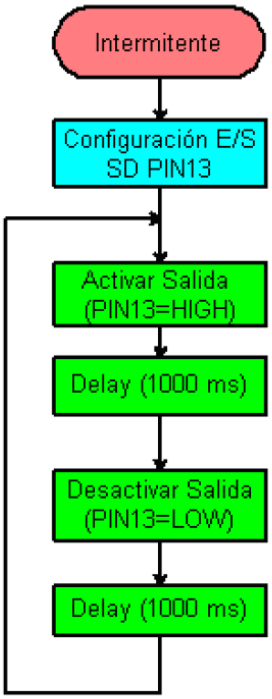
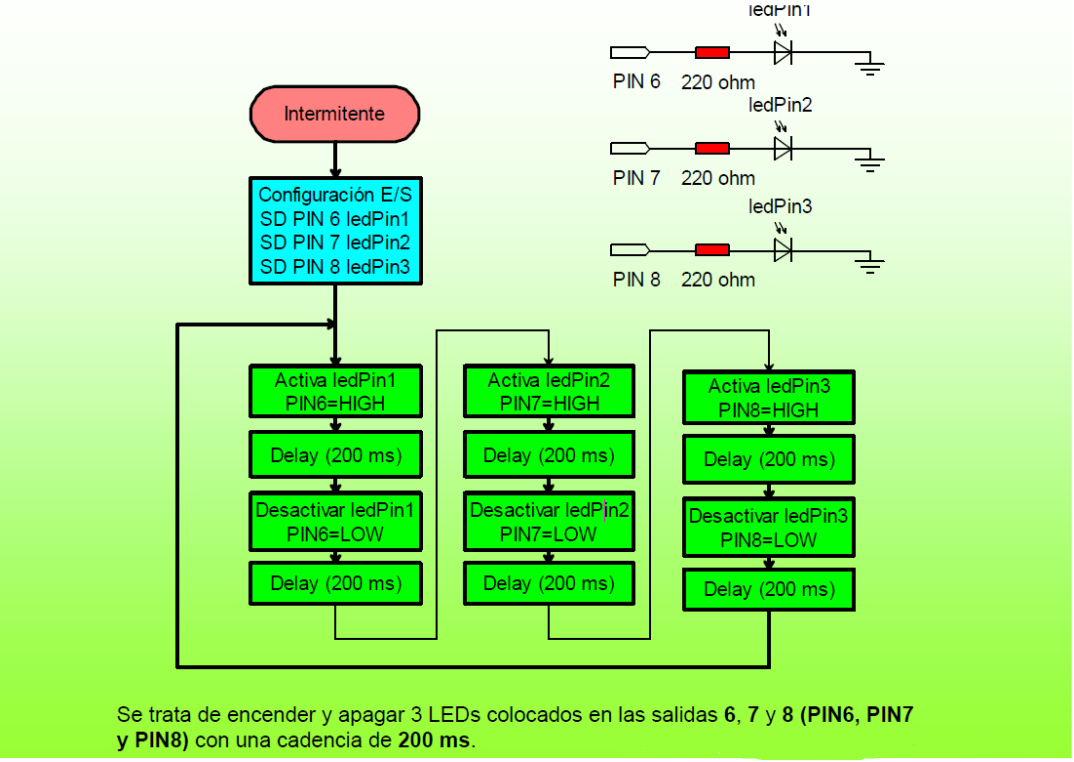
1. Intermitente

Se trata de realizar un ejercicio básico que consiste en encender y a pagar un led que conectamos en el PIN 13 de Arduino que lo configuramos como salida. El tiempo de encendido y apagado es de 1 segundo




```
/*
* Intermitente
*
* Ejemplo básico con Arduino. Encendido y apagado de un led
* con una cadencia de 1 sg. usando el PIN 13 como salida
* no es necesario usar una resistencia para el led
* la salida 13 de Arduino la lleva incorporada.
*
* http://www.arduino.cc/en/Tutorial/Blink
*/
int ledPin = 13; // Definición de la salida en el PIN 13
void setup() //Configuración
{
pinMode(ledPin, OUTPUT); // designa la salida digital al PIN 13
}

void loop() // bucle de funcionamiento
{
digitalWrite(ledPin, HIGH); // activa el LED
delay(1000); // espera 1 seg. (tiempo encendido)
digitalWrite(ledPin, LOW); // desactiva el LED
delay(1000); // espera 1 seg. (tiempo apagado)
}
```

Práctica 2: Secuencia básica de 3 LED´s. Semáforo

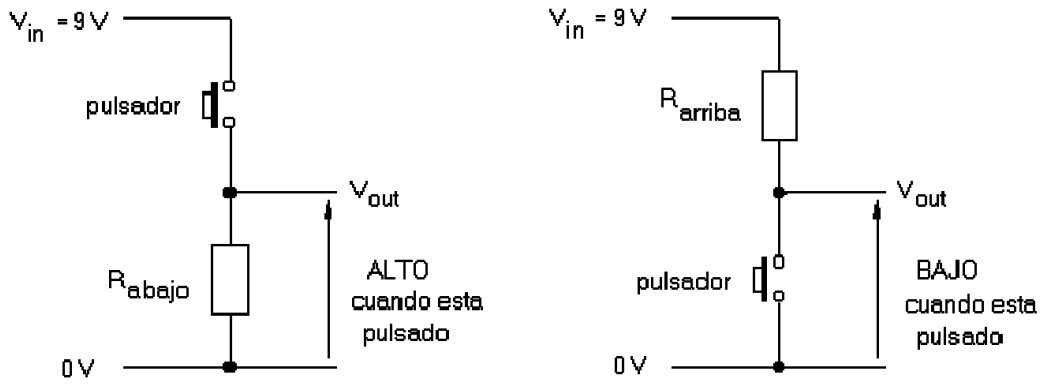


Se trata de encender y apagar 3 LEDs colocados en las salidas **6**, **7** y **8 (PIN6, PIN7 y PIN8)** con una cadencia de **200 ms**.

```
// Encedido y apagado de 3 LEDs

int ledPin1 = 6; // Define las salidas de los LED´s
int ledPin2 = 7;
int ledPin3 = 8;

void setup() // Configura las SALIDAS
 {
pinMode(ledPin1, OUTPUT); // declarar LEDs como SALIDAS
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);
digitalWrite(ledPin1, LOW); // Apaga los LEDs
digitalWrite(ledPin2, LOW);
digitalWrite(ledPin3, LOW);
}

void loop()  //Bucle de Funcionamiento
{
digitalWrite(ledPin1, HIGH); // Apaga y enciende los leds cada 200 ms
delay(200);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(200);
digitalWrite(ledPin2, LOW);
digitalWrite(ledPin3, HIGH);
delay(200);
digitalWrite(ledPin3, LOW);
}
```

