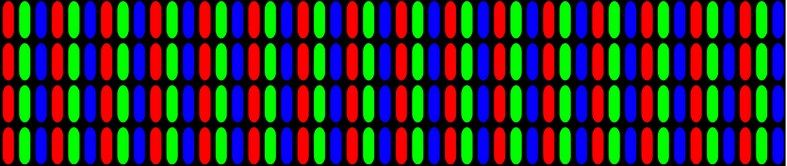
En el presente capítulo introduciremos las señales analógicas. Aprenderemos cómo generarlas usando la técnica de modulación por ancho de pulso, o PWM, que es una herramienta fundamental en el trabajo con microcontroladores. Vaya por delante que, dada la importancia de los conceptos que trabajaremos, este es un capítulo con bastante contenido teórico aunque está continuamente respaldado por ejercicios prácticos. Pero hemos preferido extendernos para que los conceptos, que no son en absoluto intuitivos, queden bien afianzados y puedan ser aplicados en contextos diferentes a los mostrados.

El material que utilizaremos es el siguiente:

* 1 led, de cualquier color.
* 1 led RGB.
* 3 resistencias de 220 Ω.

7.1 Proyecto “Rojo, verde y azul”

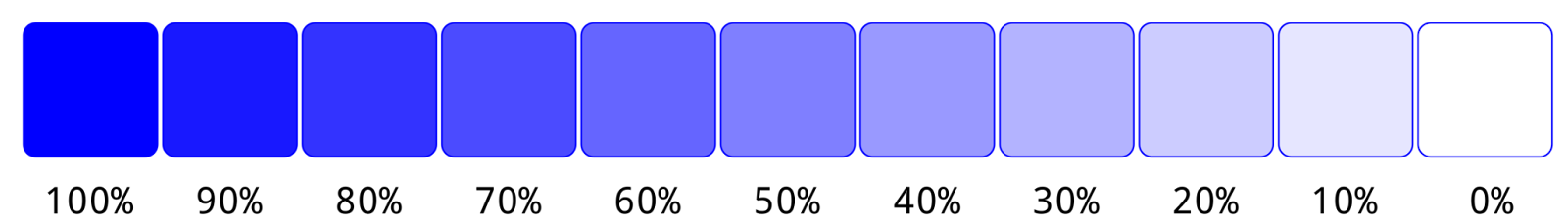
En el año 2014 el Premio Nobel de Física fue concedido a Isamu Akasaki, Hiroshi Amano y Shuji Nakamura por el invento del led azul. Los diodos semiconductores que emitían luz verde y roja ya se fabricaban desde hacía décadas, pero el azul no se inventó hasta principios de los 90. Su descubrimiento no solo propició la creación de una fuente de luz energéticamente eficiente y responsable con el medio ambiente –la iluminación con ledes–, sino que también impulsó el desarrollo de las pantallas de televisores, móviles y ordenadores que usamos en la actualidad.

Rojo, verde y azul son los colores primarios de la luz. Eso quiere decir que, juntándolos en la misma proporción, el resultado es luz blanca. En este proyecto vamos a jugar con un led RGB, un led “tres por uno” que es capaz de emitir luz de los tres colores primarios. Variando la proporción de cada uno de ellos podremos obtener luz de millones de colores diferentes, igual que hacen las pantallas de nuestros teléfonos móviles.

