TP: salida analógica en Arduino

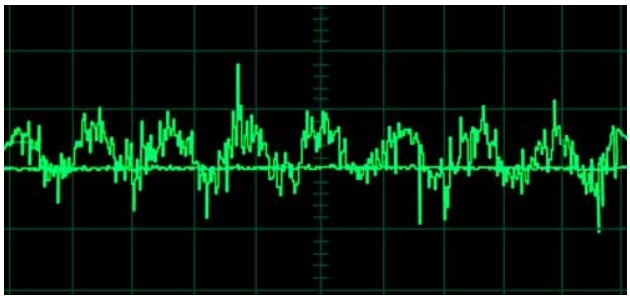
Aquí continuaremos trabajando con la plataforma Arduino, en particular nos volcaremos a elementos que servirán para interactuar con el mundo: Las salidas analógicas o también llamadas salidas PWM.

Retomado analógico vs. digital

En la clase anterior vimos cual es la diferencia entre variables analógicas y digitales, y en particular vimos como leer una señal analógica con nuestro Arduino y convertirla a un valor digital. Pregunta ¿puedo hacer la inversa? Es decir, si tengo un valor digital lo puedo convertir en algo analógico. La respuesta es NI. Vamos a poder convertir en algo casi-analógico ya que continuaremos teniendo una discretización, pero para algunas aplicaciones será lo suficientemente fina como para considerarlo analógico. Esto lo realizaremos con lo llamadas salidas PWM de nuestro Arduino, pero antes de meternos de lleno en eso debemos tener algunos conceptos más…

Concepto de señal

El concepto de señal es fundamental en muchos campos, desde la electrónica hasta la comunicación, y puede tener varias interpretaciones dependiendo del contexto. De manera general, se puede definir como una **variación de alguna magnitud física que transporta información**. Esto es muy amplio, por ejemplo desde una seña con las manos hasta un cambio de valor de tensión podremos considerarlo como señales.



Una señal es **una representación de información** que cambia con el tiempo. Esa información puede estar en forma de variaciones de voltaje, corriente, luz, sonido, temperatura, entre otros. Las señales son las que usamos para transmitir datos o para controlar dispositivos.

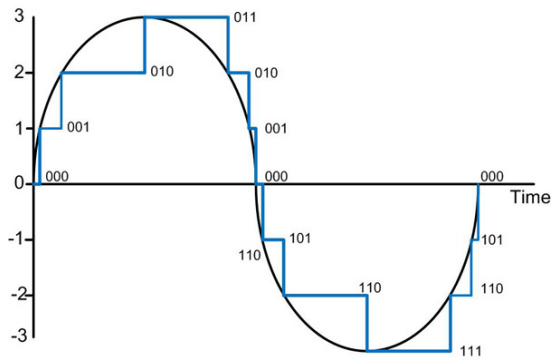
Ya algo de esto hablamos en la clase anterior donde hicimos una clasificación ellas:

Señales analógicas:

- Estas señales son **continuas** y pueden tomar un número infinito de valores en un intervalo determinado.
- Ejemplo: La señal de audio que se transmite por una radio, donde las ondas de sonido se representan como una señal de voltaje que varía suavemente con el tiempo.
- O un ejemplo común en electrónica: una señal de voltaje en un sensor de temperatura, donde la temperatura se convierte en una variación continua de voltaje.

Señales digitales:

- Son señales **discretas** que solo pueden tomar ciertos valores específicos, como "0" o "1" en el caso de la computación.
- Ejemplo: En Arduino, cuando usas el comando digitalWrite() para enviar una señal a un LED, le estás dando una señal digital: encender (1) o apagar (0) el LED.
- Las señales digitales son muy útiles porque son más fáciles de procesar y menos susceptibles a interferencias.

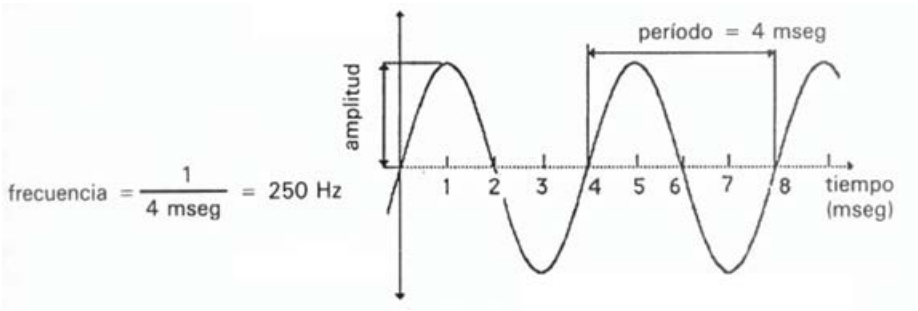


Ejemplo de una señal analógica (negro) y su versión digitalizada (azul)