3. Lectura de un pulsador

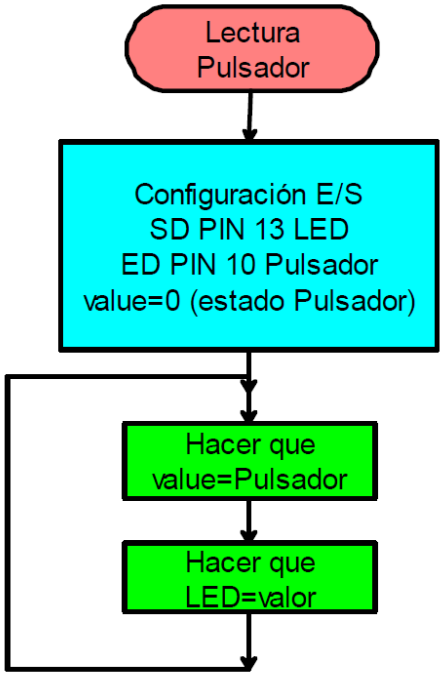
El pulsador es un componente que conecta dos puntos de un circuito cuando es presionado.
Para generar una señal de tensión con el pulsador, se necesita un divisor de tensión.
Ejemplo:

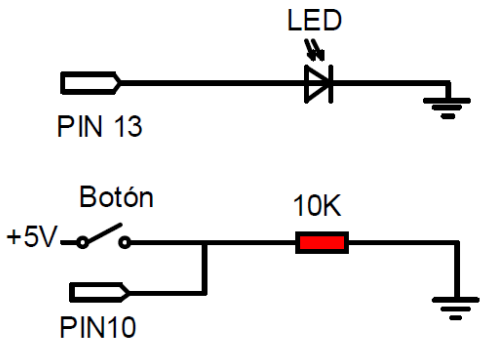


La resistencia R_{abajo}(pull-down) en el primer circuito fuerza a Vout, llegando a nivel CERO, hasta que se acciona el pulsador. Este circuito entrega una tensión alta, cuando se presiona el pulsador. Un valor para la resistencia de 10 k es adecuada.
En el segundo circuito, la resistencia R_{arriba} (pull-up) fuerza a nivel ALTO a Vout, mientras no se actúe sobre el pulsador. Al presionar el pulsador, se conecta Vout directamente con 0 V. Es decir, este circuito entrega un nivel BAJO cuando se presiona el pulsador.

Elementos necesarios:
- Un resistencia de 10K Ohmios.
- Un pulsador.
- Un diodo LED
- Cables para realizar las conexiones.

Se utiliza una resistencia pull-dowm, junto con un pulsador, para conectarla a un pin de entrada digital, y de esta forma, poder saber cuando el pulsador es presionado. Si el pulsador está presionado, el valor del pin 10 será de 0 voltios (LOW) en caso contrario será de + 5 voltios (HIGH).
En una placa protoboard debe haber una resistencia de 10K conectada entre el pin de entrada y tierra como se ve el esquema y foto inferiores.






```
/* Pulsador
* --------
*
* Detecta si el botón ha sido presionado o no
* y enciende el LED en consecuencia.
*
*
*/
int ledPin = 13; // PIN del LED
int inPin = 10; // PIN del pulsador
int value = 0; // Valor del pulsador

void setup()
{
pinMode(ledPin, OUTPUT); // Inicializa el pin 13 como salida digital
pinMode(inPin, INPUT); // Inicializa el pin 10 como entrada digital
}

void loop()
{
value = digitalRead(inPin); // Lee el valor de la entrada digital
digitalWrite(ledPin, value);
}
```

4. Condicional

if es un estamento que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves, El formato para if es el siguiente:

```
if (unaVariable ?? valor)
{
ejecutaInstrucciones;
 }
```

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se cumple (es cierta), las declaraciones dentro de los corchetes se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

Nota: Tenga en cuenta el uso especial del símbolo '=', poner dentro de if (x = 10), podría parecer que es valido pero sin embargo no lo es ya que esa expresión asigna el valor 10 a la variable x, por eso dentro de la estructura if se utilizaría X==10 que en este caso lo que hace el programa es comprobar si el valor de x es 10.. Ambas cosas son distintas por lo tanto dentro de las estructuras if, cuando se pregunte por un valor se debe poner el signo doble de igual "=="

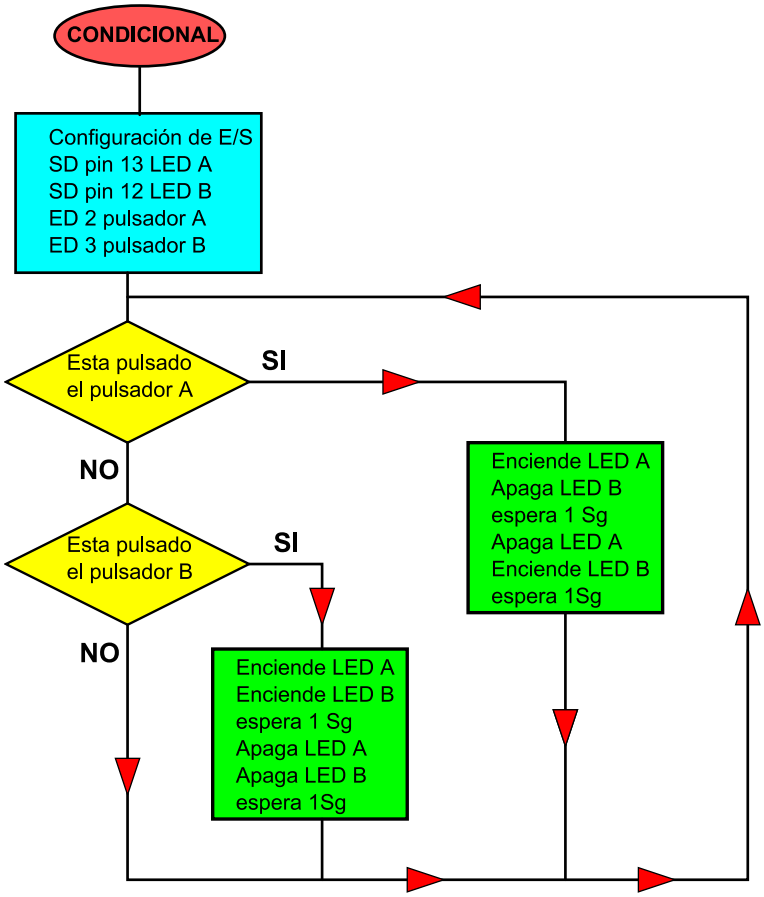
if… else viene a ser un estructura que se ejecuta en respuesta a la idea "si esto no se cumple haz esto otro". Por ejemplo, si se desea probar una entrada digital, y hacer una cosa si la entrada fue alto o hacer otra cosa si la entrada es baja, usted escribiría que de esta manera:

```
if (inputPin == HIGH) // si el valor de la entrada inputPin es alto
{
instruccionesA; //ejecuta si se cumple la condición
}
else
{
instruccionesB; //ejecuta si no se cumple la condición
}
```

