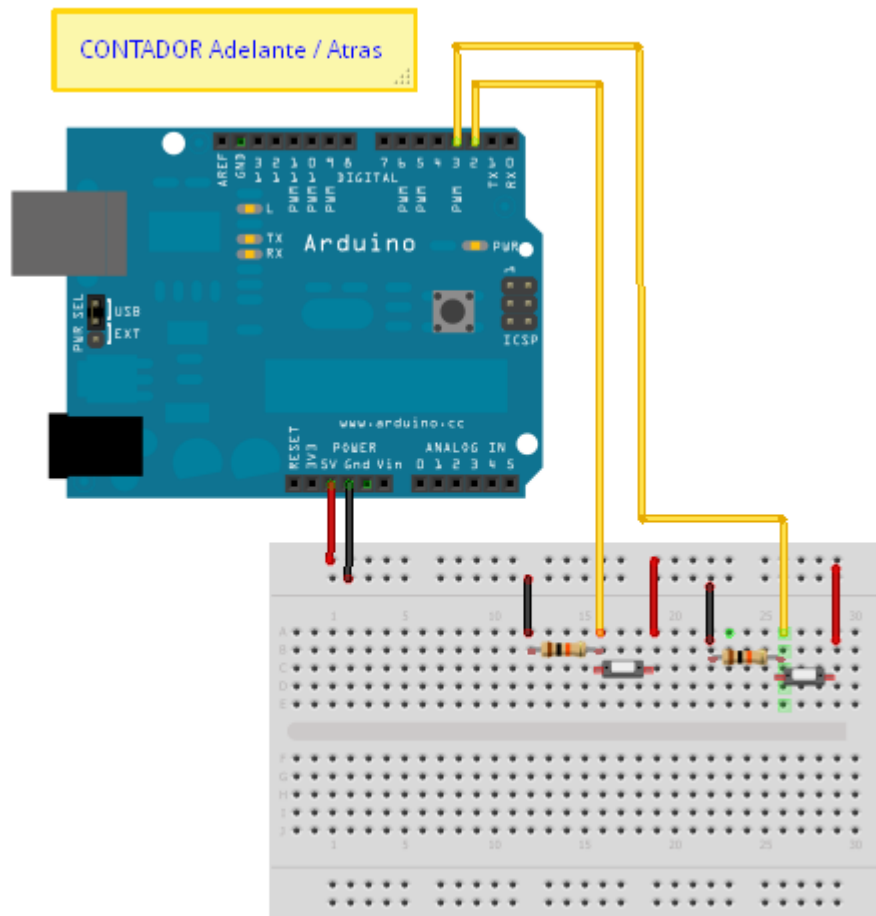
3. Contar.

En este caso el condicional prueba la entrada de contar hacia adelante **PIN 2** y lo demás es igual que en el ejemplo del contador sencillo.

Este es el esquema de bloques de la aplicación.



Este es el montaje que deberíamos realizar para probar la aplicación.



Made with  Fritzing.org

10. Contador con puesta a “cero”

En este ejemplo deseamos poder realizar la “puesta a cero” del valor del contador. Para este montaje dispondremos de dos pulsadores conectados a las entradas digitales **Digital2** y **Digital3** correspondientes a los pines **PIN 2** y **PIN 3** de la tarjeta Arduino respectivamente.

Digital2 (PIN 2)	Será la para la entrada de impulso de cuenta
Digital3 (PIN 3)	Será la entrada para la puesta a cero

De la misma manera que hemos hecho en el ejemplo anterior definimos la variable **count** que almacenará el valor de contador.

Esta vez dispondremos de dos bucles tipo “**si**” uno para cada una de las dos operaciones “**poner a cero**” y “**contar**”

1. Bucle de puesta a cero. En este caso se trata de montar un bucle condicional del tipo “**si**” cuya condición de ejecución es encontrarse pulsado el botón que colocamos en la entrada D3 **PIN 3** de la tarjeta. Si se cumple la variable se ejecuta:

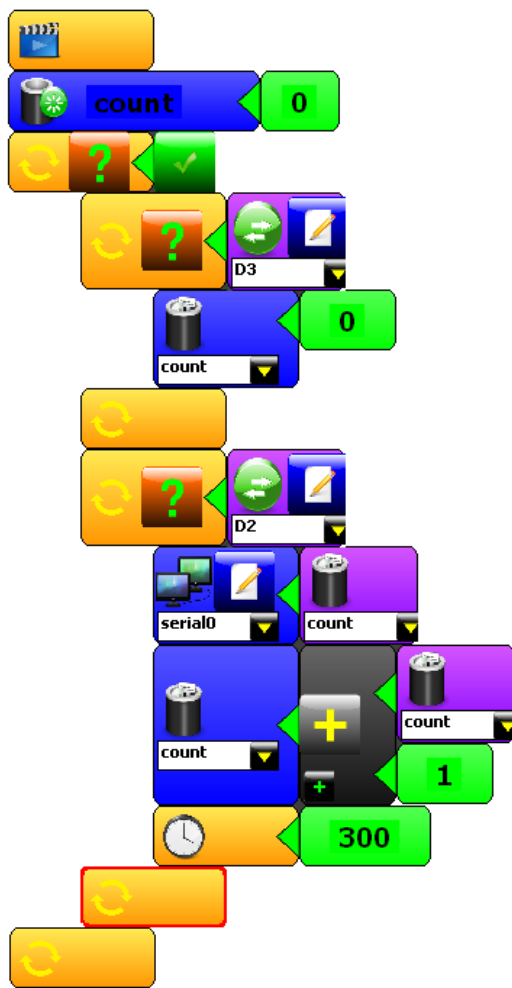
count=0

2. Bucle de cuenta. Este es exactamente igual que en los casos anteriores. El bucle se ejecuta siempre que entre un valor “**true**” por la entrada D2 **PIN 2** de la tarjeta Arduino:

count=count+1

3. Se ha previsto como en los casos anteriores la visualización del valor de la variable **count** mediante la herramienta **Terminal** de Minibloq

En la figura siguiente vemos el esquema gráfico del algoritmo que se debe crear junto con el código generado para enviar al IDE Arduino



```

#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

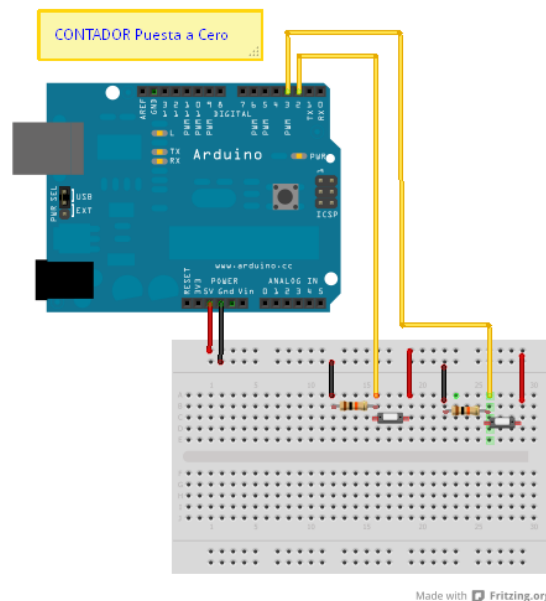
void setup()
{
  initBoard();

  float count = 0;
  while(true)
  {
    while(DigitalRead(D3))
    {
      count = 0;
    }
    while(DigitalRead(D2))
    {
      serial0.println(count);
      count = (count+1);
      delay(300);
    }
  }
}

void loop()
{
}

```

El siguiente es el montaje de la aplicación.



11. Generador de impulsos

En esta aplicación vamos a generar **cinco impulsos** en la salida la salida **PIN 13** cuando previamente se pulse un botón conectado en la entrada D2 PIN 2 de la tarjeta Arduino.

Las variables del programa son:

- **count** almacena el estado de cuenta
- D13_LED **PIN 13** salida de los impulsos
- D2 **PIN 2** Entada de pulsador para orden de generación de los impulsos
- El número de impulso a generar esta implícito en el programa y es **5**

Descripción del programa:

1. La primera acción es poner la variable `count=0`



2. A continuación, dentro de un bucle de ejecución continua pondremos dos bucles “anidados” (uno dentro del otro).
3. El primer bucle será para testear si se pulso la orden de para generar los 5 impulsos que vendrá de la entrada D2



4. A continuación. Si se cumple la condición se deben general los 5 impulsos. Por ello preguntaremos con un condicional si acaso **aun count<5** (no se han generado los impulsos) No olvidemos que el primer impulso es el numero “0” por lo tanto el último número que contara será el 4



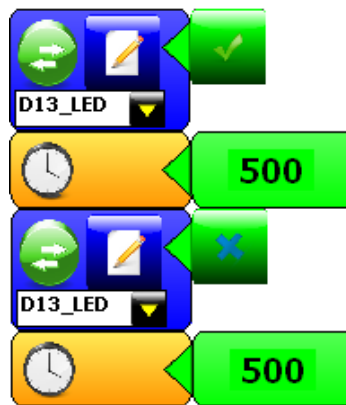
- Lo que hemos de hacer a continuación es enviar el valor de “**count**” al puerto para su visualización en el **Terminal**.