Características de una señal:

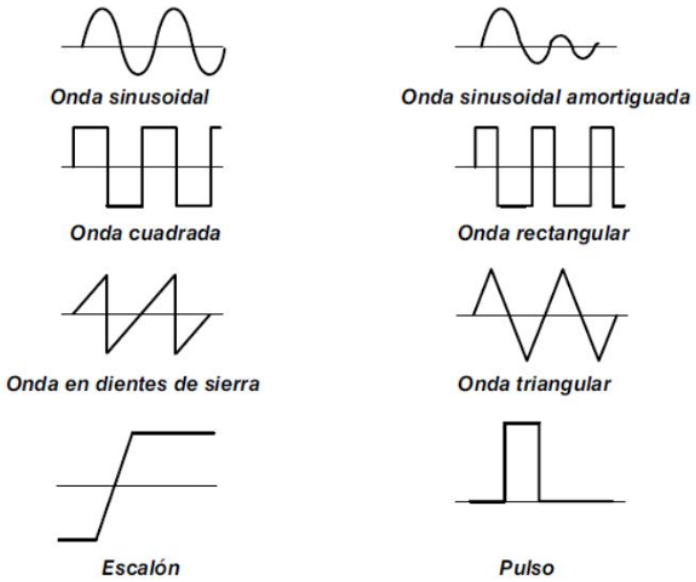
- **Amplitud**: Es la magnitud máxima de la señal, como el pico de una onda. En señales de voltaje, la amplitud puede ser el valor máximo de voltaje que alcanza.
- **Frecuencia (f)**: Es la cantidad de veces que una señal cambia o repite en un período de tiempo. Se mide en Hertz (Hz). La frecuencia es la inversa del periodo (T) de una señal, que es el tiempo que transcurre en repetirse una señal. (hay señales que se repiten en el tiempo llamadas periódicas, y otras que no, llamadas aperiódicas).

$$f = \frac{1}{T}$$



Ejemplo de una señal sinusoidal periodica, donde vemos la amplitud y periodo.

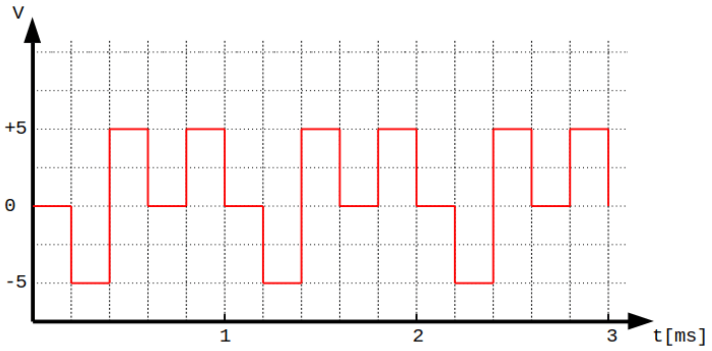
- **Forma de onda**: La forma en que cambia la señal con el tiempo. Las formas más comunes son sinusoidales (suave y continua) o cuadradas (rápidos cambios entre dos valores).



Ejemplo de diferentes formas de onda. En particular podemos ver que la Onda sinusoidal amortiguada, el escalon y el pulso son señales aperiodicas.

Actividad:

1) **La siguiente es un ejemplo de una señal digital ternaria, es decir que solo puede tener tres posibles estados. En este caso esos estados son: -5V, 0V y 5V. Al igual que en el caso anterior, la forma de onda se repite continuamente. ¿Podrías decir de cuanto es el periodo?**