Operadores condicionales

x == y // x es igual a y
x != y // x no es igual a y
x < y // x es menor que y
x > y // x es mayor que y
x <= y // x es menor o igual que y
x >= y // x es mayor o igual que y

EJEMPLO:
Si está activado el pulsador A se encienden y apagan dos led de forma alternativa
Si está activado el pulsador B se encienden y apagan dos led de forma simultanea




```
int pinledA = 13;
int pinledB = 12;
int pinpulsaA = 2;
int pinpulsaB =3;


void setup()
{
 pinMode (pinledA,OUTPUT);
 pinMode (pinledB,OUTPUT);
 pinMode (pinpulsaA,INPUT);
 pinMode (pinpulsaB,INPUT);
 }

void loop()
{
 if (digitalRead (pinpulsaA)==HIGH)
 {
  digitalWrite (pinledA,HIGH);
  digitalWrite (pinledB,LOW);
  delay(1000);
  digitalWrite (pinledA,LOW);
  digitalWrite (pinledB,HIGH);
  delay(1000);
 }

 if (digitalRead (pinpulsaB)==HIGH)
 {
  digitalWrite (pinledA,HIGH);
  digitalWrite (pinledB,HIGH);
  delay(1000);
  digitalWrite (pinledA,LOW);
  digitalWrite (pinledB,LOW);
  delay(1000);
 }

 else
 {
  digitalWrite (pinledA,LOW);
  digitalWrite (pinledB,LOW);
 }

}
```

5.- For

La declaración for se usa para repetir un bloque de sentencias encerradas entre llaves de número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración for tiene tres partes separadas por (;) vemos el ejemplo de su sintaxis:

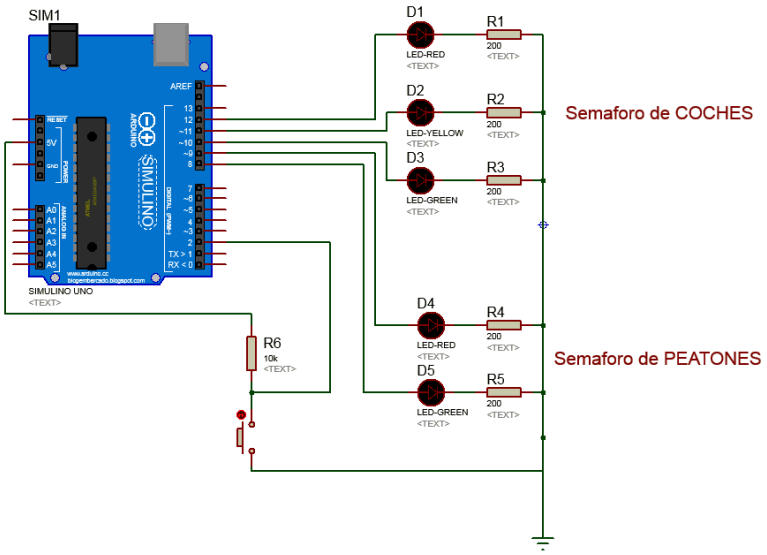
```
for (inicialización; condición; expresión)
{
ejecutaInstrucciones;
}
```

Un Ejemplo

```
for (int i=0; i<20; i++) // declara i, prueba que es menor que 20, incrementa i en 1

{
digitalWrite(13, HIGH); // envia un 1 al pin 13
delay(250); // espera ¼ seg.
digitalWrite(13, LOW); // envia un 0 al pin 13
delay(250); // espera ¼ de seg.
}
```

Práctica:
Control de un SEMAFORO con la siguiente secuencia:
1ª) Verde coches y Rojo peatones
2ª) pulsamos botón
3º) Ámbar intermitente coches y Rojo peatones
4º) Rojo coches y Verde peatones
5º) Rojo coches y Verde intermitente peatones
6º) comienzo de ciclo

Nota: Las intermitencias tanto del ámbar para los coches como la del verde de peatones se programaran utilizando la sentencia "for"

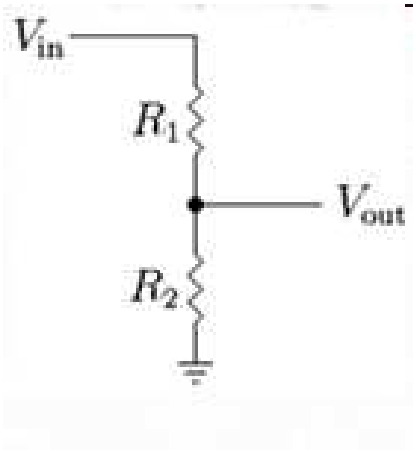


CODIGO

```
/*Control de un SEMAFORO con la siguiente secuencia:
1ª)Verde coches y Rojo peatones
2ª)pulsamos botón
3º)Ámbar intermitente coches y Rojo peatones
4º)Rojo coches y Verde peatones
5º)Rojo coches y Verde intermitente peatones
6º)comienzo de ciclo
*/

int ROJO=12;            //definimos constantes y variables
int AMBAR=11;
int VERDE=10;
int ROJO_PEATON=9;
int VERDE_PEATON=8;
int boton=2;
int t_inter=200;


void setup() {
  pinMode(ROJO, OUTPUT);
  pinMode(AMBAR, OUTPUT);
  pinMode(VERDE, OUTPUT);
  pinMode(ROJO_PEATON, OUTPUT);
  pinMode(VERDE_PEATON, OUTPUT);
  pinMode(boton,INPUT);
}

void loop()
{
  digitalWrite(ROJO, LOW);    //verde para los coches y rojo para los peatones
  digitalWrite(AMBAR, LOW);
  digitalWrite(VERDE, HIGH);
  digitalWrite(ROJO_PEATON, HIGH);
  digitalWrite(VERDE_PEATON,LOW );

  int p=0;
    p=digitalRead(boton);
  if(p==LOW)
  {
   digitalWrite(VERDE, LOW);      //la secuencia que se seguirá tras pulsar el boton

  for(int i=0;i<11;i++)
     {
       digitalWrite(AMBAR,HIGH);
       delay(t_inter);
       digitalWrite(AMBAR,LOW);
       delay(t_inter);
     }
  digitalWrite(ROJO, HIGH);
  digitalWrite(ROJO_PEATON, LOW);
  digitalWrite(VERDE_PEATON,HIGH );
  delay(5000);                    //Tiempo de paso para los peatones y de espera para los coches

  for(int i=0;i<11;i++)
    {
      digitalWrite(VERDE_PEATON,HIGH);
      delay(t_inter);
      digitalWrite(VERDE_PEATON,LOW);
      delay(t_inter);
    }
  }
}
```

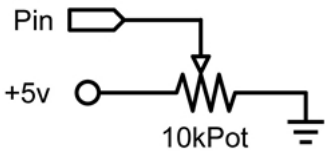
6.- Entradas analógicas

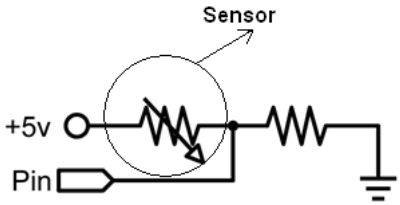
Divisor de tensión resistivo



Arduino sólo puede leer voltajes, y muchos sensores son resistivos (varían la resistencia). Por eso hay que usar circuitos de este tipo para leer el valor de los sensores.