7.2 Señales analógicas

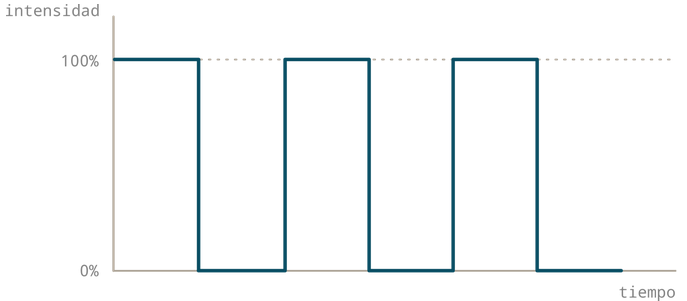
En los capítulos anteriores trabajamos con un led y con un pulsador. En ambos casos solo había dos estados posibles: encendido o apagado en el led, y pulsado o no pulsado en el pulsador. Para los objetivos que nos habíamos propuesto esto era suficiente. Pero, ¿y si quisiéramos encender el led solo a la mitad de su intensidad? El led lo podemos encender o apagar, pero las señales digitales que hemos manejado hasta ahora no tenemos manera de ponerlas en estado “medio encendido”. Ha llegado, pues, el momento de recurrir a otro tipo de señales: las analógicas.

Las señales analógicas son señales que, en contraposición con las digitales, no son del tipo “todo o nada”. Ya no se trata simplemente de encender o apagar un led, sino que con una señal analógica podemos obtener valores intermedios de la intensidad.



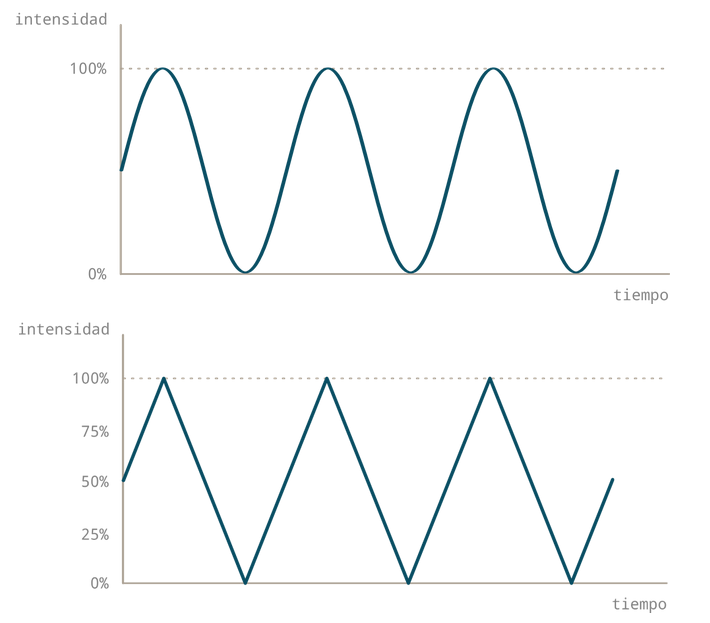
El led puede estar encendido (intensidad del 100 %) o apagado (0 %), pero entre ambos valores también puede estar encendido con diferentes porcentajes de intensidad.

Para ilustrar la diferencia entres las señales analógicas y las digitales en el mundo de los microcontroladores, pensemos en algo tan simple como encender y apagar un led de manera periódica. Hasta ahora solo conocíamos la manera “digital” de hacerlo: la intensidad pasa directamente de 100 % (encendido) a 0 % (apagado), otra vez a 100 %, y así sucesivamente, como se ilustra en la gráfica inferior.



El led se enciende y se apaga periódicamente usando una señal digital. La intensidad del 100 % indica que el led está encendido, y del 0 % que está apagado.

Pero existen otras maneras de encender y apagar el led, como se ilustra en las siguientes gráficas:



El led se enciende y se apaga periódicamente usando una señal analógica. Arriba: el cambio en la intensidad sigue una función sinusoidal. Abajo: el cambio es lineal.

En las gráficas anteriores la intensidad pasa de 100 % a 0 % y de 0 % a 100 % pero, en esta ocasión, al hacerlo va tomando también todos los valores intermedios. Esto es un ejemplo de lo que llamamos señal analógica. Podemos decir que la diferencia fundamental entre ambos tipos de señales es que una señal analógica es continua en el tiempo –es decir, puede tomar cualquier valor dentro de un rango– mientras que la digital es discreta –solo puede tomar dos valores–.