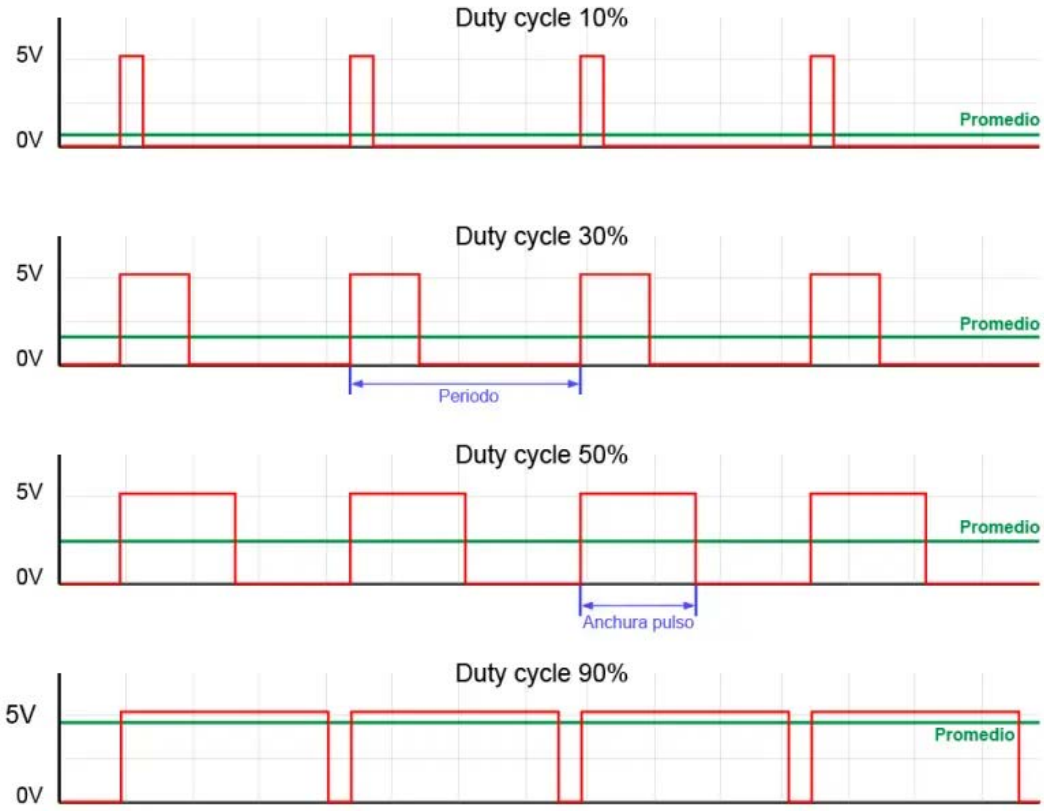


Salida PWM en Arduino (salidas analógicas)

Las salidas analógicas son algo más complicadas que las digitales (como ya pasaba con las entradas analógicas y digitales). Lo primero que tenemos que entender es que la mayoría de los automatismos (y Arduino no es una excepción) no son capaces de proporcionar una auténtica salida analógica. Ni siquiera pueden suministrar una salida analógica discretizada (es decir, a saltos) de tensión. Lo único que pueden proporcionar son salidas digitales de -Vcc o Vcc. (por ejemplo, 0V y 5V).

Para salvar esta limitación y simular una salida analógica la mayoría de los automatismos emplean un "truco", que consiste en activar una salida digital durante un tiempo y mantenerla apagada durante el resto. El promedio de la tensión de salida, a lo largo del tiempo, será igual al valor analógico deseado.

Existe más de una forma de hacer esta aproximación. Una de las más sencillas, y por ello muy empleada en automatización, es la **modulación de ancho de pulso (PWM - pulse width modulation)**. En esta modulación se mantiene constante la frecuencia (es decir, el tiempo entre disparo de pulsos), mientras que se hace variar la anchura del pulso.



Señal PWM con distintos valores de duty cycle (ciclo de trabajo).

La proporción de tiempo que está encendida la señal, respecto al total del ciclo, se denomina **Duty cycle (D).** Generalmente se expresa en tanto por ciento.

Matemáticamente a partir de la figura se puede obtener la relación entre el ciclo de trabajo (D), el periodo de la señal (T) y el valor de tensión aplicado:

$$V_{medio} = \frac{V_{cc}}{T} * \frac{D}{100}$$

$$D = 100 * \frac{V_{medio}}{V_{cc}}$$

PWM no es una señal analógica.