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$





En el caso de las entradas analógicas (marcadas de A0 a A5) Arduino utiliza un convertidor analógico-digital de 10 bits, lo que nos da 1024 niveles intermedios entre 0 y 5 voltios numerados del 0 al 1023

Es posible utilizar un potenciómetro simular una entrada analógica o bien un sensor de tipo resistivo. Es posible también conectar Sensores cuya salida sea de 0 a 5v.

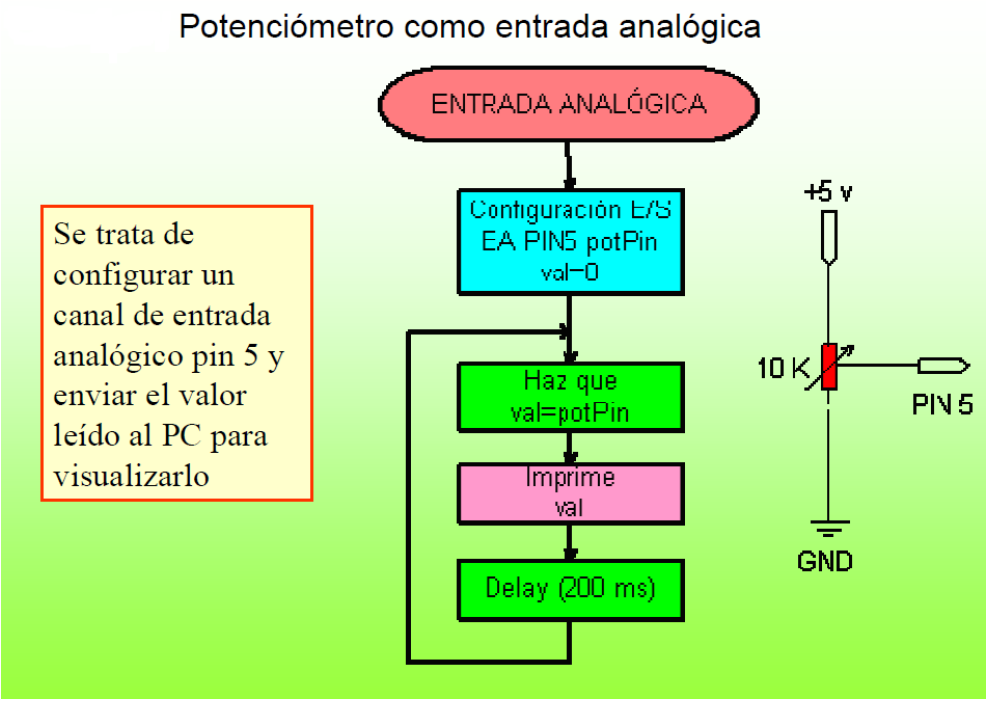
analogRead(pin)

Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. Esta instrucción sólo funciona en los pines (0-5). El rango de valor que podemos leer oscila de 0 a 1023.

valor = analogRead(pin); *// asigna a valor lo que lee en la entrada ´pin'*

Nota: Los pins analógicos (0-5) a diferencia de pines digitales, no necesitan ser declarados como INPUT u OUPUT ya que son siempre INPUT´s.

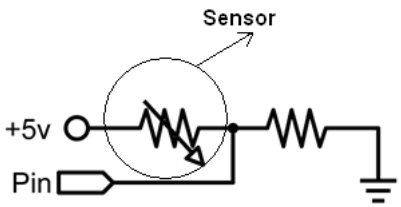
Potenciómetro como entrada analógica

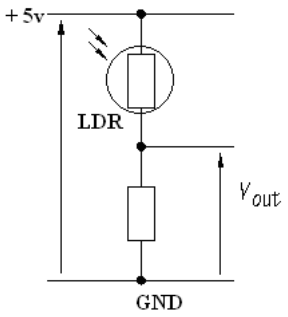


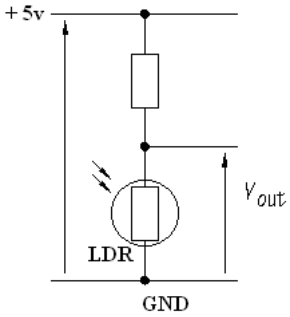

```
/* Programa Entrada Analógica
*/
int potPin = 5; // selecciona el pin de entrada para colocar el potenciómetro
int val = 0; // variable para almacenar el valor leído por la entrada analógica

void setup()
{
Serial.begin(9600);
}

void loop()
{
val = analogRead(potPin); // lee el valor del canal de ENTRADA analógica
Serial.println(val); // Envía al PC el valor analógico leído y lo muestra en pantalla

delay(100);
}
```

Midiendo Luz con Arduino

Una fotocélula (o LDR -Light-Dependent Resistor) es una resistencia que varia su intensidad según la luz que recibe. Cuanta más luz menos resistencia ejerce a la corriente.
Una LDR considera un sensor de luminosidad, se pueden usar para apagar las luces es de día o encenderlas cuando es de noche. Estas variables son de tipo analógico, como medir el sonido, la inclinación, la presión o el desplazamiento

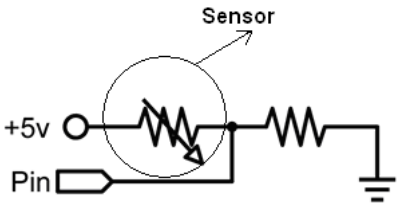


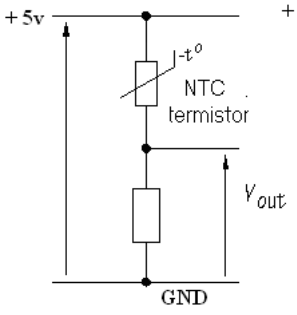


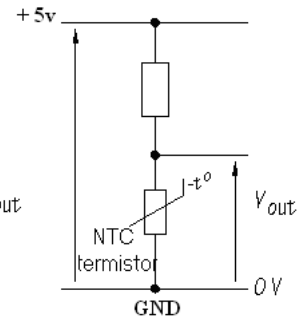


Midiendo Temperatura con Arduino

Una resistencia NTC o PTC es una resistencia variable, que varia su valor dependiendo de la temperatura ambiente.
NTC Cuanta mas temperatura menor será su resistencia.
PTC Cuanto mas temperatura mayor será la resistencia







7.- Encendido progresivo de luces según el nivel de iluminación

En este proyecto lo que queremos conseguir es que, a partir de una LDR, controlar 5 salidas de nuestro Arduino, a las que podríamos conectar líneas de iluminación para un jardín por ejemplo, de manera que según va anocheciendo, nuestra iluminación artificial va aumentando.
En este caso hemos utilizado como resistencia superior, de manera que cuanta más luz haya, más tensión tendremos a la entrada de nuestra entrada analógica pudiendo ajustar así, de una manera muy intuitiva, la tensión de referencia que ahora explicaré.
Como tercera parte del circuito, hemos colocado un potenciómetro configurado como divisor de tensión para hacer las funciones de regulación del rango de iluminación a partir del cual se activará nuestro circuito de iluminación.