Las señales analógicas son muy abundantes en la naturaleza. De hecho, la mayor parte de los sensores que se conectan a un microcontrolador leen señales analógicas: temperatura, luminosidad, presión, intensidad de campo magnético… Son también las que nos permitirán controlar la luminosidad de un led o la velocidad a la que gira un motor. En este capítulo y en el siguiente aprenderemos cómo escribir señales analógicas, es decir, generaremos estas señales con el microcontrolador. Más adelante veremos cómo leerlas.

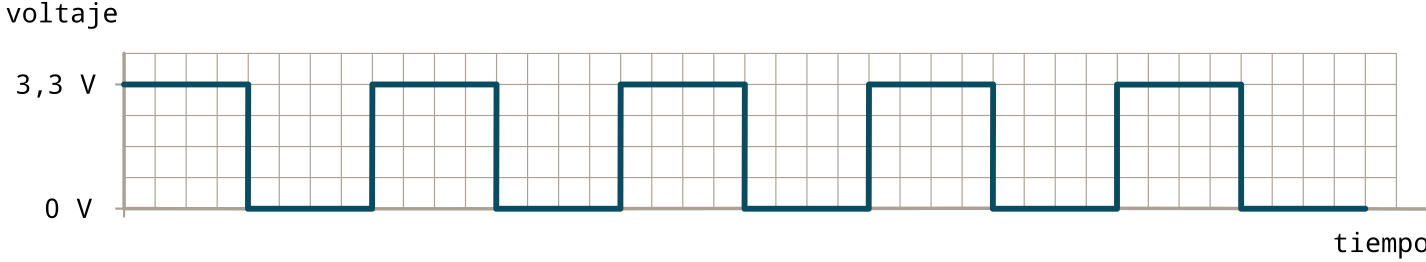
7.3 Modulación por ancho de pulso

Volvamos al problema planteado en el apartado anterior. Queremos conseguir que el led se encienda solo a la mitad de su intensidad. ¿Cómo lo hacemos?

Cuando el led está encendido, el microcontrolador le está suministrando un voltaje de 3.3 V. La manera de obtener una intensidad del 50 % sería suministrándole únicamente la mitad de esos 3.3 V, es decir, 1.65 V. Problema: para un microcontrolador eso es imposible; es 0 voltios (pin apagado) o 3.3 voltios (pin encendido), pero no se puede obtener un valor intermedio. Sin embargo, existe un procedimiento mediante el cual, usando únicamente estos dos valores discretos, se puede generar cualquier voltaje de salida: la modulación por ancho de pulso o *pulse width modulation*, PWM.

La modulación por ancho de pulso permite conseguir una salida analógica en un pin digital. En realidad se trata de una señal analógica ficticia –al fin y al cabo, el microcontrolador solo puede generar señales digitales–, pero a efectos prácticos el resultado es completamente satisfactorio. Veamos, pues, cómo un dispositivo digital puede generar una señal analógica.