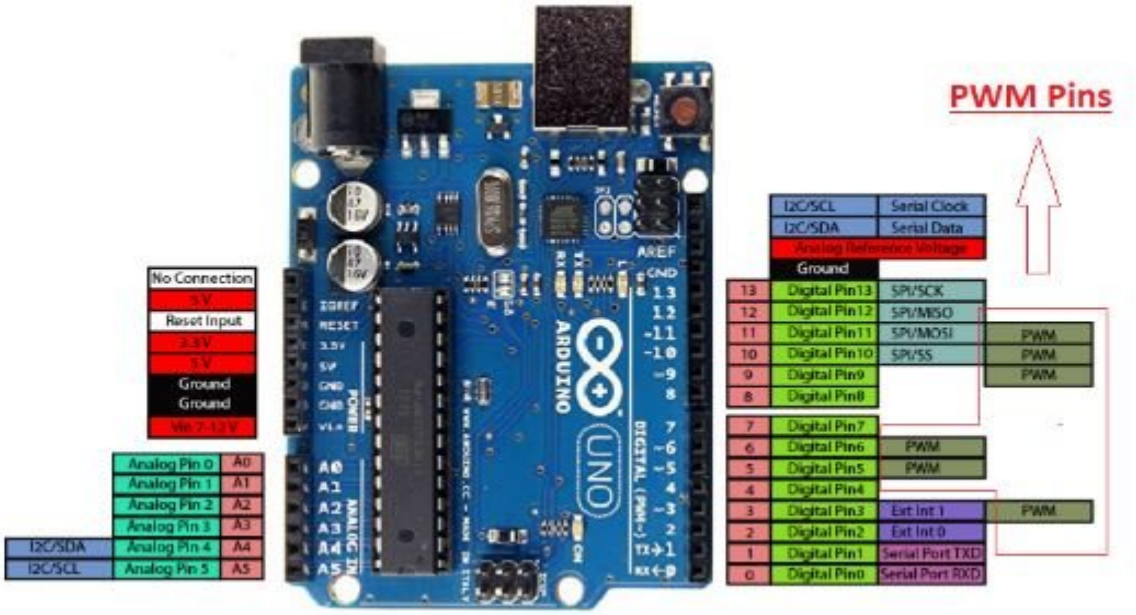
Es importante recordar en todo momento que en una salida PWM el valor de tensión realmente es Vcc. Por ejemplo, si estamos alimentando un dispositivo que necesita 3V, y usamos una señal pulsada, en realidad estaremos suministrando 5V durante un 60% del tiempo y 0V durante el 40%. Pero si el dispositivo, por ejemplo, soporta como máximo 3V, podemos dañarlo.

Una señal pulsada es suficiente para emular una señal analógica en muchas aplicaciones. Por ejemplo, podemos variar la intensidad luminosa en un LED mediante un PWM. El LED realmente se enciende y apaga varias veces por segundo, pero este parpadeo es tan rápido que el ojo no lo aprecia (nuestro ojo actúa como un filtro). El efecto global percibido es que el LED brilla con menor intensidad.

Otro ejemplo, al variar la velocidad de un motor DC con un PWM. En la mayoría de los casos la inercia del motor se encargará de que el efecto del PWM sea despreciable (el motor en este caso filtra la señal de PWM solo "viendo" el valor medio de la misma). No obstante, en función de la frecuencia podemos notar vibraciones o ruidos, en cuyo caso deberemos variar la frecuencia del PWM.

Usando las salidas PWM en Arduino

No todos los pines del Arduino podrán utilizarse como salida analógica, solo los pines indicados con una **virgulilla o tilde** (~). Que pines serán depende de la placa Arduino que utilicemos, para el caso del Arduino Uno los pines que podrán usarse con PWM serán 3, 5, 6, 9, 10, 11:



Detalle de pines del Arduino Uno.

Estos pines se encuentran conectados a un módulo especial dentro de nuestro microcontrolador llamado **DAC (Digital analog converter – conversor digital analógico)**, al igual que nos pasó con los ADC. Estos módulos discretizaran la salida (en nuestro caso en cuantos tramos podemos dividir el periodo de salida) basándose en un numero de bits, para la placa Arduino Uno el número de bits es de 8, lo que nos permitirá 256 pasos de discretización. Esto lo veremos reflejado en que el Duty Cycle que podremos imponer será un numero entre 0 (salida en 0V constante) y 255 (salida en 5V constante).

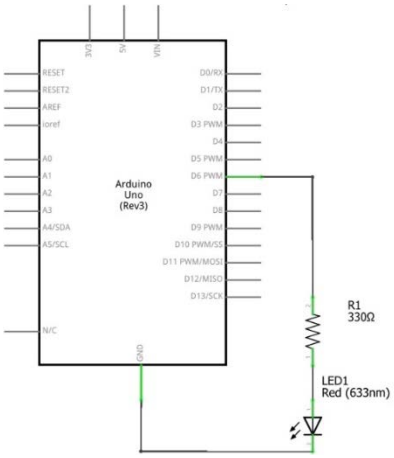
Finalmente, para generar estas señales por código utilizaremos la función:

analogWrite(pin,D);