¿Como hemos conseguido esto?

Primero hay que pensar un par de cosas, nuestra foto-resistencia configurada como divisor resistivo, nos va a dar 0v cuando este COMPLETAMENTE a oscuras, y +5v cuando esté COMPLETAMENTE iluminada, situaciones que pueden ser difíciles de conseguir dependiendo del entorno en el que trabajemos, y por otra parte, ese rango de 0v a 5v habrá que dividirlo en 1024 "pasos" que va a leer nuestro Arduino, pero quizás solo vayamos a trabajar en 500 o 600 pasos, desperdiciando mucha resolución.

Para solucionar esto, además de evitar cambiar el programa de Arduino cada vez que lo cambiemos de entorno, he decido usa una función que nos proporciona Arduino muy internaste, el pin "Aref"

¿Qué es el pin AREF?

Cuando Arduino toma una señal analógica y la convierte a digital en 1024 cachitos, lo hace por defecto pensando que la señal que va a recibir puede variar entre 0v y 5v lo que nos da aproximadamente un valor por cada 4,88 mV, pero nosotros podemos decirle que no, que realmente va a trabajar entre 0v y 3v, obteniendo así 1024 cachitos distribuidos entre 0v y 3v, lo que nos da un valor por cada 2,9 mV, es decir una resolución mucho mayor.

Debemos tener en cuenta que este pin lleva internamente una resitencia de 32KΩ para evitar dañar el chip, de manera que si vamos a conectar el pin mediante una resistencia fija, se va a producir un nuevo divisor de tensión, con lo que habrá que calcular la resistencia adecuada para la tensión exacta que queremos introducir mediante una sencilla fórmula:

$$Vreferencia = \frac{R\,\mathrm{int}\,erna}{R\,\mathrm{int}\,erna + \mathrm{Re}\,xterna} \times Vin$$

Donde:

Vin = Tensión que introducimos a Vref

Rinterna = Resistencia interna de Arduino de 32KΩ

Rexterna = Resistencia mediante la cual alimentamos Vref

Vrefencia = La tensión sobre que tomará referencia nuestro programa.

De manera que si por ejemplo, estamos introduciendo una tensión de +3v a través de una resistencia de 5KΩ, la tensión real de referencia en nuestro Arduino será de:

(32000/(32000+5000)) x 3 = 2,59v

¿Porqué evita cambiar el programa continuamente?

Por que variando la señal de referencia, le estamos diciendo que nos tome 1024 valores entre 0v una determinada tensión máxima, y el reparto de esos valores ya los hemos fijado equitativamente en nuestro programa para que haga una activación progresiva de las líneas de iluminación, por tanto, aunque tengamos un rango muy bajo de iluminación con el que jugar, de 1v por ejemplo, nos va a distribuir siempre proporcionalmente la activación de las salidas.

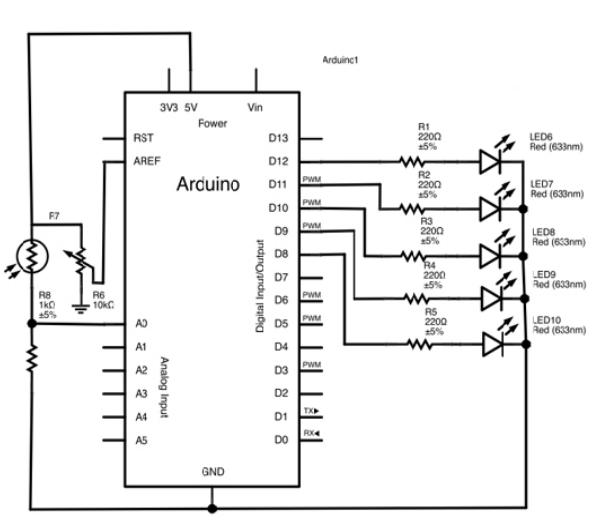
Esa tensión de referencia, será el punto de iluminación a partir del cual, si decrece la luz ambiente, empezará a decrecer los valores obtenidos en la señal analógica de entrada.

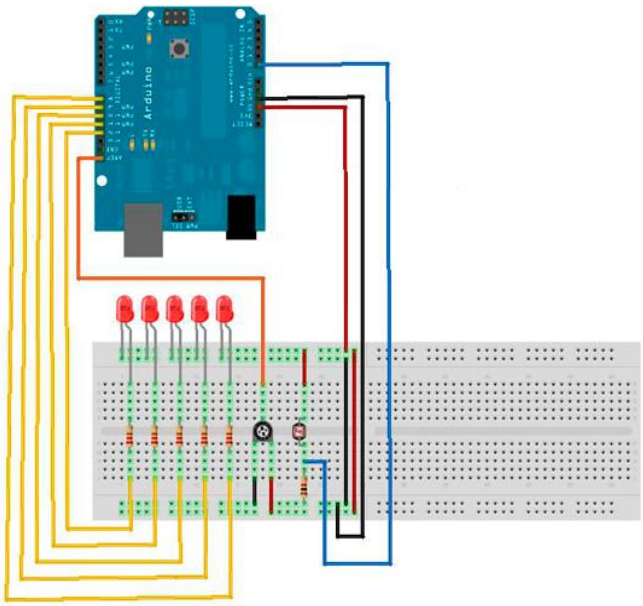
Si ponemos esa referencia muy baja, empezará a funcionar los led con menos luz ambiente que si ponemos una señal muy alta, recordemos:

Más luz = Menos Resistencia = Mayor Vout

Menos luz = Mayor resistencia = Menor Vout

Veamos el esquema eléctrico del circuito al completo:


Arduinc1
3V3 5V
Vin
Fower
RST
AREF
Arduino
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PWM
TX
RX
Digital Input/Output
Analog Input
A0
A1
A2
A3
A4
A5
GND
F7
R8
1kΩ
±5%
R6
10kΩ
R1
220Ω
±5%
R2
220Ω
±5%
R3
220Ω
±5%
R4
220Ω
±5%
R5
220Ω
±5%
LED6
Red (633nm)
LED7
Red (633nm)
LED8
Red (633nm)
LED9
Red (633nm)
LED10
Red (633nm)