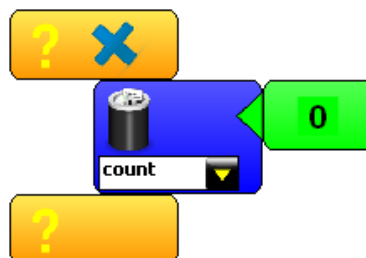
- Seguidamente se incremente la variable **count**



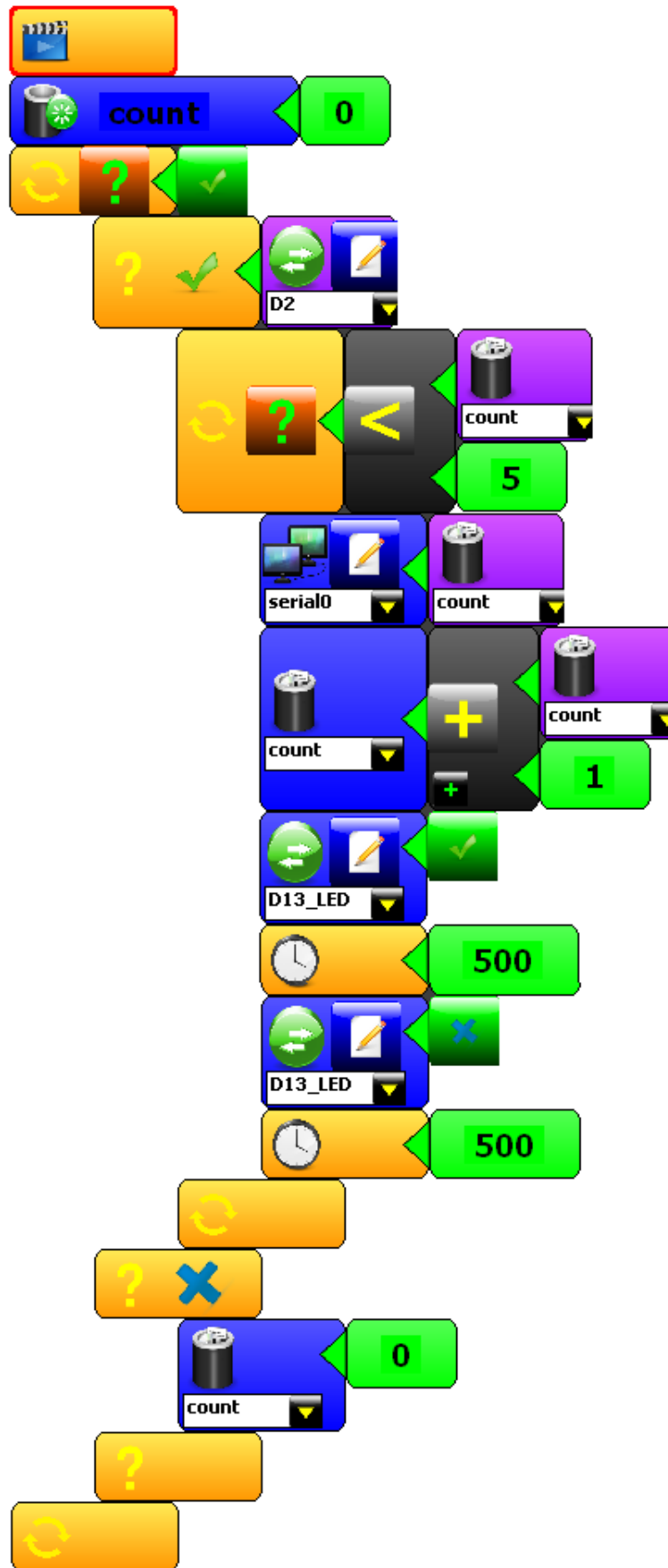
- Después se pasa a generar un impulso en la salida **PIN 13**.



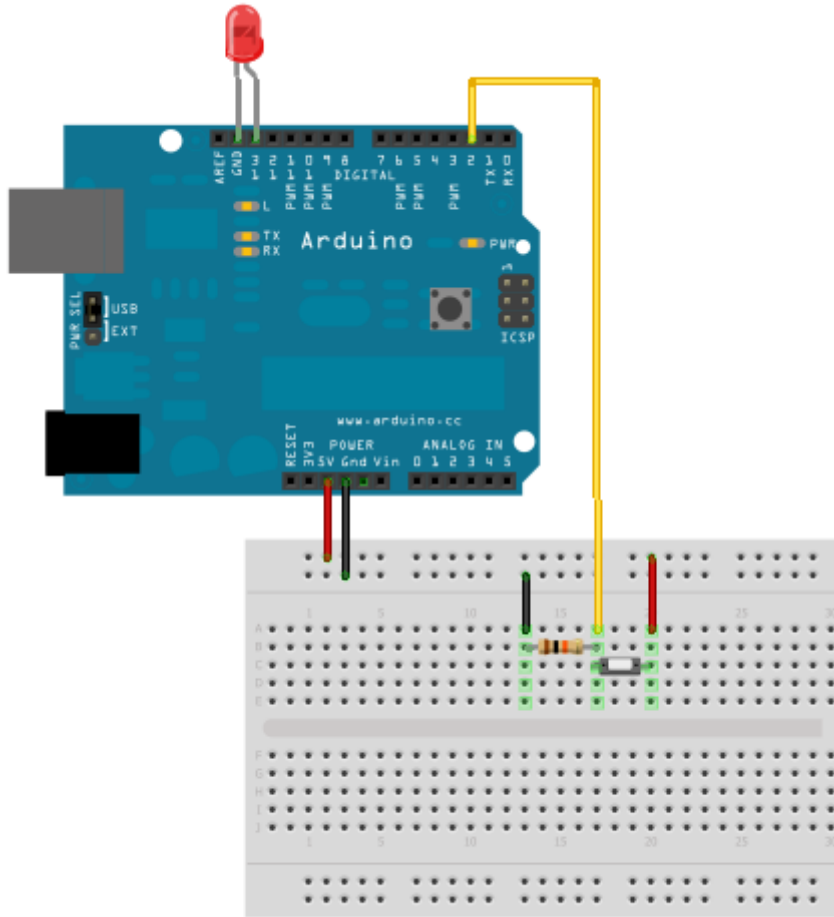
- Finalmente fuera del bucle en el bucle anterior se debe poner de nuevo a cero la variable **count** ya que se supone que si no se cumple la condición **count<5** es porque ya se ha alcanzado el final de cuenta.



Seguidamente, en la página siguiente, aparece el programa completo.



La siguiente figura muestra el montaje sobre protoboard.



Made with  Fritzing.org


12. Función lógica

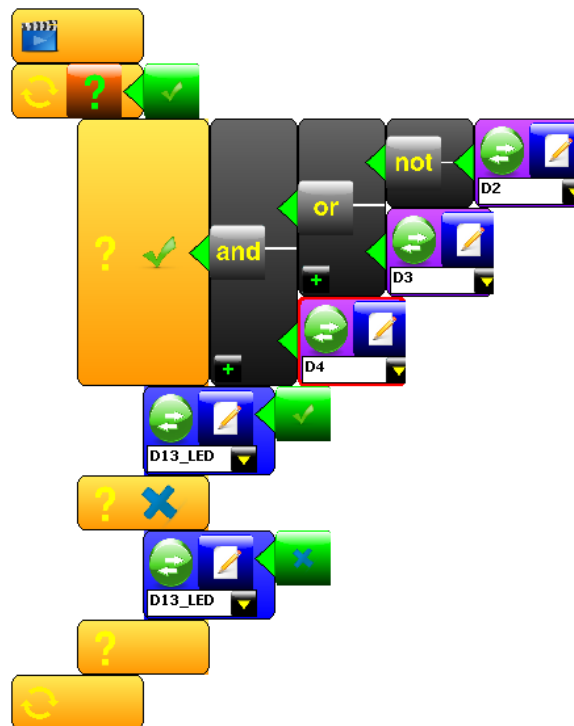
Con este ejemplo vamos a demostrar como se pueen programar funciones lógicas mediante Minibloq.

Vamos a realizar una función con tres variables de entrada D2, D3 y D4 (PIN 2, PIN 3 y PIN 4 de Arduino). La salida se llevará a la salida digital D13_LED PIN 13 de Arduino. La función será:

$$D13_LED=(D2' \text{ OR } D3) \text{ AND } D4$$

Modo de resolución.

1. Recurrimos a un bloque de función tipo “si”  en el que ponemos la condición haciendo uso de los operadores “and”, “or” y “not”.
2. En la parte de la “condición cumplida” ponemos un bloque “IOpin (setter)” con el que activamos la salida D13_LED.
3. Si la condición n se cumple desactivamos la salida con un bloque “IOpin (setter)”



El código generado es el siguiente:

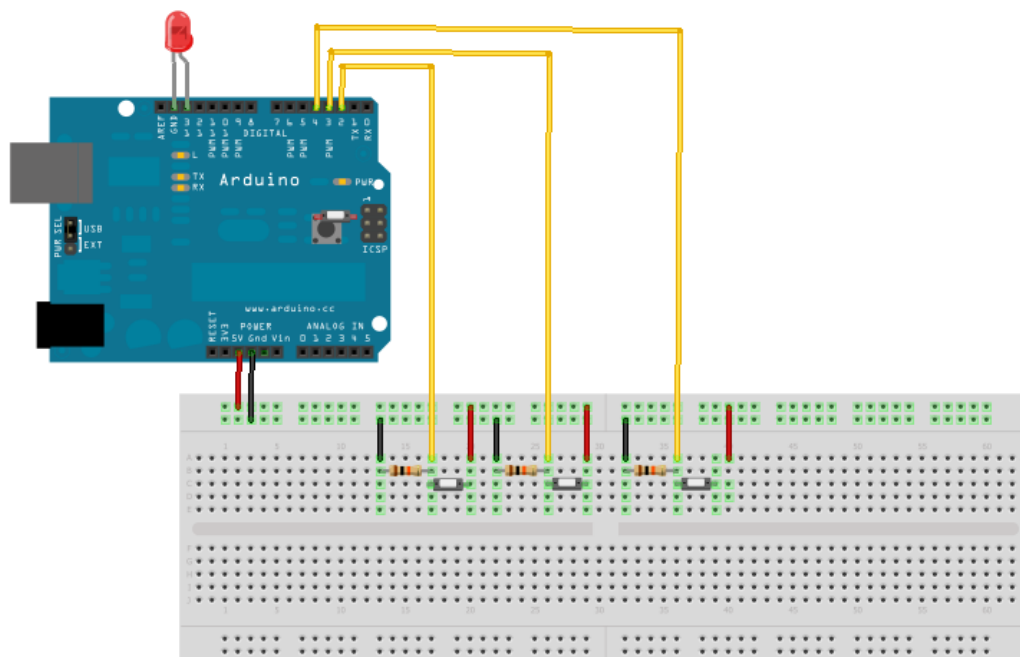
```
#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

void setup()
{
  initBoard();

  while(true)
  {
    if(((!(DigitalRead(D2)) || DigitalRead(D3)) && DigitalRead(D4)))
    {
      DigitalWrite(D13_LED, true);
    }
    else
    {
      DigitalWrite(D13_LED, false);
    }
  }
}

void loop()
{
}
```

Esquema de la aplicación.



Made with  Fritzing.org

13. Movimiento cíclico +180° -180° de un servo.

Con el siguiente montaje realizamos el movimiento cíclico de un servo (180°) del tipo:

De 0° a 180° y de 180° a 0° en pasos de dos grados.

Se trata de posicionar el servo en la “posición de referencia (0°)” y desde esta realizar un avance en sentido ascendente en pasos de 2° hasta llegar a 180. Seguidamente se realiza el retorno a la posición de 0° en pasos de 2°.