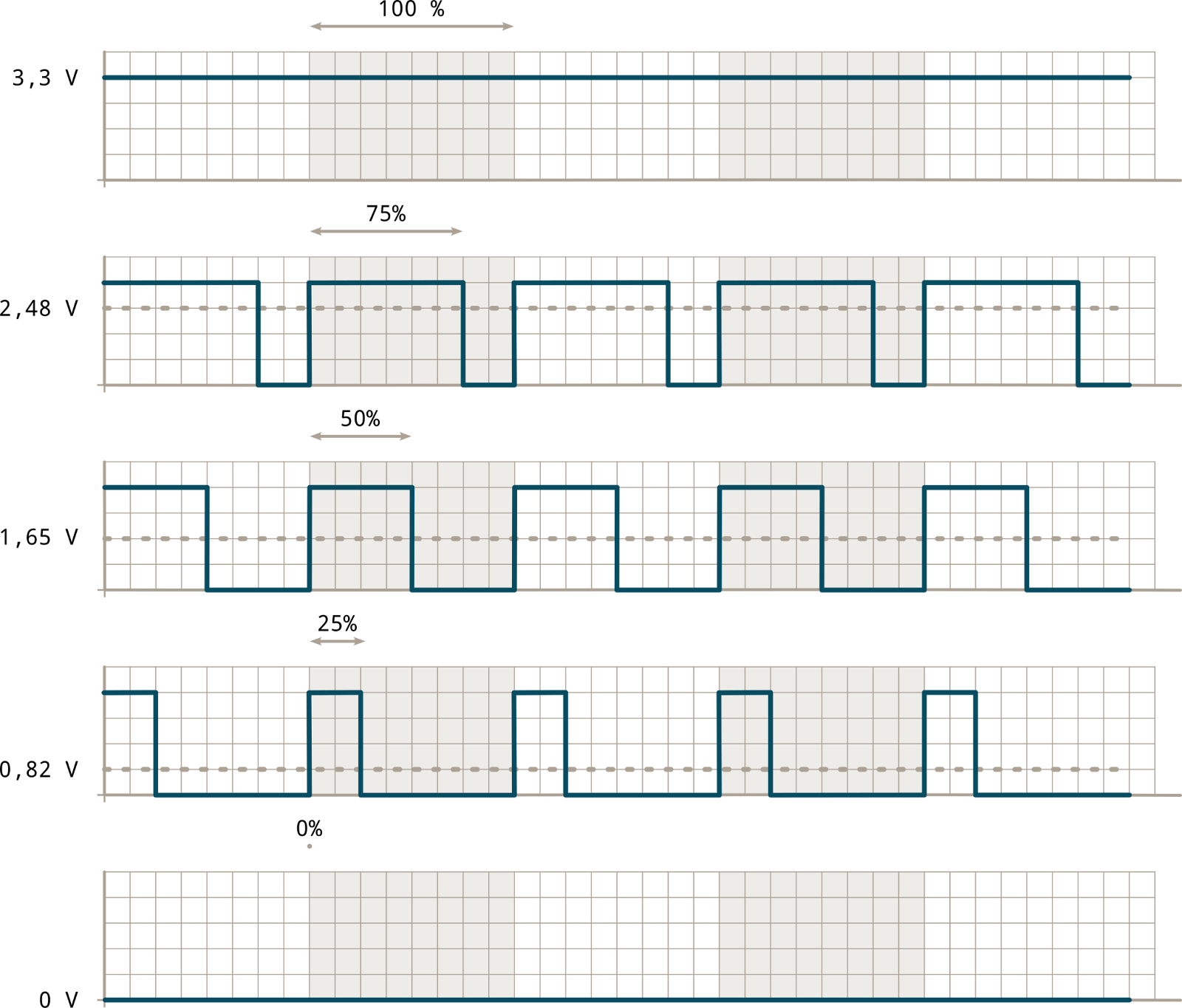
Sabemos que el microcontrolador puede poner un pin en estado on (3.3 V) y off (0 V). Ahora imaginemos que, de alguna manera (ya veremos cómo), somos capaces de generar en el pin una señal cuadrada. Esto es equivalente a encender y apagar el pin de manera periódica.



Señal cuadrada en un pin digital. El voltaje está el 50 % del tiempo en 0 V (apagado) y el otro 50 % en 3.3 V (encendido).

En este caso, ¿cuál es el voltaje que recibe el pin? La mitad del tiempo está a 3.3 V, pero la otra mitad está a 0 V… Entonces, es como si el pin recibiera la media de estos dos valores, o sea, 1.65 V. Y eso es precisamente lo que buscábamos: hemos obtenido un valor intermedio entre 0 y 3.3 voltios. ¡Misión cumplida!

Aunque lo que hemos conseguido supone un gran avance, no es suficiente: necesitamos ser capaces de obtener otros voltajes. La manera de hacerlo es básicamente la misma que en el caso anterior. Si en lugar de mantener el pin en estado on un 50 % del tiempo lo hacemos solo un 25 %, el voltaje efectivo se reduce a la cuarta parte de 3.3 V: ya tenemos 0.82 V. ¿Un 75 % del tiempo en estado on? Obtenemos las tres cuartas partes, 2.48 V. Y así sucesivamente.