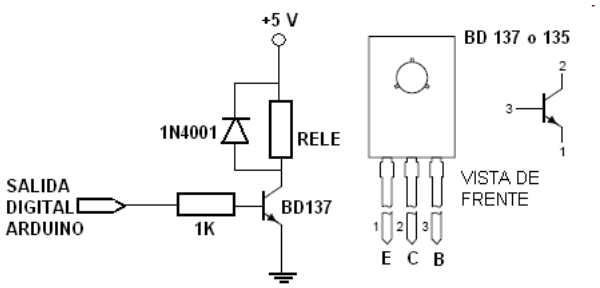


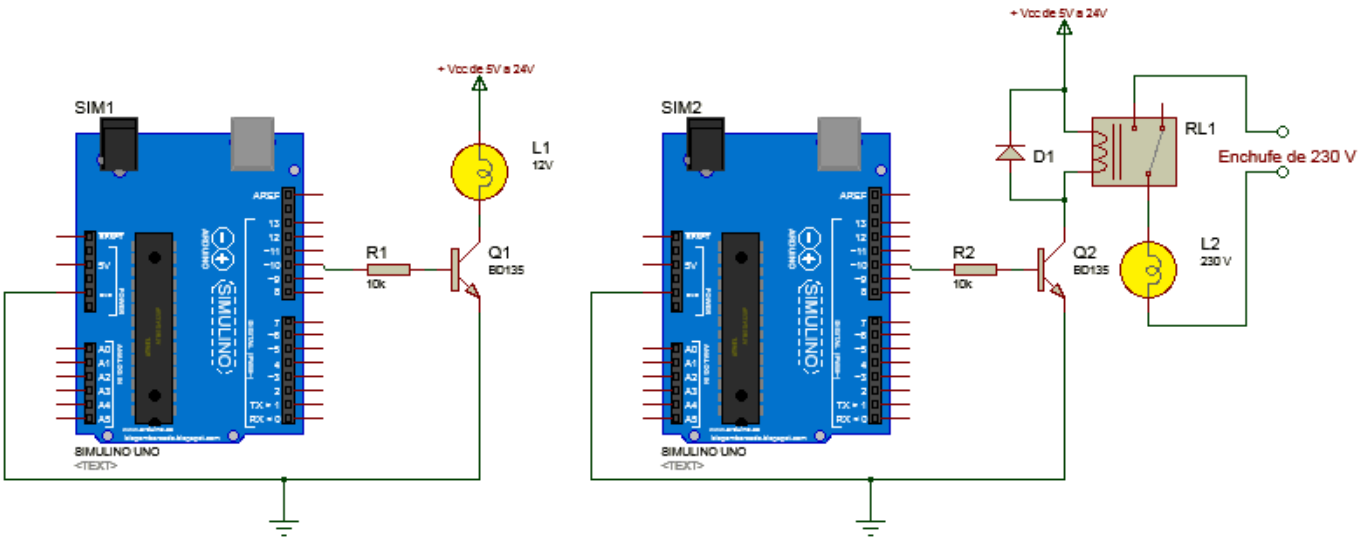

LEO EL VALOR DEL SENSOR LDR
¿ES MAYOR O IGUAL QUE 1023?
SI
NO ACTIVO NINGUNA SALIDA
NO
¿ES MAYOR O IGUAL QUE 823?
SI
ACTIVO UNA SALIDA
NO
¿ES MAYOR O IGUAL QUE 623?
SI
ACTIVO DOS SALIDAS
NO
¿ES MAYOR O IGUAL QUE 423?
SI
ACTIVO TRES SALIDAS
NO
¿ES MAYOR O IGUAL QUE 223?
SI
ACTIVO CUATRO SALIDAS
NO
ACTIVO TODAS LAS SALIDAS

```
//Conectamos una foto-resistencia a la entrada analógica para controlar cinco salidas en función de la luz ambiente.

int valorLDR = 0;  //Aquí almacenamos los datos recogidos del LDR:
int pinLed1 = 12; //Decimos que pines vamos a utilizar para LED
int pinLed2 = 11;
int pinLed3 = 10;
int pinLed4 = 9;
int pinLed5 = 8;
int pinLDR = 0; //Y que pin para la LDR

void setup() {
  pinMode(pinLed1, OUTPUT);  //Establecemos como salida los pines para LED
  pinMode(pinLed2, OUTPUT);
  pinMode(pinLed3, OUTPUT);
  pinMode(pinLed4, OUTPUT);
  pinMode(pinLed5, OUTPUT);

  analogReference(EXTERNAL);  //Le decimos que vamos a usar una referencia externa
}

void loop()
{
  valorLDR = analogRead(pinLDR);  //Guardamos el valor leido en una variable

  if(valorLDR >= 1023)  //Y comenzamos las comparaciones:
  {
    digitalWrite(pinLed1, LOW);
    digitalWrite(pinLed2, LOW);
    digitalWrite(pinLed3, LOW);
    digitalWrite(pinLed4, LOW);
    digitalWrite(pinLed5, LOW);
  }
  else if((valorLDR >= 823) & (valorLDR < 1023))
  {
    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, LOW);
    digitalWrite(pinLed3, LOW);
    digitalWrite(pinLed4, LOW);
    digitalWrite(pinLed5, LOW);
  }
  else if((valorLDR >= 623) & (valorLDR < 823))
  {
    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, HIGH);
    digitalWrite(pinLed3, LOW);
    digitalWrite(pinLed4, LOW);
    digitalWrite(pinLed5, LOW);
  }
  else if((valorLDR >= 423) & (valorLDR < 623))
  {
    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, HIGH);
    digitalWrite(pinLed3, HIGH);
    digitalWrite(pinLed4, LOW);
    digitalWrite(pinLed5, LOW);
  }
  else  if((valorLDR >= 223) & (valorLDR < 423))
  {
    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, HIGH);
    digitalWrite(pinLed3, HIGH);
    digitalWrite(pinLed4, HIGH);
    digitalWrite(pinLed5, LOW);
  }
  else
  {
    digitalWrite(pinLed1, HIGH);
    digitalWrite(pinLed2, HIGH);
    digitalWrite(pinLed3, HIGH);
    digitalWrite(pinLed4, HIGH);
    digitalWrite(pinLed5, HIGH);
  }
}
```

8.- Control de potencia

Este sencillo ejemplo enseña como encender una bombilla de 220V de corriente alterna (AC) mediante un circuito de 5V de corriente continua (DC) gobernado por Arduino. Se puede utilizar con cualquier otro circuito de 220V con un máximo de 10A (con el relé del ejemplo).






```
/* Enciende y apaga una bombilla de 220V, cada 2 segundos, mediante un
relé conectado al PIN 10 de Arduino */

int relayPin = 10; // PIN al que va conectado el relé

void setup()
{
pinMode(relayPin, OUTPUT);
}

void loop()
{
digitalWrite(relayPin, HIGH); // ENCENDIDO
delay(2000);
digitalWrite(relayPin, LOW); // APAGADO
delay(2000);
}
```

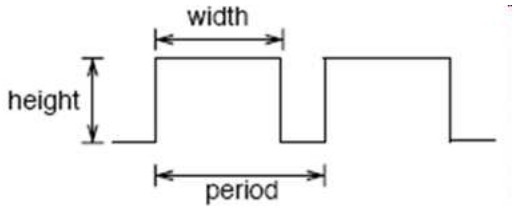
9.- Salidas analógicas

El chip Atmega de Arduino, como muchos microcontroladores, no puede generar una salida analógica, por lo que tiene que utilizar la técnica PWM (Pulse Width Modulation).