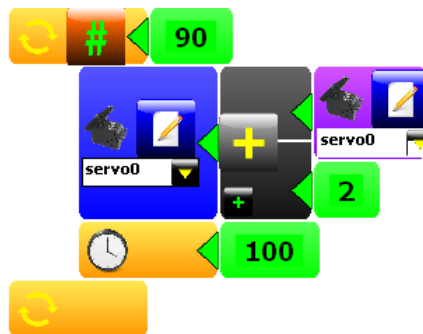
Se recurre a la implantación de dos bucles que se ejecutaran hasta 90 veces.

Proceso:

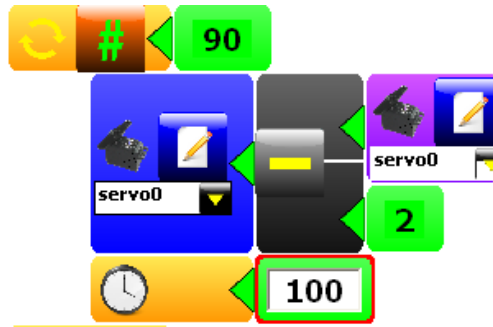
1. En primer lugar se pone el servo en la posición 0. “**ServoRC (setter)**”



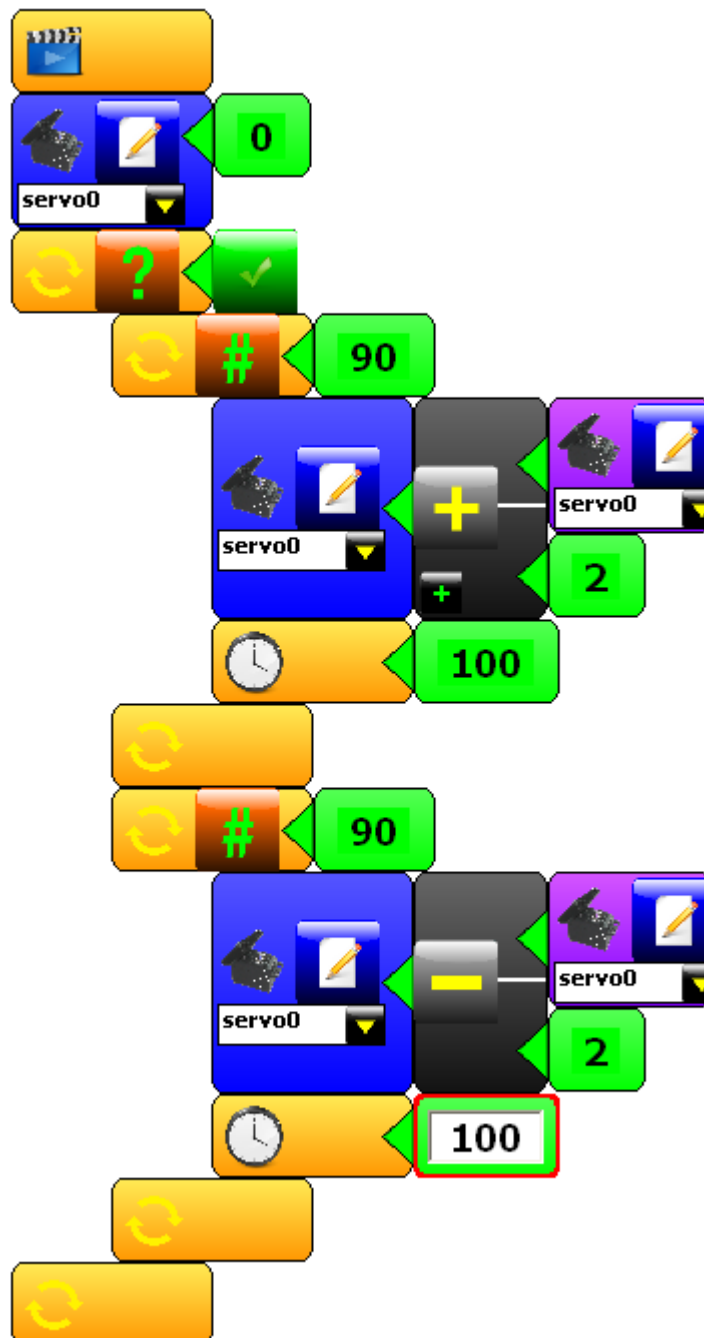
2. Realización del primer bucle: En este primero se realiza el bucle durante 90 veces. El avance el servo se realiza sumando 2° a la posición anterior (partiremos de 0°) con retardos de 0,1 segundo. “**ServoRC (setter)**” y “**retraso**”



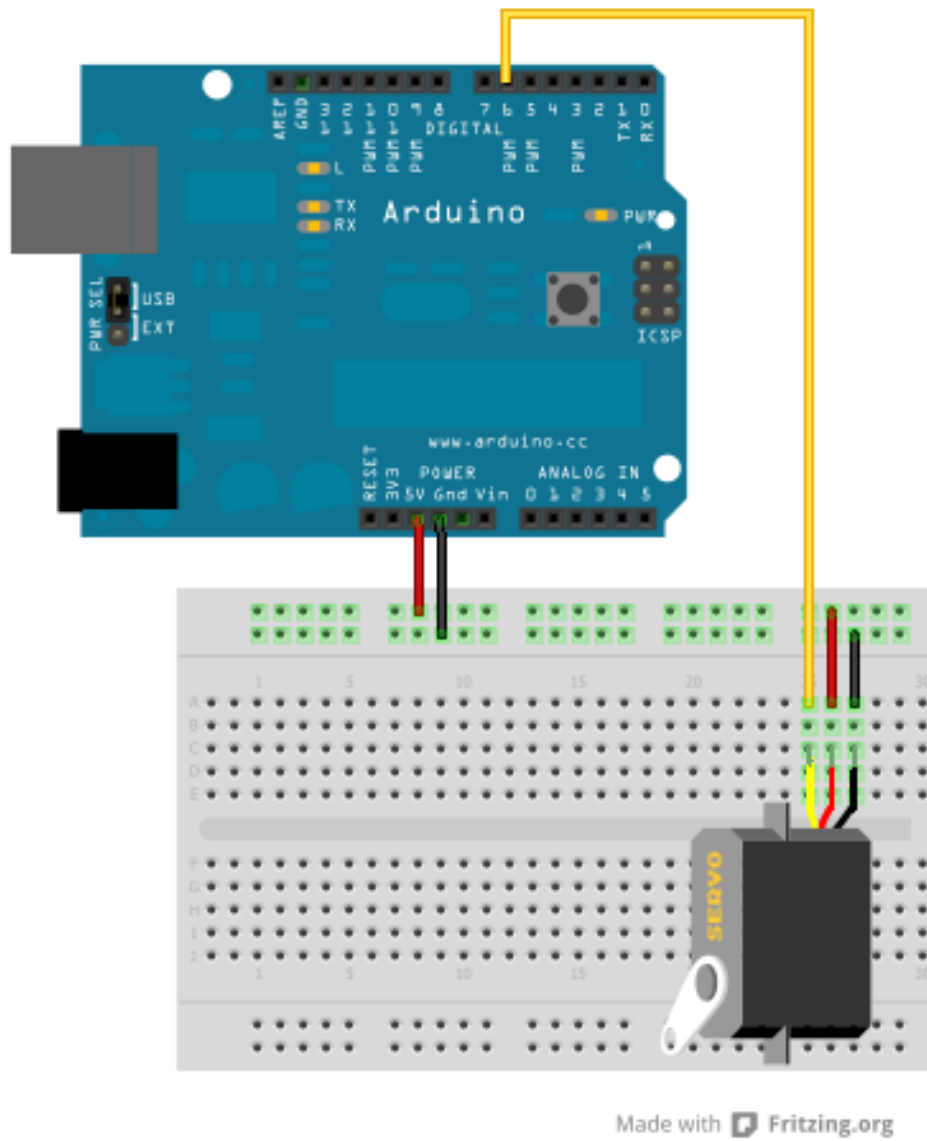
3. Realización del segundo bucle: En este segundo bucle también se realiza durante 90 veces. El retroceso del servo se realiza estando 2° a la posición anterior (partiremos de 180°) con retardos de 0,1 segundo. “**ServoRC (setter)**” y “**retraso**”



A continuación vemos el esquema completo.



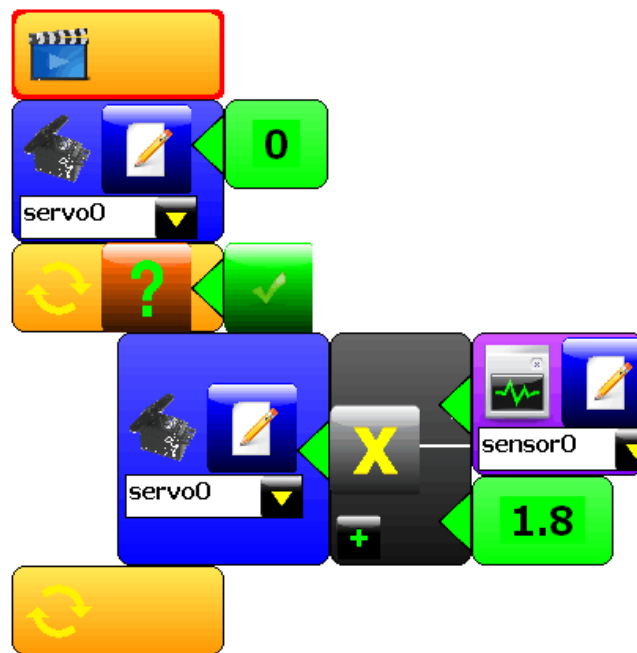
El esquema de montaje es el siguiente.



Se ha utilizado para la práctica un servo: **Futaba S3003**

14. Control de posición simple de un servo (180°)

El control de la posición de un servo es una tarea muy común en los sistemas de control. En este ejemplo vamos a realizar el control de la posición (ángulo girado) de un servo ayudándonos de una de las entradas analógicas de Arduino **Analog0** (**sensor0** en la nomenclatura de Minibloq) a la que le colocamos un potenciómetro. El servo sobre el que actuaremos es el **Servo 0** que implementa el software Minibloq conectado al terminal **PIN 6** de la tarjeta.



El algoritmo es muy sencillo.