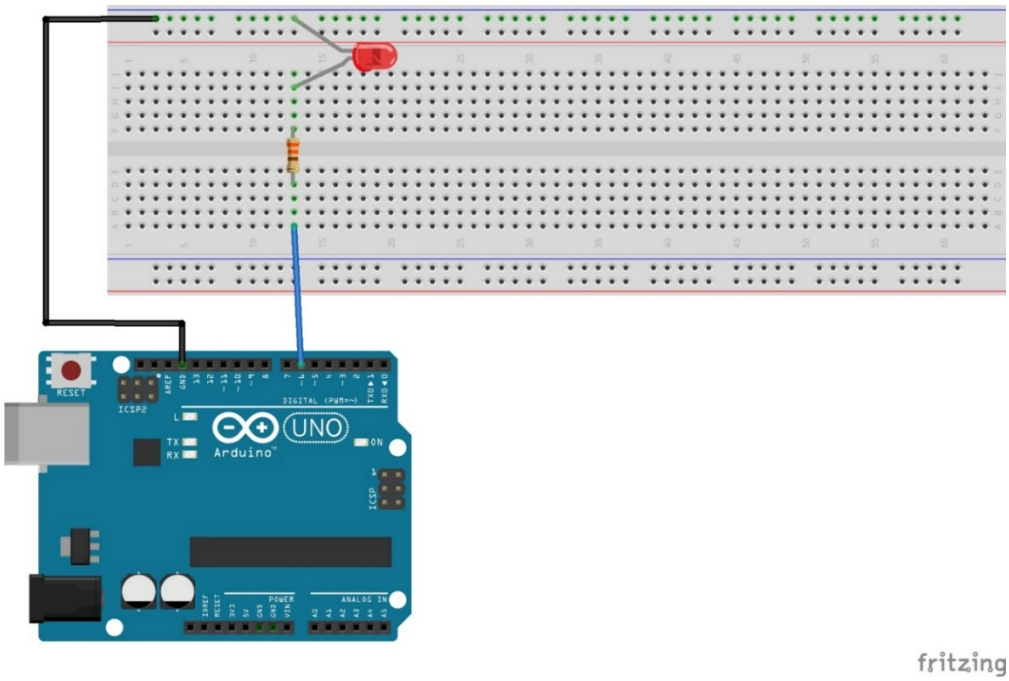
Donde pin corresponderá al pin a utilizar (siempre que este habilitado para PWM) y D al ciclo de trabajo (número entre 0 y 255).

Ejemplo 1: Cambiando la intensidad de un led.

Circuito:



Posible armado:



Programa:

```
// Ejemplo: PWM simple_1
#define led 6 // En cualquier lugar que escribamos led lo reemplazara por el numero 6
int duty_cycle = 110;       // valor del PWM
void setup()
{
}
void loop()
{
analogWrite(led,duty_cycle);
}
```

Explicación: Aquí aprovechamos a introducir una nueva línea, llamada **#define** esta servirá para generar una **etiqueta**, esto es simplemente una palabra que si la escribimos en nuestro código el IDE al compilarlo la reemplazara por el número o valor que este a la derecha del #define. Para el ejemplo, la palabra "led" escrita en cualquier parte del código será reemplazada por el número 6.

Es usual encontrar los #define al inicio de los programas, ya que permiten visualizar claramente que pines están conectados a cada actuador o sensor, o como constantes de las cuales depende el código. Es muy recomendable utilizarlos por que nos permiten modificar más fácilmente nuestros programas.

Siguiendo el programa lo que encontramos es la definición de una variable donde guardaremos el valor del duty cycle. Luego el setup está vacío, al igual que sucedía con las entradas analógicas **no es necesario declarar que usaremos las salidas PWM**. Continuando un poco más lo que encontramos en el loop, es la función analogWrite que indica el pin a utilizar y el valor de duty cycle a aplicar.

Pruebe el programa con varios valores de duty cycle y observe como cambia la intensidad del led.

Ejemplo 2: Variando la intensidad de un led cada cierto tiempo.

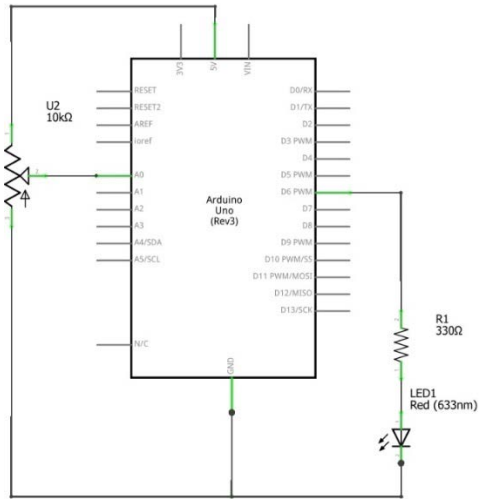
El ejemplo anterior fue interesante, pero se apreciaba poco el cambio de intensidad. ¿Y si esto lo hacemos automático? Podemos aprovechar la repetición del loop y en cada vuelta aumentar el valor del duty cycle. Pero atención que el mismo podría estar entre 0 y 255, por tanto, tendremos que agregar un condicional que para resetear el valor del duty cycle al llegar a 255. Usando el mismo circuito y el siguiente código podemos lograr este cometido:

```
// Ejemplo: variando intensidad de led
#define led 6 // En cualquier lugar que escribamos led lo reemplazara por el numero 6
int duty_cycle = 110;       // valor del PWM
void setup()
{
}
void loop()
{
analogWrite(led,duty_cycle);
duty_cycle=duty_cycle+1; // incremento de variable duty_cycle en 1
delay(10); // retardo solo para hacer mas lento el encendido del led
if(duty_cycle>255) // condicional para asegurarme no sobrepasar 255
{
duty_cycle=0;
}
}
```