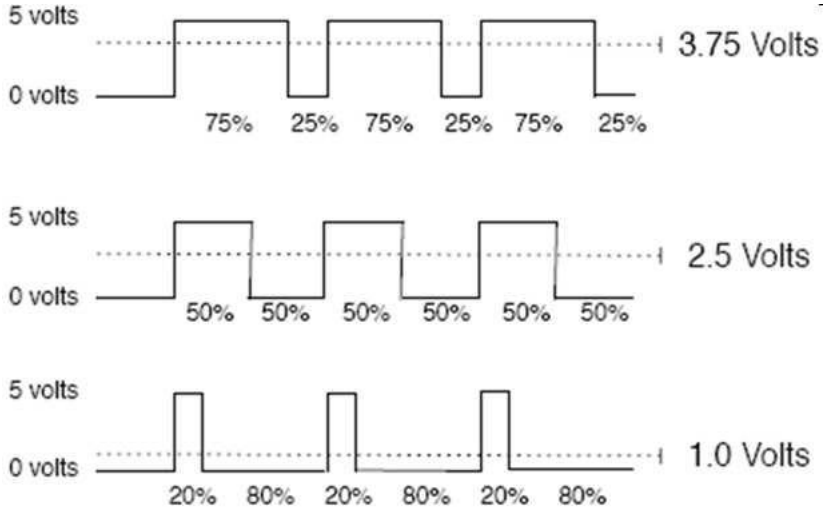
PWM consiste en simular mediante una salida digital un salida analógica.

Tres características de las señales PWM:

- Altura (heigth)
– Anchura (width)
- Periodo (period



El voltaje de salida es la media del tiempo que está a 5V con respecto del tiempo que está a 0V



Para las salidas analógicas Arduino UNO utiliza los pines 3, 5, 6, 9, 10 y 11 con una señal PWM de 8 bits, lo que nos da 256 niveles numerados del 0 al 255.

A diferencia de las entradas analógicas en las que el conversor da valores entre 0 y 1024 para generar una salida digital el rango es de 0 a 255. Donde 0 equivale a 0v. Y 255 equivale a 5 v.

Los pines analógicos, al contrario que los digitales no necesitan ser configurados como INPUT (entrada) u OUPUT (salida)

analogWrite(pin, value)

analogWrite(pin, valor); *// escribe 'valor' en el 'pin' definido como analógico*

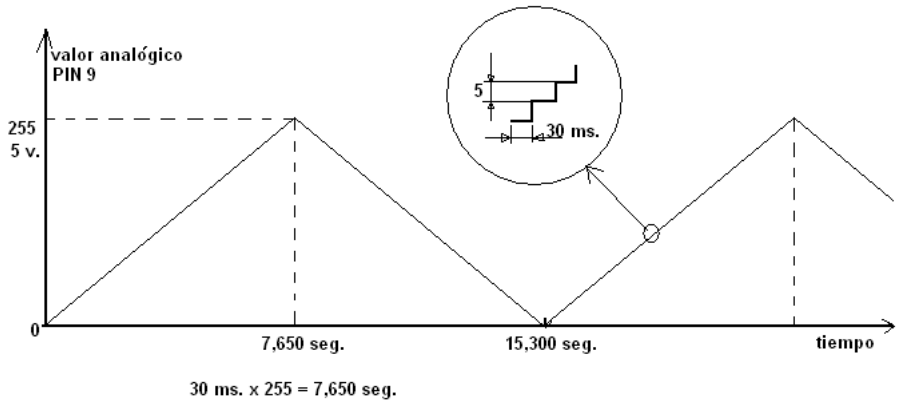
Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pin´s de Arduino marcados como “pin PWM”..
El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.

Importante: No es necesario configurar como salida las salidas PWM, por defecto Arduino considera que son salidas.

Práctica 9-a:Variando la Intensidad de un LED

Se trata de que enviemos hacia la salida 9 un valor analógico ascendente y descendente cíclicamente comprendido entre 0 y 255 en incrementos de 5.

Para la realización de este ejercicio se debe empleará una estructura de programación tipo for que realice el incremento o decremento de una variable entre 0-255 y 255-0 con un valor de retardo entre cambio de valor de 30 mseg

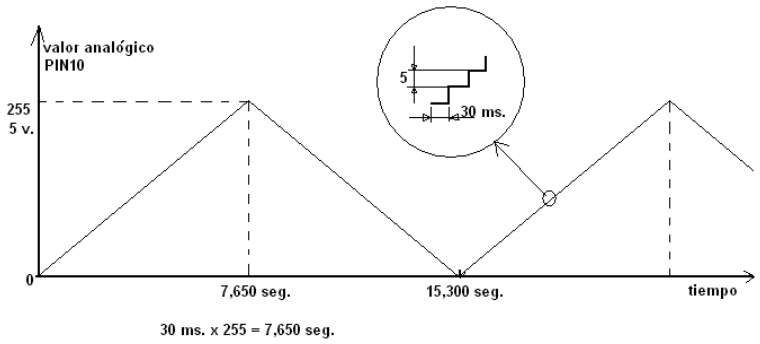


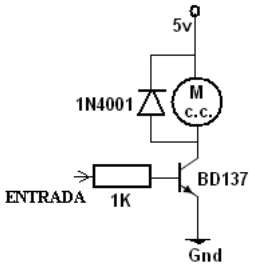
PROGRAMA

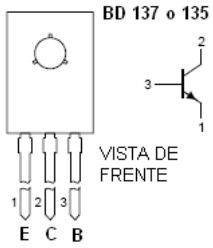
```
/*
Variando la intensidad de un LED
*/
int value = 0; // Valor a sacar por la salida analógica PIN 9
int ledpin = 9; // Salida analógicas PIN 9

void setup() { } // no es necesario

void loop()
{
for(value = 0 ; value <= 255; value+=5) // Variación de la variable se salida ente el MIN yMAX
        {
        analogWrite(ledpin, value); // Enviar valor a la salida (entre 0 y 255)
        delay(30); // Esperar 30 ms para ver el efecto de variación
        }

for(value = 255; value >=0; value-=5) // Variación de la variable de salida entre MAX y MIN
        {
        analogWrite(ledpin, value);
        delay(30);
        }
}
```