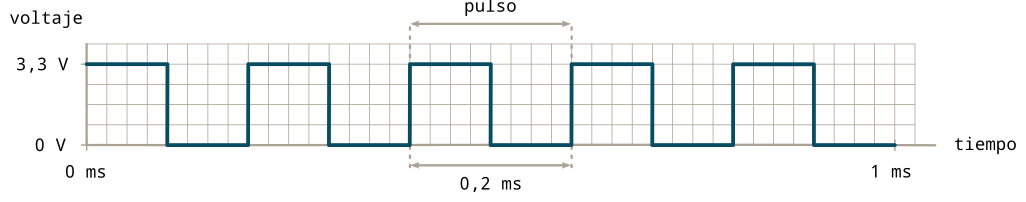
Esta estrategia de alternar periódicamente entre encendido y apagado es la base de la técnica de modulación por ancho de pulso.

7.4 Señales periódicas

Dado que, como acabamos de ver, para obtener un voltaje mediante PWM hay que generar una señal periódica, antes de continuar comentaremos varios conceptos fundamentales sobre este tipo de señales. En particular nos interesan la frecuencia de la señal y el ciclo de trabajo, ya que cuando usemos la técnica PWM estos serán los dos parámetros que manejaremos.

Ciclo o pulso

Una señal periódica está formada por una “unidad”, llamada ciclo o pulso, que se repite en el tiempo, una y otra vez, siempre de la misma manera.



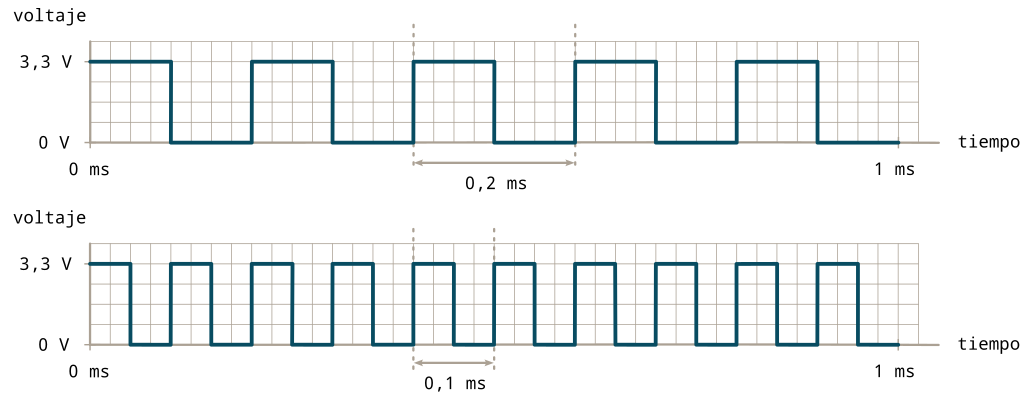
En la imagen se han representado cinco ciclos o pulsos de la señal. Cada pulso tiene una duración de 0.2 ms.

La duración de un pulso (llamada período) se mide en segundos o, más habitualmente, en milisegundos (ms), microsegundos (μs) o, incluso, nanosegundos (ns). En la señal anterior, por ejemplo, el pulso tiene una duración de 0.2 ms, ya que en un milisegundo hay cinco ciclos.

Frecuencia

La frecuencia de una señal es el número de veces que se repite el ciclo en un segundo. Se mide en ciclos por segundo o hercios (Hz), aunque son muy habituales los múltiplos: kilohercio (kHz), megahercio (MHz) y gigahercio (GHz). En la señal anterior el pulso se repite 5 veces en un milisegundo; por tanto tiene una frecuencia de 5 ciclos por milisegundo o, lo que es lo mismo, 5000 ciclos por segundo. Es decir, la señal tiene una frecuencia de 5000 Hz o 5 kHz.