1. En primer lugar se fija a cero la posición del servo (posición de referencia o reposo) mediante el bloque “**ServoRC (setter)**”
2. Seguidamente dentro del bucle de repetición de se vuelve a poner el bloque “**ServoRC (Setter)**”
3. El parámetro que ponemos al servo (ángulo a girar) se obtiene mediante un sencillo cálculo que se implementa en el propio algoritmo:
4. **Angulo= Valor Sensor0 * 1.8**

5. Esto se hace porque el valor máximo que miden los sensores en esta implementación de Minibloq es 100 por lo que para alcanzar el valor máximo de ángulo (180) tenemos que multiplicar por 1.8

Seguidamente vemos el código generado pro Minibloq

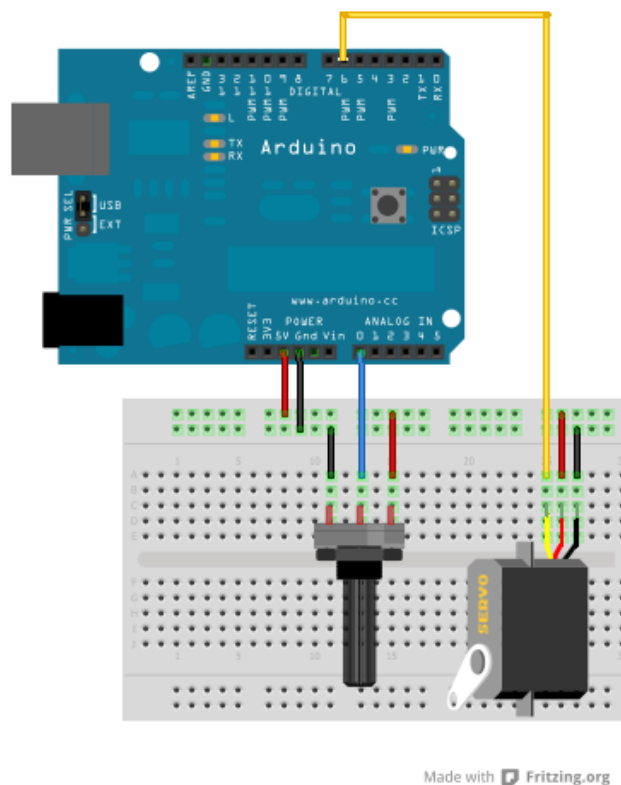
```
#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

void setup()
{
  initBoard();

  servo0.attachAndWrite(0);
  while(true)
  {
    servo0.attachAndWrite((AnalogRead(sensor0)*1.8));
  }
}

void loop()
{
}
```

El esquema de montaje es el siguiente.



Se ha utilizado para la práctica un servo: **Futaba S3003**

15. Control simple de un motor de cc.

En este caso vamos a controlar un motor de corriente continua que marchará hacia adelante a una velocidad de **45** durante **1seg.** y retrocederá a la misma velocidad -**45** durante 1seg. Esto lo hará mientras que mantengamos pulsado un pulsador conectado a la entrada digital D9 **PIN 9** de Arduino



```

#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

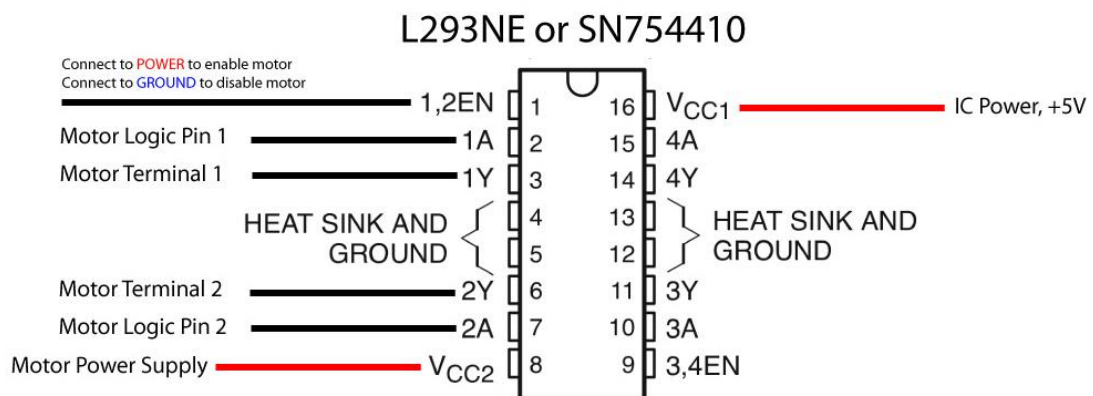
void setup()
{
  initBoard();

  while(true)
  {
    while(DigitalRead(D9))
    {
      motor0.setSpeed(45);
      delay(1000);
      motor0.setSpeed(-(45));
      delay(1000);
      motor0.setSpeed(0);
    }
  }

  void loop()
  {
  }
}

```

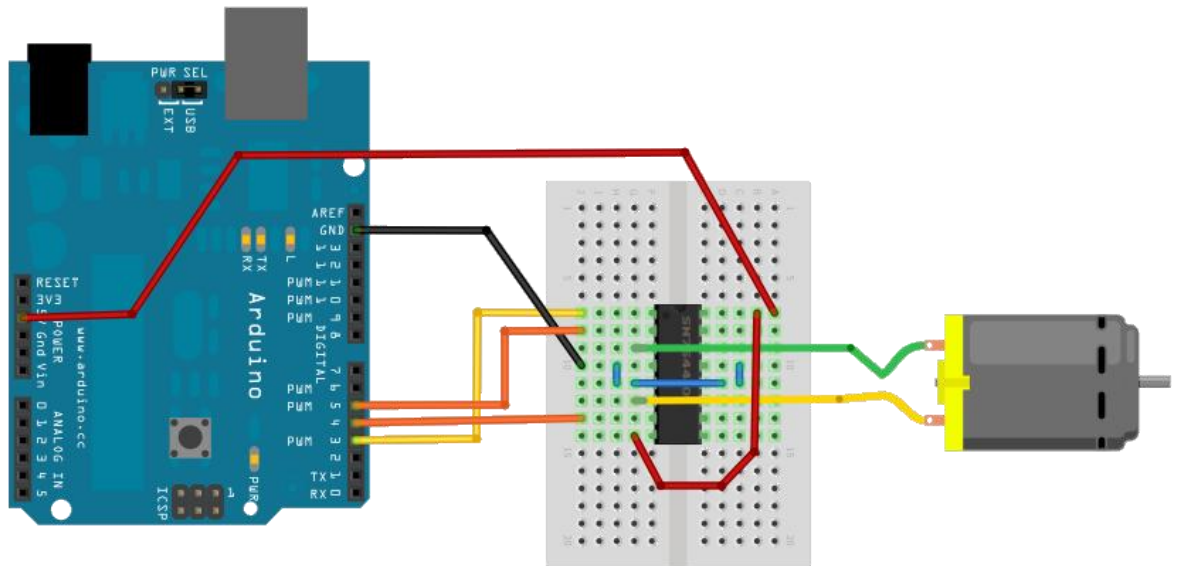
El motor se controla con un circuito integrado L293NE o SN754410



EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

Esquema del montaje en protoboard.

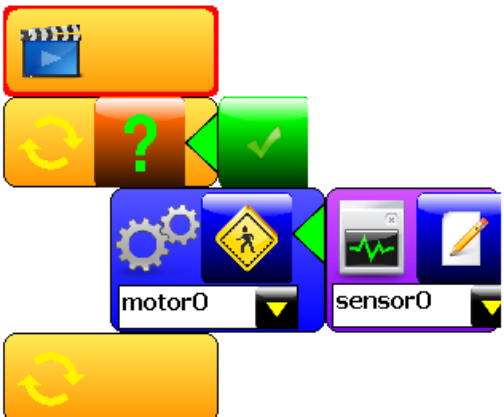


16. Control de velocidad de un motor de cc.

En este ejemplo vamos a controlar la velocidad de un motor de cc. A través del valor de un sensor colocado en la entrada **Analogica1** de la Tarjeta Arduino (**sensor0** en la aplicación Minibloq)

Una vez seleccionado el bloque de inicio **“mientras que”**, dentro de él, pondremos un bloque **“motor (setter)”** cuyo parámetro de entrada será el valor entregado por el **sensor0**.

A continuación se muestra el programa gráficamente y su listado de código.



The image shows a graphical programming interface with several blocks. At the top is a 'while' loop block (orange with a play button icon). Inside the loop is a 'motor0' block (blue with a gear icon) and a 'sensor0' block (purple with a graph icon). Below the loop is a 'repeat' block (orange with a circular arrow icon).

```

#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

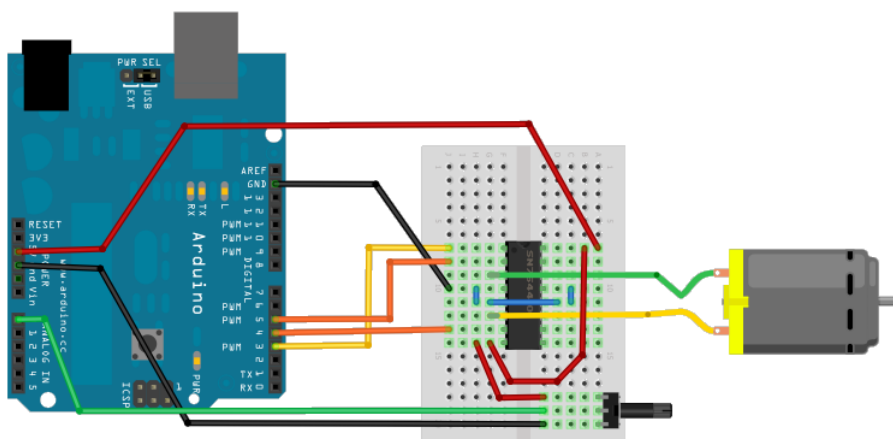
void setup()
{
  initBoard();

  while(true)
  {
    motor0.setSpeed(AnalogRead(sensor0));
  }
}

void loop()
{
}


```

En la figura vemos el montaje a realizar.



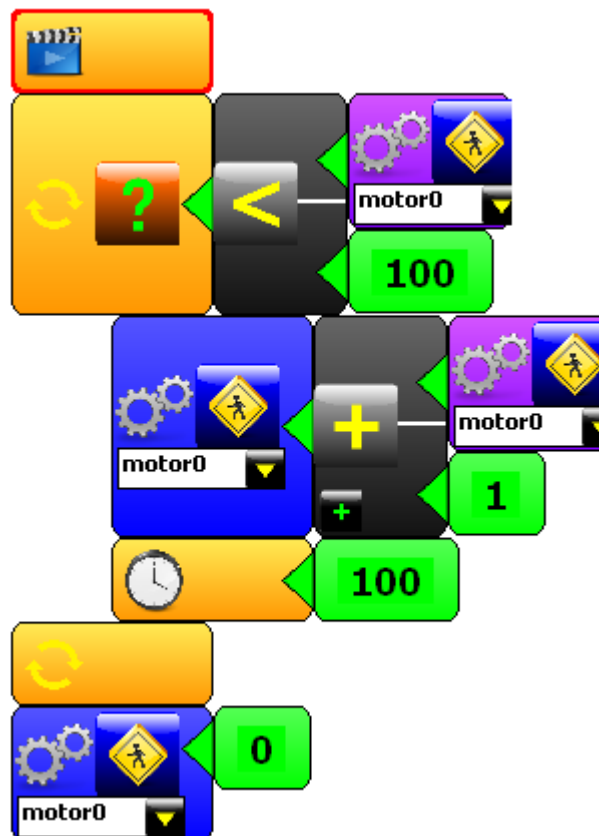
17. Aceleración de un motor de cc.