Práctica 9-b: Control de la velocidad de un motor

Con este ejemplo vamos a controlar la velocidad de un motor de cc mediante la utilización de un transistor BD137. Se trata de utilizar la posibilidad de enviar una señal de PWM a una de las salidas configurables como salidas analógicas







La tensión que sacaremos a la salida 10 (analógica tipo PWM) variara en forma de rampa ascendente y descendente de manera cíclica tal como vemos en la figura. Este efecto lo conseguimos con una estructura del tipo for:

for(valor = 0 ; valor <= 255; valor +=5) (ascendente)
for(valor = 255; valor >=0; valor -=5) (descendente)

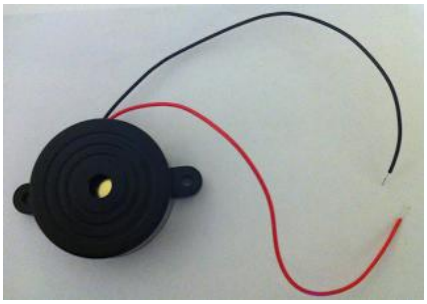
Obsérvese que los incrementos del valor de la tensión van de 5 en 5 y tenemos que considerar que 0v equivale a 0 y 5 v. equivale a 255.

PROGRAMA

```
/* Control de la velocidad de un motor
*/
int valor = 0; // variable que contiene el valor a sacar por el terminal analógico
int motor = 10; // motor conectado al PIN 10

void setup() { } // No es necesario

void loop()
{
for(valor = 0 ; valor <= 255; valor +=5)
        {
        // se genera una rampa de subida de tensión de 0 a 255 es decir de 0 a 5v
        analogWrite(motor, valor);
        delay(30); // espera 30 ms para que el efecto sea visible
        }
for(valor = 255; valor >=0; valor -=5)
        {
        // se genera una rampa de bajada de tensión de 255 a 0 es decir de 5 a 0v
        analogWrite(motor, valor);
        delay(30);
        }
}
```

10.- Sonidos con Arduino

El zumbador –buzzer o piezo speaker en inglés-, es un elemento capaz de transformar la electricidad en sonido. Puede usarse para aplicaciones como timbres, alarmas, etc. En este tutorial aprenderemos a utilizarlo y a programar unas melodías básicas con nuestro Arduino.



El buzzer. 1

¿Cómo funciona?

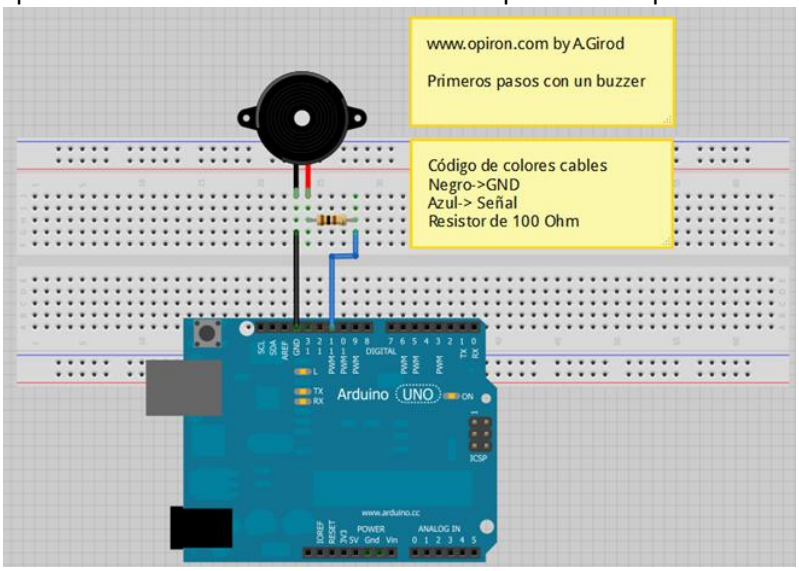
El corazón de los buzzer piezoeléctricos es un simple disco piezo, que consiste de una placa cerámica con una capa metálica. Si el disco es controlado por un circuito oscilante externo se habla de un transductor piezo eléctrico. Si el circuito oscilador está incluido en la carcasa, se le denomina zumbador piezoeléctrico.

Aplicaciones

Los generadores de sonidos piezoeléctricos son dispositivos aptos para el diseño de alarmas y controles acústicos de estrecho rango de frecuencia, por ejemplo en aparatos domésticos y de medicina.

Montaje

Primero vamos a fijarnos en el buzzer, como veréis tiene dos cables, el negro y el rojo, que conectaremos:
El negro al GND de la placa de Arduino.
El rojo entre una resistencia de 100 Ohm y una salida digital de Arduino.
Es importante recordar que la presencia de los dos colores de los cables revelan que los piezos tienen polaridad, y que los cables indican precisamente como conectar nuestro dispositivo a la placa. Haz el siguiente montaje:



Ojo con los pines que uséis, solo podemos usar los pines PWM.

Programación del Arduino

Desarrollo una función a la que llamo sonar, en la que haré que el buzzer "toque" 2 tonos diferentes, parando entre cada uno de ellos 500ms:

```
// Haciendo "cantar" a un buzzer
void setup()
{
    pinMode(11, OUTPUT);   // El pin 11 es donde tenemos conectado el Buzzer
}
void loop() {
  ring();   // con este ejemplo hacemos que el buzzer vaya tocando
            // tonos de diferentes frecuencias alternativamente
}
void ring () {
    analogWrite(11,20);    //emite 1 sonido
    delay(500);            //espera medio segundo
    analogWrite(11,120); //emite otro sonido
    delay(500);            //espera medio segundo
}
```

La función tone de Arduino

La función tone() de Arduino genera ondas PWM a una frecuencia deseada con un ciclo de trabajo del 50

La sintaxis de la función es:

- tone(pin, frecuencia)
- tone(pin, frecuencia, duración)

Esta función nos será realmente valiosa para generar 2 parámetros de las notas músicales: frecuencia (tono o altura en música) y la duración.

Haremos una programación con otro código sobre Arduino para ver cómo funciona:

Primero de todo declaramos las variables pin, frecuencia y duración:
Es decir, tocaremos una nota de 250Hz durante un segundo (1000ms).La ponemos dentro de la función setup, para que se ejecute cuando arranquemos el programa:

```
// Haciendo "cantar" a un buzzer  2

int pin=11;
int frecuencia=250;
int duracion= 1000;

 void setup() {

     tone (pin, frecuencia, duracion);
}

void loop() {
 // Nada
}
```