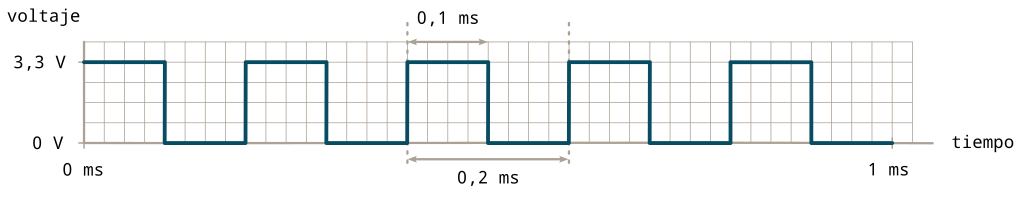
La frecuencia de una señal está íntimamente relacionada con la duración del pulso: cuanto mayor es la frecuencia de una señal menor es la duración de un pulso, y a la inversa.



Dos señales de distinta frecuencia.

Ancho del pulso

El ancho del pulso se define como el tiempo, dentro de un pulso, que la señal está en estado on. Volviendo a la primera señal, cuyo período es de 0.2 ms, el ancho del pulso es 0.1 ms, ya que la mitad del pulso la señal está a 3.3 V.

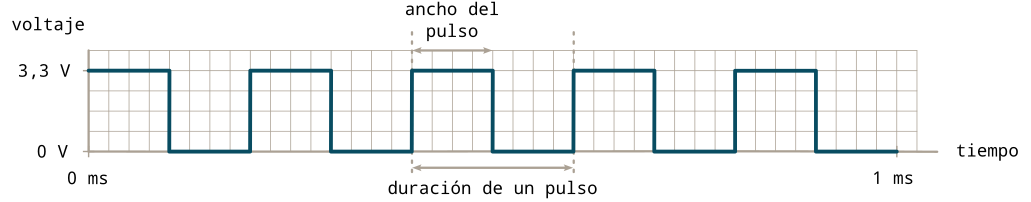


La duración de un pulso es 0,2 ms y el ancho del pulso es 0,1 ms.

Ciclo de trabajo o *duty cycle*

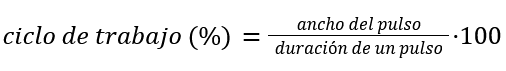
El ciclo de trabajo o *duty cycle*, α, es el cociente entre el ancho del pulso y la duración de un pulso. Toma, por tanto, un valor entre 0 y 1.





En la señal del ejemplo un pulso dura 0.2 s y el ancho del pulso es 0.1 s. Su ciclo de trabajo es 0.1 ms/0.2 ms, es decir, α=0.5.

Habitualmente el valor de α se multiplica por 100 para convertirlo en un porcentaje. Visto así, el ciclo de trabajo indica el porcentaje del tiempo en que la señal está en estado encendido.



El ciclo de trabajo de la señal del ejemplo es, por tanto, del 50 %.

El ciclo de trabajo y el voltaje

En el apartado anterior vimos que para obtener diferentes voltajes en un pin digital debemos variar la proporción del tiempo en que la señal está encendida, es decir, el ciclo de trabajo. Es fácil comprobar que, si la señal oscila entre los valores 0 V y 3.3 V, una señal cuyo ciclo de trabajo es α equivale a un voltaje efectivo *Ve* cuyo valor es:

*Ve*=3.3· α

Es decir, podemos obtener cualquier voltaje sin más que escoger el ciclo de trabajo adecuado.

7.5 La clase **PWM**

Hemos visto que para obtener una señal analógica en un pin digital debemos usar la técnica de modulación de ancho de pulso, que consiste en generar una señal periódica con un ciclo de trabajo determinado. Pues bien, para crear con MicroPython una señal PWM se utiliza la clase PWM del módulo machine. Usando el constructor PWM() se crea un objeto PWM, que sirve para inicializar el pin. A continuación a este objeto, que es la abstracción del pin físico, se le aplican los métodos de la clase PWM para obtener el comportamiento deseado.