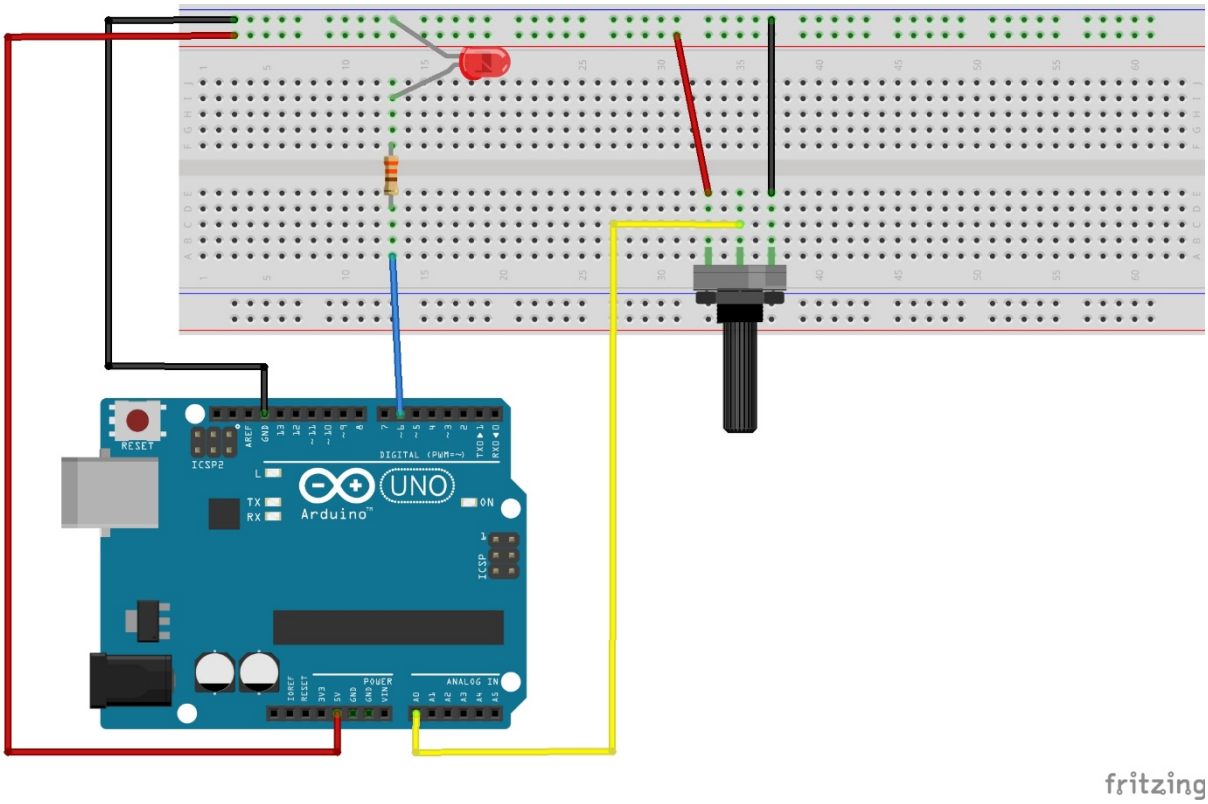
Ejemplo 3: Creando un dimmer.

Un dimmer es un regulador de intensidad, como los que podemos encontrar en algunos artefactos de iluminación. En este ejemplo utilizaremos un potenciómetro como entrada (Sí usaremos una entrada analógica) y mediante el regularemos la intensidad de un led como salida (Sí usaremos también una salida analógica).