En el siguiente ejemplo vamos a mover un motor de cc. Asociado la variable motor0 de Minibloq.

1. La aplicación se implementa en torno a un bloque “**mientras que**”  que se definirá con **motor0<100** como parámetro de entrada dado que queremos que se recorra el bucle 100 veces, hasta alcanzar el valor máximo de 100 como velocidad
2. La entrada del parámetro velocidad del motor se alimenta del propio valor anterior de velocidad que se obtiene mediante el bloque “**Motor (captador)**”



Este bloque devuelve el estado de un motor eléctrico conectado a una de las salidas del motor del controlador. Se sumará 1 a este valor



3. Se ha terminado con un retardo “**retraso**” de 0.1 seg. para que esos incrementos de velocidad sean más perceptibles.
4. Finalmente, ejecutado el bucle se detiene el motor

A continuación se escribe el código generado.

```

#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

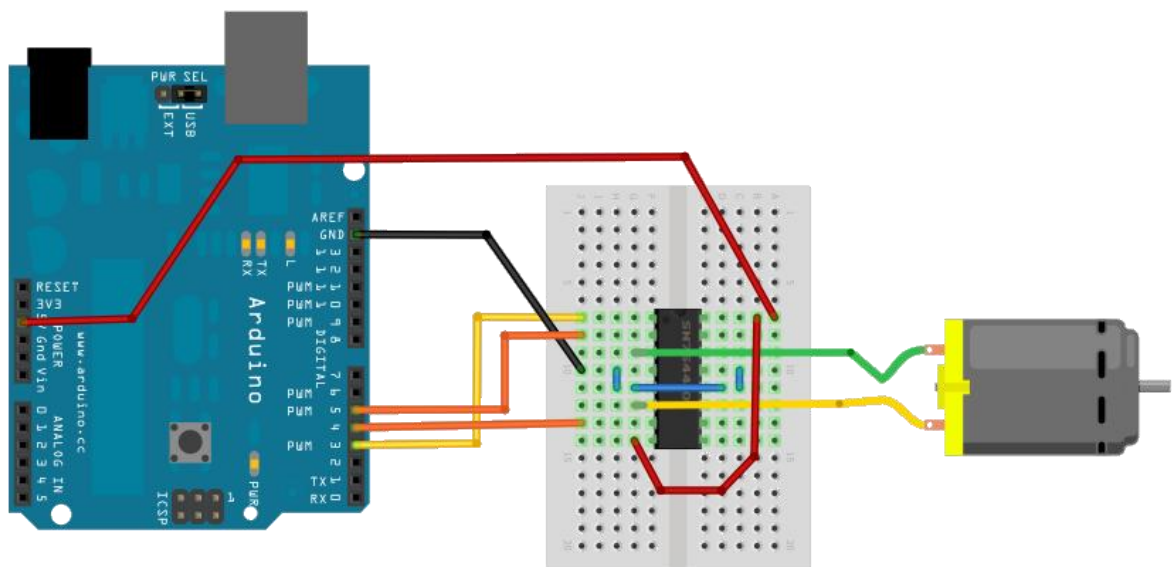
void setup()
{
  initBoard();

  while((motor0.getSpeed()<100))
  {
    motor0.setSpeed((motor0.getSpeed()+1));
    delay(100);
  }
  motor0.setSpeed(0);
}

void loop()
{
}

```

Este es el montaje en protoboard.

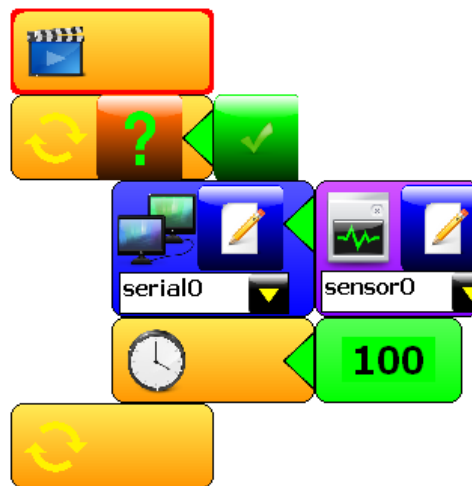


18. Lectura de un canal analógico de entrada

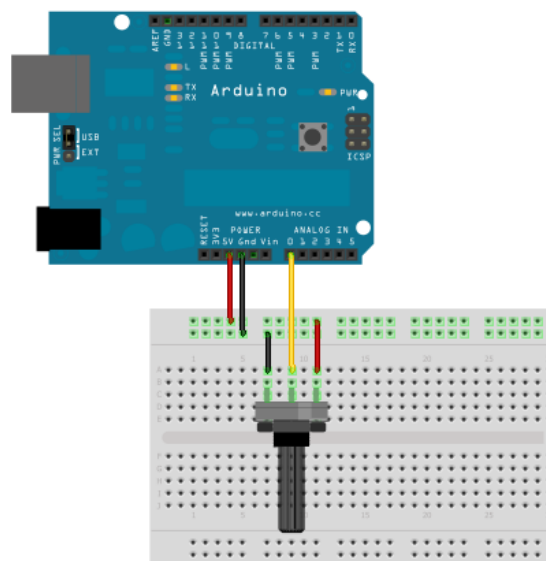
Con este ejemplo queremos leer el valor de uno de los canales de entrada analógica **Analog1** (**sensor0** de Minibloq) de la Tarjeta Arduino y mostrar su valor en la pantalla “**Terminal**” del entorno **Minobloq**.

El programa es muy sencillo. Dentro del bucle de repetición continua colocamos el bloque de escritura en el puerto al que alimentamos con el “sensor0” “**Serie (setter)**”

Se introduce un retardo de 0.1 seg. para poder visualizar el valor en el “**Terminal**” de Minibloq.



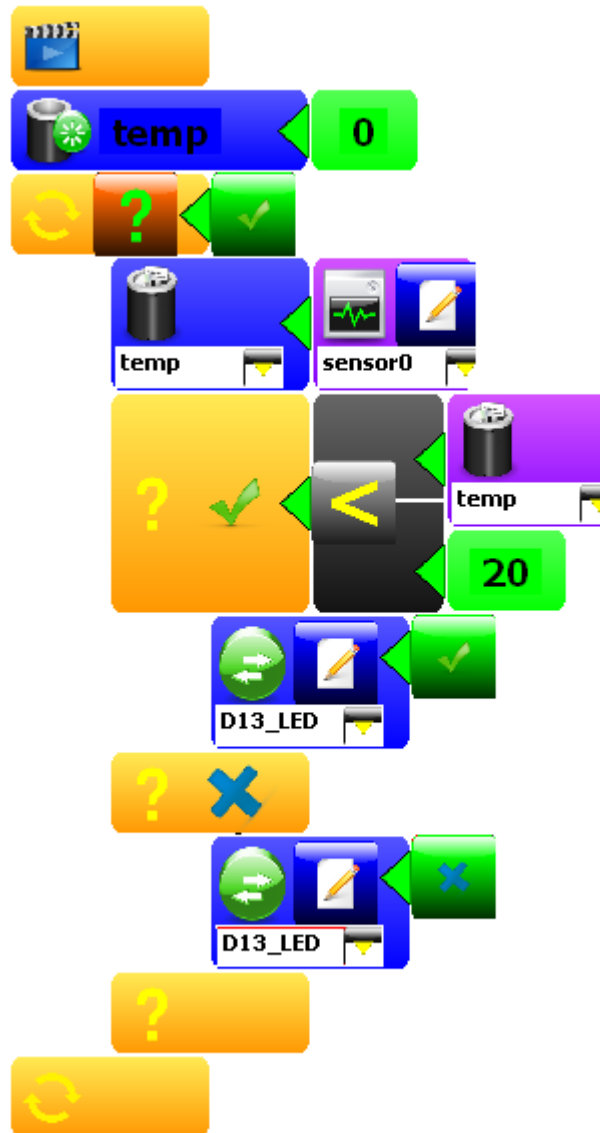
En la figura vemos el montaje en protoboard de la aplicación.



Made with  Fritzing.org

19. Simulación de un Termostato


En muchos automatismos y sistemas de control el estado de un valor digital en una salida depende del valor de una variable analógica de entrada. Este es el típico caso de un “termostato”



Proceso a seguir.

1. Esa vez definiremos la variable analógica de entrada como **temp** y le asignaremos el valor “0” “**variable (crear)**”
2. Esta variable, ya dentro del bucle de repetición continua del programa, la reasignamos al valor tomado del canal **Analog0** de Arduino equivalente a la

variable **sensor0** de Minibloq al canal analógico **sensor0 Analog0** mediante la

función “**variable (asignar)**” 

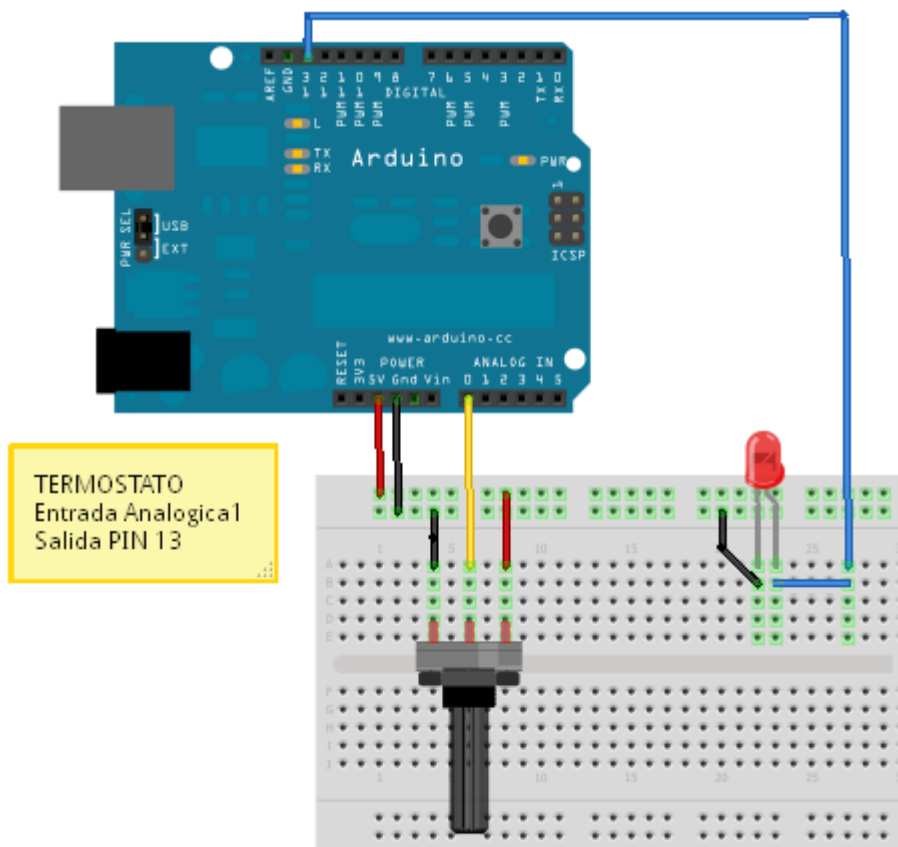
- Lo que procederá a continuación es establecer un condicional compuesto “**si ,si no**” en el que se interrogue por el valor de “**temp**”, estableciéndose que:



si **temp <20** entonces salida digital **PIN 13=true** “**digital 13 encendido**”

y en caso de no cumplirse la condición **PIN 13=false** “**digital 13 apagado**”

El montaje del circuito es el siguiente.

