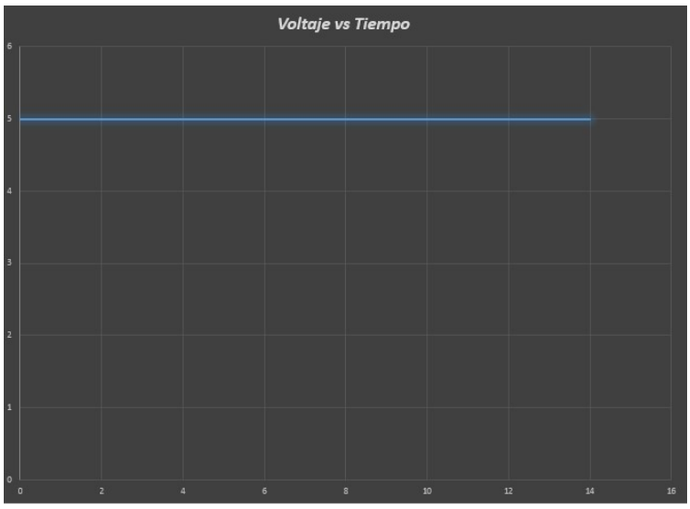
Arduino PWM: Modulación por ancho de pulsos *(Versión 17-6-19)*

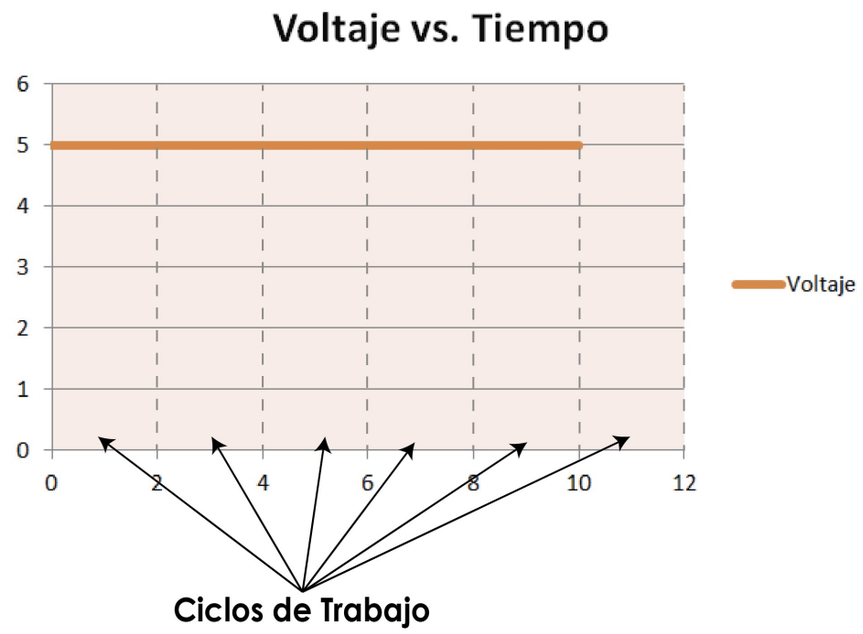
Por Antony García González - Panama Hitek

Antes de explicar lo que es el PWM (Pulse width modulation, modulación por ancho de pulsos), vamos a ver primero como se ve una gráfica de **voltaje vs tiempo**.

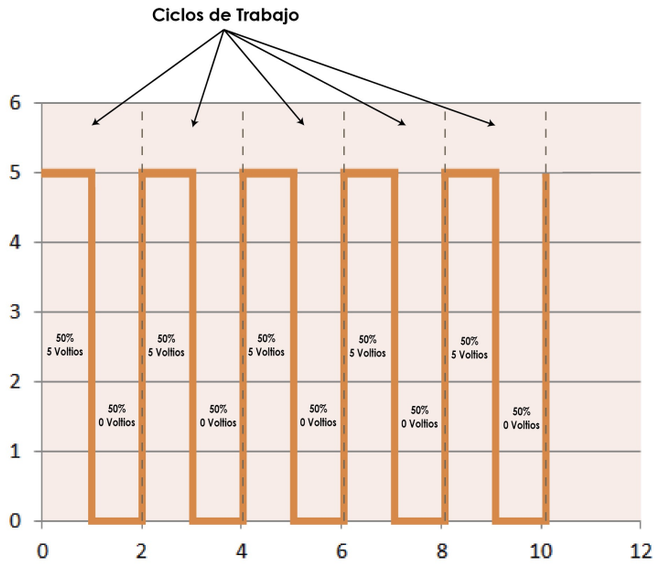
Cuando se trabaja con corriente directa el signo del voltaje permanece invariable en el tiempo, es decir, no cambia de signo como es el caso de la corriente alterna. O se mantiene positivo o se mantiene negativo. Cuando el voltaje directo se ha regulado, obtenemos un valor invariable en el tiempo. En el caso de Arduino el voltaje es 5 voltios en los pines digitales. Cuando programamos un Output, la gráfica de **voltaje vs tiempo** luce así:



El voltaje permanece constante. No varía en el tiempo. Sin embargo, es posible dividir esa onda en ciclos de trabajo. Los ciclos de trabajo duran un tiempo determinado, por lo que una onda de voltaje puede ser dividida en cuadrados.



El dividir el voltaje en ciclos de trabajo ofrece muchas ventajas. La información puede ser transmitida modulando la duración y la prolongación del voltaje en un ciclo de trabajo. De igual forma se puede regular la energía que se entrega a una determinada carga. Esta es la función que nos interesa en este post ya que con Arduino podemos modular el ancho de los pulsos de una señal.
Esto quiere decir que podemos reducir en determinado porcentaje la prolongación del voltaje en cada ciclo de trabajo. Digamos que decidimos enviar un pulso de 5 voltios en un 50% del ciclo de trabajo. El resultado será el siguiente:



Cuando se está trabajando al 50% de un ciclo de trabajo, solo se proporciona energía a una carga durante la mitad de un ciclo de trabajo. Durante la otra mitad, se suspende la entrega de energía. El resultado es una onda cuadrada.

Cabe destacar que es posible trabajar a diferentes porcentajes del total del ciclo de trabajo.

PRESENTAMOS LAS SALIDAS ANALOGICAS DE ARDUINO Y SU USO

Para utilizar las salidas analógicas tenemos la instrucción:

analogWrite(pin, value)

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 más reciente Arduino, que implementa el chip ATmega168, permite habilitar como salidas analógicas tipo PWM los pines 3, 5, 6, 9, 10 y 11.

Los modelos de Arduino más antiguos que implementan el chip ATmega8, solo tiene habilitadas para esta función los pines 9, 10 y 11. 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 a 255.

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

Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 5 voltios - el valor HIGH de salida equivale a 5v (5 voltios). Teniendo en cuenta el concepto de señal PWM , por ejemplo, un valor de 64 equivaldrá a mantener 0 voltios de tres cuartas partes del tiempo y 5 voltios a una cuarta parte del tiempo; un valor de 128 equivaldrá a mantener la salida en 0 la mitad del tiempo y 5 voltios la otra mitad del tiempo, y un valor de 192 equivaldrá a mantener en la salida 0 voltios una cuarta parte del tiempo y de 5 voltios de tres cuartas partes del tiempo restante.

Debido a que esta es una función de hardware, en el pin de salida analógica (PWN) se generará una onda constante después de ejecutada la instrucción analogWrite hasta que se llegue a ejecutar otra instrucción analogWrite (o una llamada a digitalRead o digitalWrite en el mismo pin).

Las salidas analógicas a diferencia de las digitales,

no necesitan ser declaradas como INPUT u OUTPUT.

Así podríamos por ejemplo reducir la luminosidad de un LED en un porcentaje de su luminosidad total y eso es lo que vamos a hacer.
Lo que haré es utilizar 4 LEDs para mostrar 3 niveles distintos de luminosidad. Los 3 LEDs necesitarán ser protegidos con resistencias. Hay que usar los pines digitales de Arduino con capacidad de utilizar PWM, es decir, aquellos que tienen el
Símbolo ~ a un lado.

Usaremos los pines 8, 9 y 10 (los pines que tienen el símbolo ~ a su izquierda son los que están habilitados para PWM). El código es el siguiente:

```
void setup()
{

analogWrite(9, 25);// led AZUL
analogWrite(10, 50);//led VERDE
analogWrite(11, 255);//led ROJO

}

void loop()
{

}
```

Subiendo este código y conectando LEDs con sus resistencias en los pines 8, 9 y 10 se obtiene lo que se muestra en la siguiente foto:

Simulación Proteus