Circuito:



Posible armado:



Programa:

```
// Ejemplo: dimmer
#define led 6 // pin salida led
#define pote A0 // pin entrada potenciometro
int lectura = 0; //variable que guadara lectura potenciometro
float aux=0; // varible usada para conversion
int duty_cycle = 0; // valor del PWM inicial
void setup()
{
}
void loop()
{
lectura=analogRead(pote); //leo estado del potenciometro

aux=float(lectura)*255/1023; // conversion entre rangos
duty_cycle=int(aux);

analogWrite(led,duty_cycle);
delay(100); // retardo}
```

Explicación: Vemos que al variar el potenciómetro varia la intensidad del led, ahora como logramos esto:

Lo primero que encontramos son los #define que definen que pines utilizaremos, luego encontramos las definiciones de las variables:

- lectura: variable de tipo **int** (que solo guardara enteros), aquí con valor inicial cero. La usaremos para recuperar la lectura del potenciómetro.
- duty_cycle: variable de tipo **int**, aquí con valor inicial cero. La usaremos para indicar el ciclo de trabajo al led.
- aux: variable de tipo **float** (este es un tipo de dato que nos permite guardar números racionales (o números con coma)).

Luego como en los ejemplos anteriores no necesitamos declarar nada dentro del setup.

Siguiendo el programa encontramos el loop, dentro de él la primer línea recupera el valor del potenciómetro en la variable lectura (recordar que este valor se encontrara entre 0 y 1023).

Las siguientes dos líneas que encontramos realizan una adaptación entre el rango de 0 a 1023 de la entrada analógica, al de 0 y 255 de la salida analógica. Para esto lo que aplicamos matemáticamente es la regla de tres simples que ya vimos. Pero al momento de ver las líneas de código nos encontramos con que algunas de las variables se encuentran entre paréntesis y con un tipo de dato delante:

Por ejemplo, **float(lectura)**, esto en programación es lo que se llama un **cast**. Lo que estamos haciendo es convertir un tipo de dato en otro. Aquí la variable lectura es de tipo int, pero la cuenta que estamos llevando adelante implica una división que lo más probable nos dé un número con coma de resultado. Sino hiciéramos este cast el Arduino interpretaría todo como un numero entero. La siguiente línea hace la conversión inversa pasa la variable aux que es de tipo float a int y la guarda en duty_cycle, esto lo hacemos por que la función analogWrite necesita que le demos un valor entero.

Finalmente, con analogWrite generamos la señal PWM de acuerdo con las variables que le pasamos.

Alternativa de código:

Una alternativa mós simple es utilizar una función propia del lenguaje Arduino llamada **map()** esta crea un escalado entre dos rangos de valores, y directamente nos devuelve un entero:

map(valor, mínimo original, máximo original, mínimo a escalar, máximo a escalar);

Por ejemplo, en el caso anterior nos quedaría:

map(lectura,0,1023,0,255);

El código completo:

```
// Ejemplo: dimmer
#define led 6 // pin salida led
#define pote A0 // pin entrada potenciometro
int lectura = 0; //variable que guadara lectura potenciometro
int duty_cycle = 0; // valor del PWM inicial
void setup()
{
}
void loop()
{
lectura=analogRead(pote); //leo estado del potenciometro
duty_cycle=map(lectura,0,1023,0,255); //conversion de rangos
analogWrite(led,duty_cycle);
delay(100); // retardo
}
```

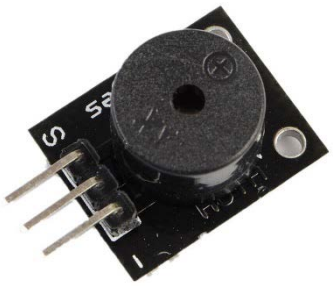
Actividades: (basándonos en los que vimos las ultimas clases)

1) **Realice el conexionado de un led a un pin PWM de una Arduino y cree un programa donde el led se prenda en forma paulatina y luego se apague en la misma forma.**
2) **Se tiene un sensor de nivel de líquido con salida analógica en el rango de 0 a 5V (0 mínimo, 5 máximo), se desea armar un circuito que encienda 3 leds a medida que el liquido aumenta de nivel. Utilice un potenciómetro para simular la salida del sensor.**

Haciendo ruido: conexión de un buzzer

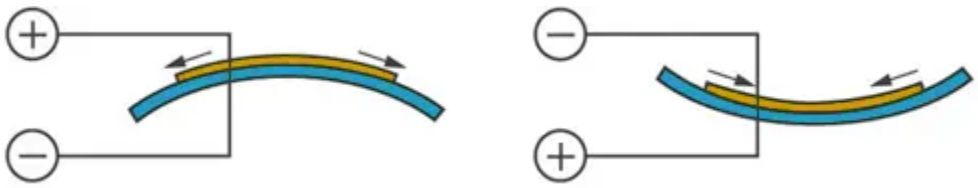
Un buzzer pasivo es un dispositivo que permite convertir una señal eléctrica en una onda de sonido. Estos dispositivos no disponen de electrónica interna, por lo que tenemos que proporcionar una señal eléctrica para conseguir el sonido deseado.