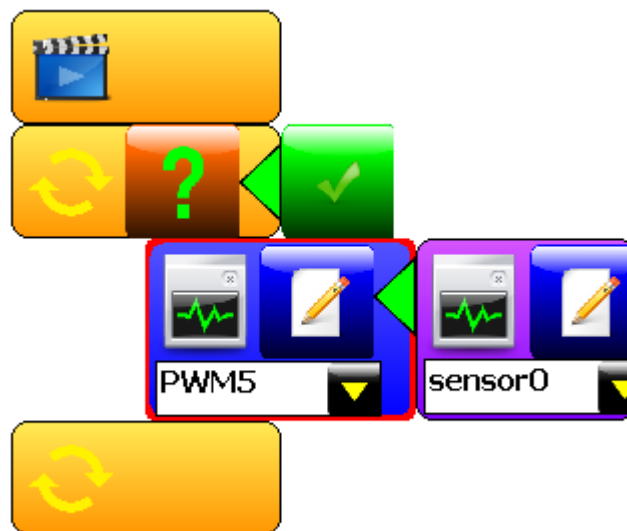


Made with  Fritzing.org

20. Traspaso de un valor analógico de entrada a una salida analógica

En el ejemplo siguiente se recogerá un valor de un canal de entrada analógica “**Analógica1**” de la tarjeta Arduino “**sensor0**” de Minibloq y se dirigirá a una de las salidas analógicas **PIN 5, PWM5** de Minibloq

El siguiente es el esquema de la aplicación



Este será el código generado

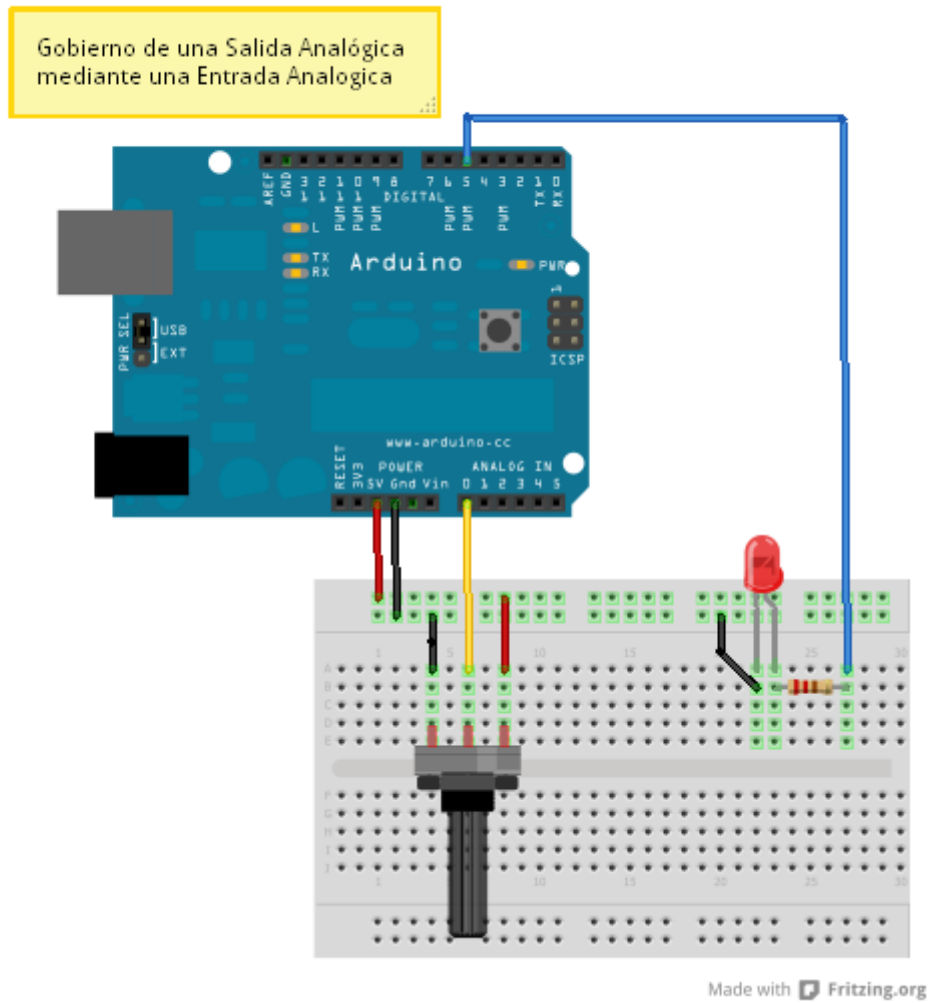
```
#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

void setup()
{
  initBoard();

  while(true)
  {
    AnalogWrite(PWM5, AnalogRead(sensor0));
  }
}

void loop()
{
}
```


El siguiente es el esquema de la aplicación



21. Termómetro con leds y sensor LM35

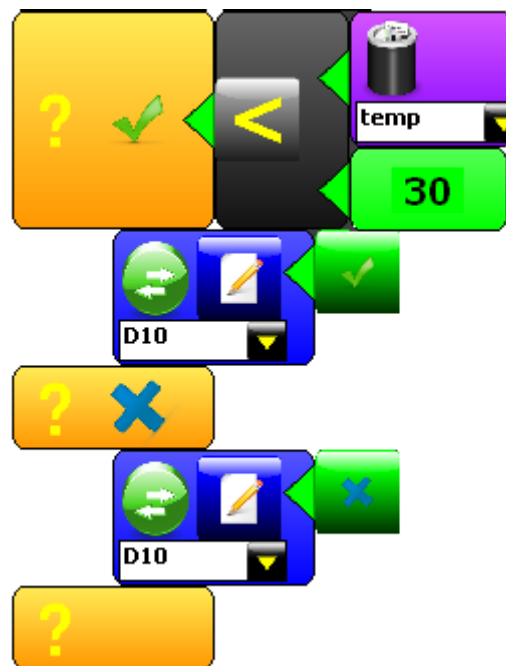
La siguiente aplicación permite la lectura de un canal de entrada analógico **Analog1** de **Arduino** y **sensor0** de Minibloq, y la posterior comparación con unos rangos de valor que permitieran el gobierno de 3 salidas digitales.

En primer lugar se definirá una variable analógica que llamaremos **“temp”** y cuyo valor se asignará al canal **Analog1** de la Tarjeta Arduino mediante el bloque de función **“variable (asignar)” (temp -> sensor0)**

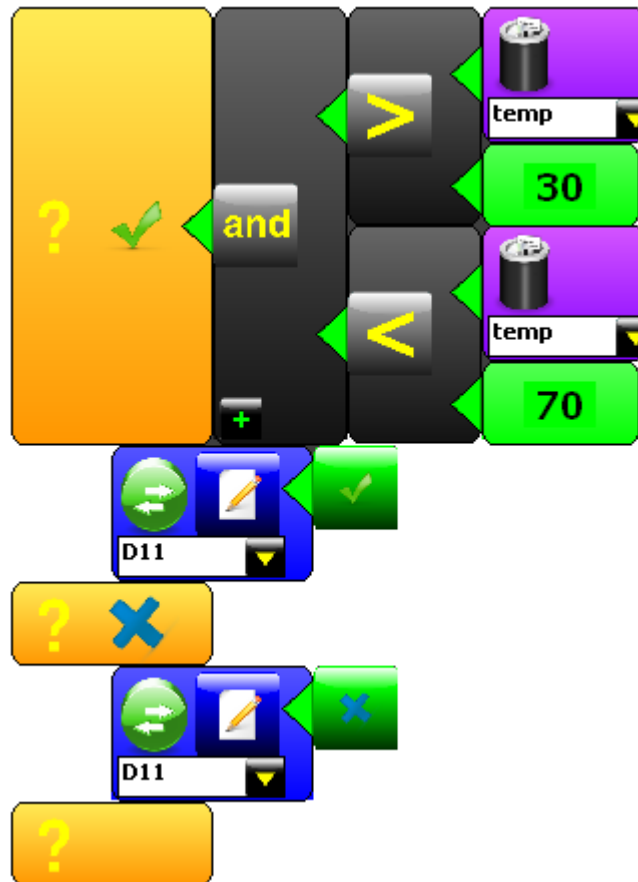
Las salidas a gobernar serán 3: **PIN 10, PIN 11 y PIN13 (D10, D11 y D13** de Minoibloq)

Las condiciones que se establecen para el gobierno de las salidas vienen dadas por los rangos que figuran en la cabecera de las funciones condicionales **“si”**

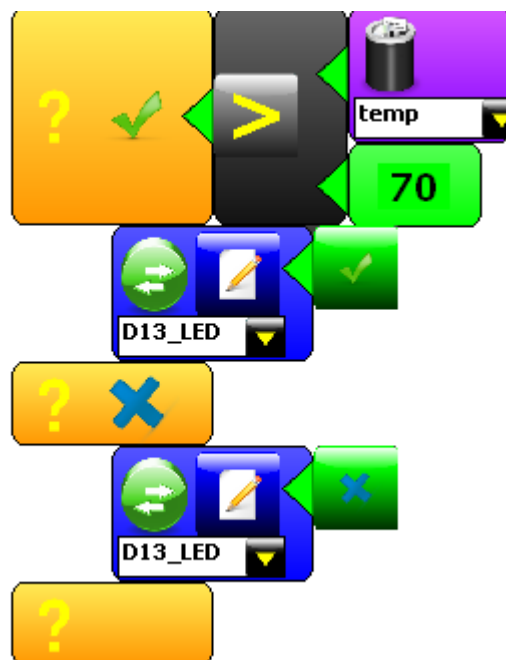
Si **temp < 30** entonces **PIN 10 =true**