El constructor

El constructor transforma un pin del microcontrolador en una salida analógica. Existen más características de la señal que se pueden configurar con MicroPython, pero para empezar nos llega con las que indicamos a continuación.

|  |
| --- |
| PWM(pin, freq=x, duty=x)  PWM(pin, freq=x, duty\_u16=x) |
| Genera una señal analógica en el pin indicado. Argumentos:   * pin: es un objeto tipo Pin que indica el pin de la placa en el que se establece la salida PWM. Es el único argumento obligatorio. * freq: establece la frecuencia de la señal. * duty o duty\_u16: establecen el ciclo de trabajo de la señal.   Si no se indican los argumentos freq y duty o duty\_u16 se les asigna el valor por defecto (este valor depende de la placa). |

Los métodos

En la siguiente tabla se recogen los principales métodos de la clase PWM que actúan sobre un objeto PWM previamente creado.

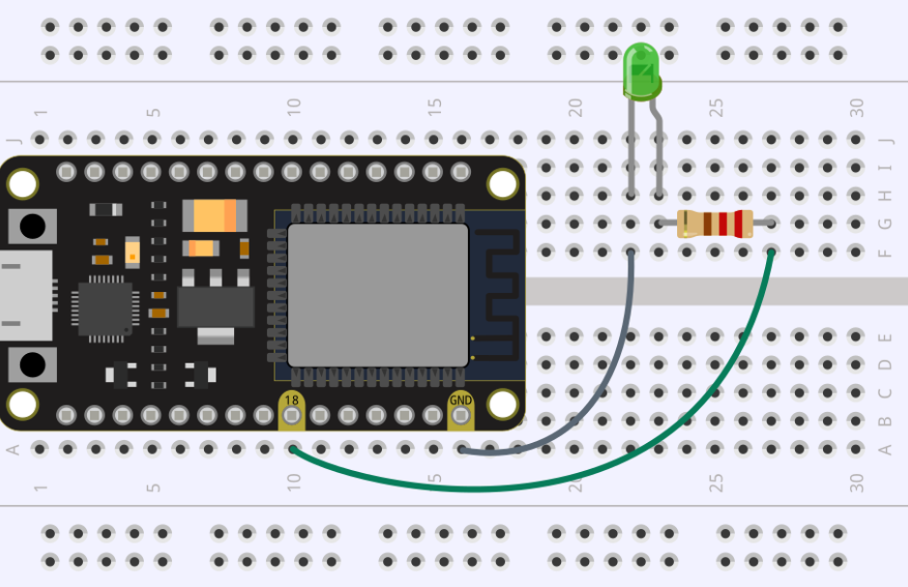
|  |  |
| --- | --- |
| freq() | Devuelve la frecuencia de la señal, en hercios. |
| freq(x) | Establece la frecuencia de la señal. El argumento es un número entero que indica la frecuencia, en hercios. |
| duty() | Devuelve el valor del ciclo de trabajo de la señal con una resolución de 10 bits (en el rango 0-1023). Este método no está disponible para la Raspberry Pi Pico. |
| duty(x) | Establece el ciclo de trabajo de la señal con una resolución de 10 bits. El argumento es un número entero entre 0 y 1023, ambos incluidos (ya que 210=1024). El valor 0 equivale a un ciclo de trabajo del 0 %, 512 de un 50 % y 1023 del 100 %. Este método no está disponible para la Raspberry Pi Pico. |
| duty\_u16() | Devuelve el valor del ciclo de trabajo de la señal con una resolución de 16 bits (en el rango 0-65535). |
| duty\_u16(x) | Establece el ciclo de trabajo de la señal con una resolución de 16 bits. El argumento es un número entero entre 0 y 65 535, ambos incluidos (ya que 216=65 536). El valor 0 equivale a un ciclo de trabajo del 0 %, 32 768 de un 50 % y 65 535 del 100 %. |
| init(freq=x, duty=x)  init(freq=x, duty\_u16=x) | Modifica la frecuencia (freq) o el ciclo de trabajo (duty o duty\_u16) de la señal. |
| deinit() | Desactiva la salida PWM. |