Los buzzer pasivos nos permiten variar el tono emitido modificando la señal que aplicamos al altavoz, lo que nos permite generar melodías. Y como ya podrás imaginar si aplicamos una señal PWM a este dispositivo obtendremos distintos tonos (pero todos con la misma amplitud – volumen).



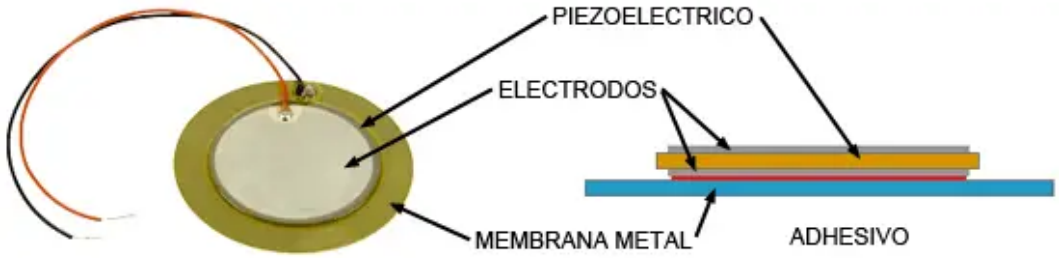
Modulo buzzer

¿Cómo funciona un buzzer? Técnicamente los buzzers son transductores electroacústicos, es decir, dispositivos que convierten señales eléctricas en sonido. Los buzzer son transductores piezoeléctricos. Los materiales piezoeléctricos tienen la propiedad especial de variar su volumen al ser atravesados por corrientes eléctricas.



Deflexión material piezoeléctrico al aplicar una tensión

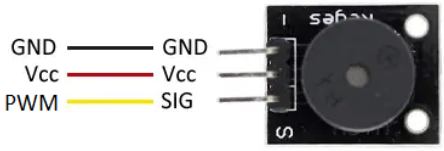
Un buzzer aprovecha este fenómeno para hacer vibrar una membrana al atravesar el material piezoeléctrico con una señal eléctrica.



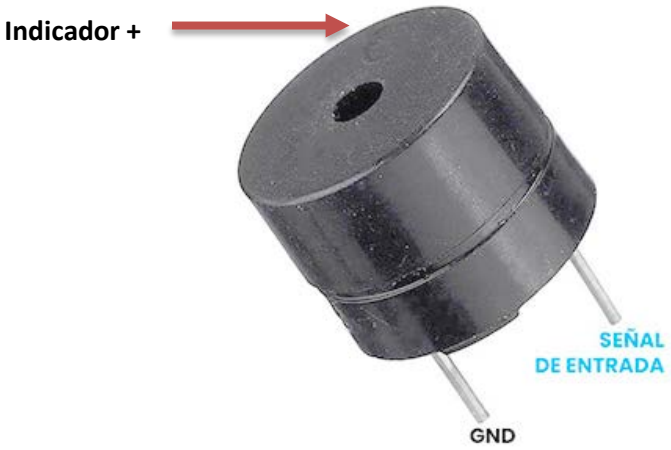
Partes internas de un buzzer

Los buzzer son dispositivos pequeños y compactos, con alta durabilidad, y bajo consumo eléctrico. Por contra, la calidad de sonido es reducida.

El esquema de conexión visto desde el componente sería el siguiente:



Nota: en el caso que solo tengamos el componente (no en modulo como en la imagen anterior), el mismo solo tiene dos patas de conexión. Una que se conectara a GND y la otra directamente a la señal que generemos desde nuestro Arduino (esta pata esta indicada con un símbolo +)



A continuación, tenemos un código de ejemplo que mediante una señal PWM podemos escuchar un “beep” a un tono especifico:

```
// Ejemplo: buzzer simple
#define buzzer 6 // pin buzzer
#define pausa 100 // pausa del beep
int duty_cycle = 40; // valor del PWM
void setup()
{
}
void loop()
{
analogWrite(6, duty_cycle);   //Encendido del buzzer a un tono determinado
delay(pausa);              // Espera
analogWrite(6, 0);          // Apaga
delay(pausa);
}
```

Actividades:

1) **Modifique el programa de ejemplo del buzzer de forma que en cada beep el ciclo de trabajo aumente en 30, y al llegar a 255 vuelva a cero. Pruebelo ¿nota los distintos tonos?**

2) **La bomba: Con un pulsador y un buzzer arme un circuito que simule el conteo de una bomba. Cuando se toque el pulsador la bomba activara su secuencia, luego el buzzer sonara cada vez más rápido hasta que quede en un sonido continuo. Luego el sistema se reseteará.**