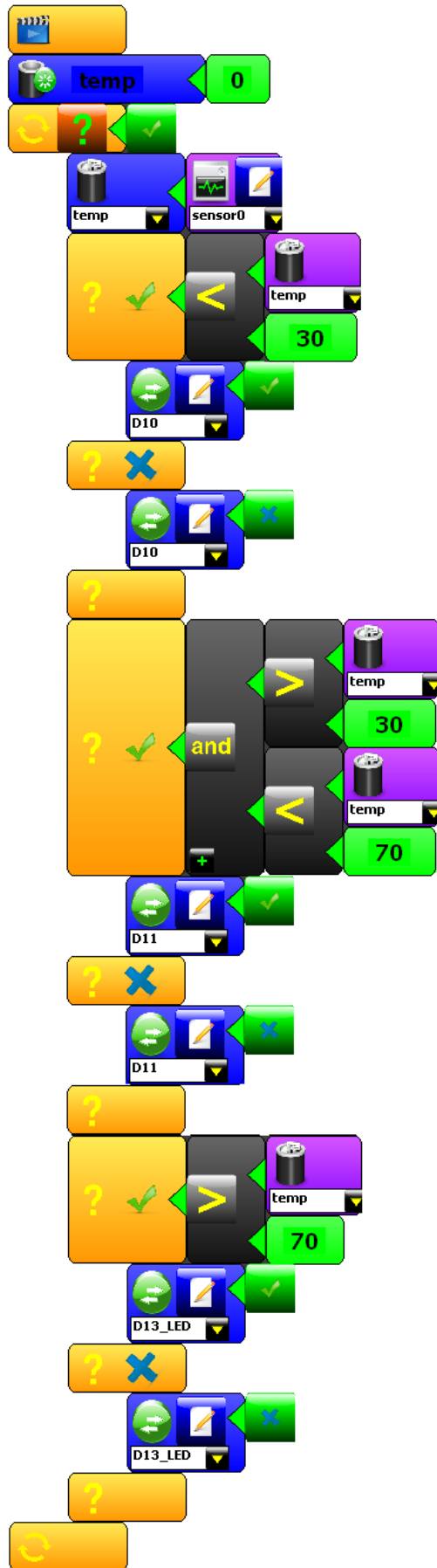


Si $70 > \text{temp} > 30$ entonces PIN 11 =true



Si $\text{temp} > 70$ entonces PIN 13 =true





El listado de código generado es el siguiente.

```
#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

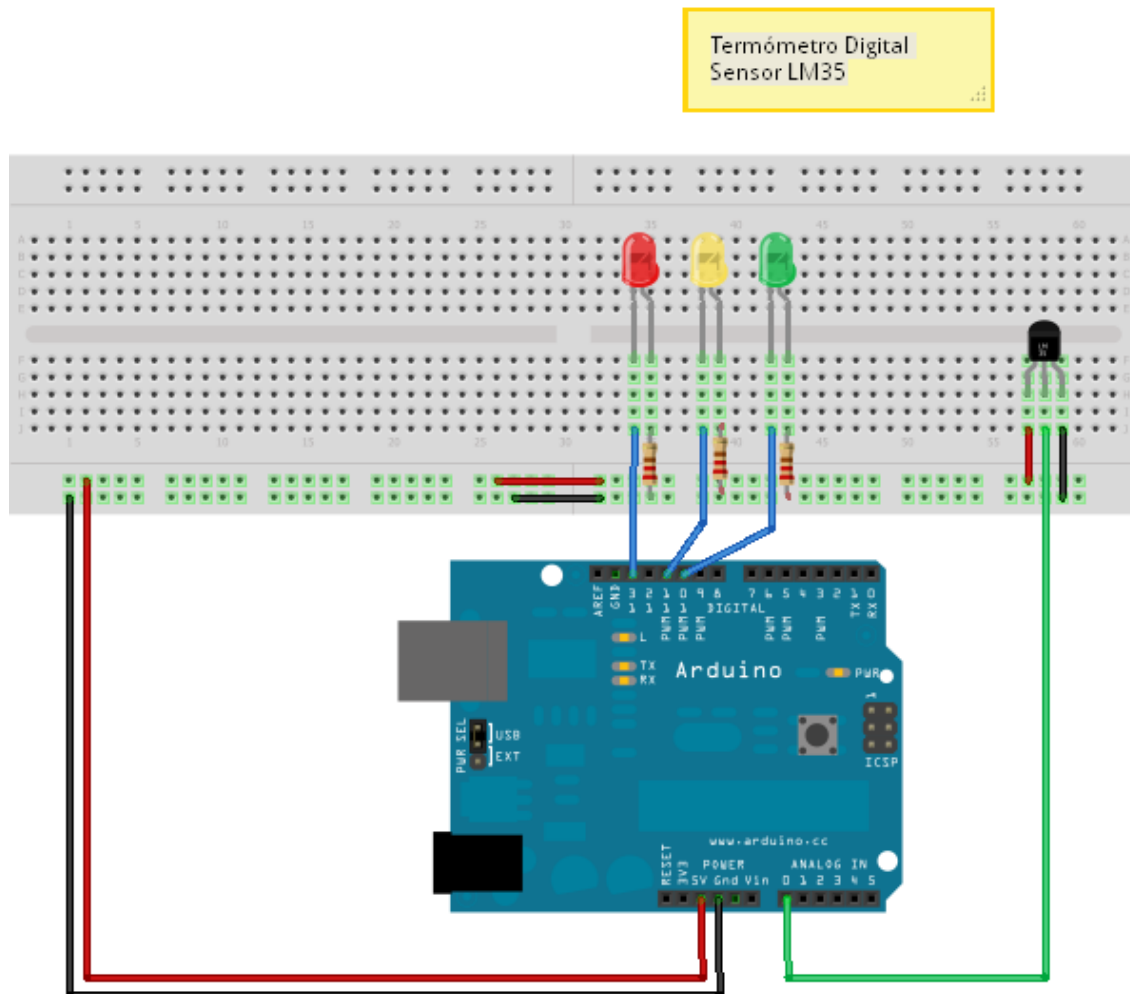
void setup()
{
  initBoard();

  float temp = 0;
  while(true)
  {
    temp = AnalogRead(sensor0);
    if((temp<30))
    {
      DigitalWrite(D10, true);
    }
    else
    {
      DigitalWrite(D10, false);
    }
    if(((temp>30)&&(temp<70)))
    {
      DigitalWrite(D11, true);
    }
    else
    {
      DigitalWrite(D11, false);
    }
    if((temp>70))
    {
    }
    else
    {
      DigitalWrite(D13_LED, true);
    }
    DigitalWrite(D13_LED, false);
  }
}

void loop()
{
}
```

A continuación se muestra el esquema de montaje en protoboard de esta aplicación.

Se ha tomado como sensor de temperatura un sensor de semiconductor LM35



22. Generación de dos notas musicales

Sabemos que con Arduino se pueden generar notas musicales. Minibloque tiene habilitada una de las salidas para sacar sonidos. PIN 12

Se trata de sacar el sonido NOTE_C4, esperar 0.3 seg. y NOTE_B4 durante 0.2 seg. de manera permanente.



Se hace uso del bloque “**Buzzer (setter)**”

```

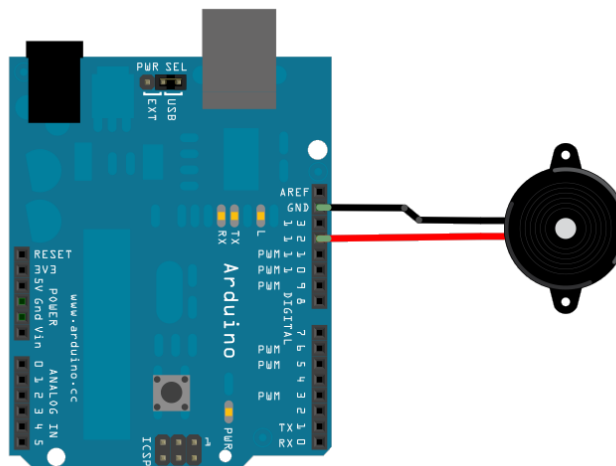
#include "stdlib.h"
#include "IRremote.h"
#include "pitches.h"
#include "Minibloq.h"

void setup()
{
  initBoard();

  while(true)
  {
    toneWithDelay(BuzzerPin, NOTE_C4, 300);
    toneWithDelay(BuzzerPin, NOTE_B4, 200);
  }
}

void loop()
{
}
        
```

El esquema eléctrico de conexionado es el siguiente.



23. ANEXO

Biblioteca de Bloques de Minibloq para Arduino


Selector de acciones

Selector de imágenes	Nombre y descripción	Parámetros
	<p>componente de inicio Este bloque indica el inicio del programa (o de componentes ").</p> <p>Nota: Este bloque no está en el selector de acciones, pero es una acción de todos modos. Minibloq añade un componente de bloque de inicio de cada nuevo componente de forma automática</p>	ninguno
	<p>mientras que Este bloque es uno ", mientras que" el ciclo de comenzar. Todos los bloques entre éste y el siguiente bloque de fin de ciclo se repite mientras la condición sigue siendo cierto.</p>	condición: bool
	<p>repetir Este bloque es uno ", mientras que" el ciclo de comenzar. Todos los bloques entre éste y el siguiente bloque de fin de ciclo se repite mientras la condición sigue siendo cierto.</p>	número de iteraciones: número
	<p>si Este bloque permite la toma de decisiones, basado en la condición lógica especificada.</p>	condición: bool
	<p>retraso Este bloque de fuerzas que el programa espere el intervalo de tiempo especificado (en milisegundos).</p>	intervalo (ms): número de

	<p>variable (crear)</p> <p>Este bloque se crea una variable y permite que lo inicie, con el fin de almacenar un número, o el resultado de una expresión, por lo que se puede utilizar en otras secciones del programa.</p>	<p>Valor inicial: número</p>
	<p>variable (asignar)</p> <p>Este bloque permite asignar un valor a la variable seleccionada.</p>	<p>valor: número</p>
	<p>IOPin (setter)</p> <p>Este bloque permite establecer el Estado de un alfiler de salida del controlador digital.</p>	<p>valor: bool</p>
	<p>AnalogWrite</p> <p>Este bloque permite el control de una salida analógica (PWM).</p>	<p>valor: número</p>
	<p>Serie (setter)</p> <p>Este bloque permite que el programa para enviar datos desde el puerto USB del controlador (o de un puerto serie, dependiendo del hardware del controlador) en el equipo.</p>	<p>valor: número</p>
	<p>Buzzer (setter)</p> <p>Este bloque permite emitir sonidos. El primer parámetro establece la frecuencia (nota) y la segunda establece la duración.</p>	<p>nota (Hz): número de</p> <p>duración (ms): número de</p>
	<p>Motor (setter)</p> <p>Este bloque permite controlar un motor eléctrico conectado a una de las salidas del motor del controlador.</p>	<p>de energía (-100 a 100): número de</p>

	<p>ServoRC (setter)</p> <p>Este bloque permite establecer la posición (en grados) de un servo R / C conectado a la placa.</p>	ángulo: el número
	<p>De número_pantalla (setter)</p> <p>Este bloque permite mostrar números -99 a 99 en la pantalla del controlador (con LED de 8x8 pantallas de matriz).</p>	valor (-99 a 99): número de
	<p>ScreenBars (setter)</p> <p>Este bloque permite mostrar cuatro números (cada número con un rango de 0 a 100) con formas de barras en la pantalla del controlador (con LED de 8x8 pantallas de matriz).</p>	<p>bar0 (0 a 100): número de</p> <p>bar1 (0 a 100): número de</p> <p>bar2 (0 a 100): número de</p> <p>Bar3 (0 a 100): número de</p>
	<p>ScreenSprite (setter)</p> <p>Este bloque permite mostrar uno de los sprites preinstalado en la pantalla del controlador (con LED de 8x8 pantallas de matriz).</p>	valor: Sprite

número (selector contextual)

Selector de imágenes	Nombre y descripción	Parámetros
	<p>variable (getter)</p> <p>Este bloque devuelve el valor almacenado en la variable seleccionada.</p>	ninguno

	<p><i>Motor (captador)</i></p> <p><i>Este bloque devuelve el estado de un motor eléctrico conectado a una de las salidas del motor del controlador.</i></p>	ninguno
	<p><i>ServoRC (captador)</i></p> <p>Este bloque devuelve la posición (en grados) de un servo R / C conectado al pin digital del controlador.</p>	ninguno
	<p><i>AnalogRead</i></p> <p>Este bloque devuelve el valor de la entrada del sensor analógico seleccionado.</p>	ninguno
	<p><i>timeStamp</i></p> <p>Este bloque devuelve el número de milisegundos desde el inicio del programa.</p>	ninguno
	<p><i>random</i></p> <p>Este bloque devuelve un número pseudoaleatorio entre 0 y 100.</p>	ninguno
	<p><i>buzzerNote (constante)</i></p> <p>Este bloque devuelve la frecuencia que pertenece a la nota seleccionada.</p>	ninguno
	<p><i>número (constante)</i></p> <p>Este bloque devuelve un número constante.</p>	ninguno
	<p><i>pi (constante)</i></p> <p>Este bloque devuelve la constante pi con un número limitado de decimales (3,14159265358979323846).</p>	ninguno






	<p>e (constante)</p> <p>Este bloque devuelve la constante e (o "el número de Euler") con un número limitado de decimales (2.7182818284590452354).</p>	ninguno
	<p>añadir</p> <p>Este bloque permite añadir números, variables y otras expresiones aritméticas.</p>	<p>valor1: número</p> <p>valor2: número</p> <p>Nota: Más operandos (valores) se puede añadir con el botón "Añadir parámetros":</p> 
	<p>sustraer</p> <p>Este bloque permite restar números, variables y otras expresiones aritméticas.</p>	<p>valor1: número</p> <p>valor2: número</p>
	<p>multiplicar</p> <p>Este bloque permite multiplicar números, variables y otras expresiones aritméticas.</p>	<p>valor1: número</p> <p>valor2: número</p> <p>Nota: Más operandos (valores) se puede añadir con el botón "Añadir parámetros":</p> 
	<p>dividir</p> <p>Este bloque permite dividir números, variables y otras expresiones aritméticas. También le permite escribir números como fracciones.</p>	<p>Numerador: Número</p> <p>Denominador: número de</p>
	<p>negativo (menos unario)</p> <p>Este bloque hace negativo el número o la expresión aritmética a su izquierda.</p>	valor: número
	<p>poder</p> <p>Este bloque permite subir el primer parámetro numérico (base) y el exponente (o potencia) dada por el parámetro</p>	<p>base: número</p> <p>exponente: el número</p>


	numérico segundos.	
	abs Este bloque devuelve el valor absoluto del número o la expresión aritmética a su izquierda.	valor: número
	módulo (resto) Este bloque devuelve el resto de la división es entre 2 parámetros.	valor1: número valor2: número
	mínimo Este bloque devuelve el mínimo de dos números, variables u otras expresiones aritméticas.	valor1: número valor2: número
	máximo Este bloque devuelve el máximo de dos números, variables u otras expresiones aritméticas.	valor1: número valor2: número
	mapa Este bloque permite mapear linealmente un valor numérico de un rango de valores (fromLow a fromHigh) a otro rango (toLow a toHigh).	valor: número fromLow: número fromHigh: número toLow: número toHigh: número
	constreñir Este bloque permite limitar un número, una variable o cualquier otra expresión aritmética a un valor entre un mínimo (a) y un máximo (b).	valor: número a: número de b: número
	seno Este bloque devuelve el seno del ángulo dado por el número o la expresión aritmética a su izquierda (en radianes).	valor: número

	<p>coseno</p> <p>Este bloque devuelve el coseno del ángulo dado por el número o la expresión aritmética a su izquierda (en radianes).</p>	<p>valor: número</p>
	<p>tangente</p> <p>Este bloque devuelve la tangente del ángulo dado por el número o la expresión aritmética a su izquierda (en radianes).</p>	<p>valor: número</p>
	<p>arcsin</p> <p>Este bloque devuelve el ángulo (en radianes) cuyo seno es el número o la expresión de la izquierda.</p>	<p>valor: número</p>
	<p>arcocoseno</p> <p>Este bloque devuelve el ángulo (en radianes) cuyo coseno es el número o la expresión de la izquierda.</p>	<p>valor: número</p>
	<p>arcotangente</p> <p>Este bloque devuelve el ángulo (en radianes) cuya tangente es el número o la expresión de la izquierda.</p>	<p>valor: número</p>
	<p>IrRemote</p> <p>Este bloque devuelve el número que se obtiene a partir de un sensor de control remoto por infrarrojos.</p>	<p>ninguno</p>
	<p>Ping</p> <p>Este bloque devuelve la distancia (en cm), medido con un sensor ultrasónico PING como Parallax))) [TM], SEN136B5B Seedstudio o similar.</p>	<p>ninguno</p>

bool (selector contextual)

Selector de imágenes	Nombre y descripción	Parámetros
	<p>IOPin (captador) Este bloque devuelve el estado del pin digital del controlador.</p>	ninguno
	<p>real (constante) Este bloque siempre devuelve "true".</p>	ninguno
	<p>falsa (constante) Este bloque siempre devuelve "false".</p>	ninguno
	<p>igual Este bloque permite comparar los números, y devuelve "true" si se mantiene la igualdad.</p>	<p>valor1: número valor2: número</p>
	<p>no es igual Este bloque permite comparar números y devuelve "true" si estos no son iguales.</p>	<p>valor1: número valor2: número</p>
	<p>inferior Este bloque permite comparar los números (o expresiones aritméticas), y devuelve "true" si el primer parámetro (valor 1) es menor que el segundo (valor 2).</p>	<p>valor1: número valor2: número</p>

	<p>mayor que</p> <p>Este bloque permite comparar los números (o expresiones aritméticas), y devuelve "true" si el primer parámetro es mayor que el segundo.</p>	<p>valor1: número</p> <p>valor2: número</p>
	<p>menor o igual</p> <p>Este bloque permite comparar los números (o expresiones aritméticas), y devuelve "true" si el primer parámetro es menor o igual a la segunda.</p>	<p>valor1: número</p> <p>valor2: número</p>
	<p>mayor o igual</p> <p>Este bloque permite comparar los números (o expresiones aritméticas), y devuelve "true" si el primer parámetro es mayor o igual a la segunda.</p>	<p>valor1: número</p> <p>valor2: número</p>
	<p>no</p> <p>Este bloque permite negar la lógica (boolean) expresiones.</p>	<p>valor: bool</p>
	<p>y</p> <p>Este bloque hace un lógicas (booleanas) "Y" y devuelve el resultado.</p>	<p>valor1: bool</p> <p>valor2: bool</p> <p>Nota: Más operandos (valores) se puede añadir con el botón "Añadir parámetros":</p> 
	<p>O</p> <p>Este bloque hace un lógicas (booleanas) "o" los retornos y el resultado.</p>	<p>valor1: bool</p> <p>valor2: bool</p> <p>Nota: Más operandos (valores) se puede añadir con el botón "Añadir parámetros":</p> 

	<p>XOR (o-exclusivo) Este bloque hace un lógicas (booleanas) "Exclusivo-O" (XOR) y devuelve el resultado.</p>	<p>valor1: bool valor2: bool</p>
---	--	--

sprite (selector contextual)

Selector de imágenes	Nombre y descripción	Parámetros
	<p>sonrisa (constante) Este bloque devuelve la "cara de la sonrisa" sprite.</p>	ninguno
	<p>cualquiera que sea (constante) Este bloque devuelve el "Sea cual sea la cara" sprite.</p>	ninguno
	<p>enojado (constante) Este bloque devuelve la "cara enojada" sprite.</p>	ninguno
	<p>heartSmall (constante) Este bloque devuelve el "corazón pequeño" sprite.</p>	ninguno
	<p>heartBig (constante) Este bloque devuelve el "corazón grande" sprite.</p>	ninguno

	<p>invader0 (constante) Este bloque devuelve el "Invader 0" sprite.</p>	ninguno
	<p>invader1 (constante) Este bloque devuelve el "Invader 1" sprite.</p>	ninguno
	<p>invader2 (constante) Este bloque devuelve el "Invader 2" sprite.</p>	ninguno
	<p>invader3 (constante) Este bloque devuelve el "Invader 3" sprite.</p>	ninguno

Este ANEXO ha sido extraído de la página Web que alberga e software Minibloq