Los *timers*

Una vez configurado un pin como salida analógica (es decir, una vez que se llama al constructor PWM) es el hardware del microcontrolador –un *timer* interno– el encargado de generar la señal continuamente en dicho pin. El hecho de que sea un *timer* el responsable de la señal PWM tiene varias consecuencias. Por un lado, permite que los recursos del procesador queden libres para realizar mientras tanto otras tareas. Pero, por otro, el uso de otras funciones que requieran de los *timers* puede ser incompatible con el uso simultáneo de esta técnica en algunos pines. Además, al tratarse de un componente hardware hay diferencias grandes de implementación según el tipo de placa, por lo que, como veremos más adelante, algunas características cambian en función de la placa.

7.6 Los valores por defecto del objeto **PWM**

Para probar la técnica PWM vamos a crear una señal analógica con la que encenderemos un led. El montaje es el mismo que usamos en capítulos anteriores, por lo que utilizaremos como salida analógica el GPIO18. Preparemos el circuito y probemos las instrucciones en el REPL.



|  |  |  |  |
| --- | --- | --- | --- |
|  | Arduino | ESP32 | Raspberry Pi Pico |
| GPIO18 | D9 | D18 / G18 | 24 |

Empezamos por crear el objeto PMW asociado a la señal analógica con la que manejaremos el led. Para ello necesitamos importar la clase PWM del módulo machine. Dado que el primer argumento (y único obligatorio) es un objeto Pin, no podemos olvidarnos de importar también la clase Pin.

>>> from machine import Pin, PWM

En el constructor PWM usamos como argumento el pin de conexión, en nuestro caso el GPIO18 (aunque estamos definiendo un pin de salida, en el objeto Pin no se debe especificar el modo Pin.OUT, ya que el canal se inicializa internamente en modo PWM al pasarlo al constructor PMW). Como siempre, asociaremos el objeto creado a una variable (la llamaremos LED, pero puede ser cualquier otro nombre) para poder utilizarlo posteriormente.

>>> LED = PWM(Pin(18))

Dependiendo de la placa con la que estemos trabajando, al ejecutar esta instrucción es posible que no haya pasado nada –si tenemos una Pico– o que el led se haya encendido –si la placa es ESP32 o Arduino–. ¿Por qué esta diferencia?

Como ya hemos comentado, la técnica PWM depende del hardware de la placa. Al crear el objeto PWM, en el constructor no hemos indicado el valor de ningún otro argumento aparte del pin, por lo que los demás parámetros asociados a la señal toman el valor por defecto. Consultemos estos valores en función de la placa.

Frecuencia y ciclo de trabajo por defecto, con ESP32 o Arduino

Para ver la frecuencia aplicamos el método freq() al objeto LED; el valor obtenido será probablemente 5000 Hz.

>>> LED.freq()

5000

Para consultar el valor del ciclo de trabajo, con la ESP32 o una placa Arduino podemos usar duty() o duty\_u16(). Ambas funciones devuelven el valor del *duty cycle*; la diferencia es que duty() nos da el valor del ciclo de trabajo con una resolución de 10 bits (es decir, como un entero entre 0 y 1023) mientras que duty\_u16() devuelve este valor con resolución de 16 bits (entre 0 y 65535).

>>> LED.duty()

512

>>> LED.duty\_u16()

32768

En ambos casos vemos que el ciclo de trabajo por defecto es la mitad del valor máximo, es decir, del 50 %.

Recapitulando, al llamar al constructor se genera en el pin una señal analógica que tiene, por defecto, una frecuencia de 5000 Hz y un ciclo de trabajo del 50 %. Este es el motivo por el cual el led se encendió, pero solo con la mitad de intensidad.

Frecuencia y ciclo de trabajo por defecto, con Raspberry Pi Pico

Veamos cuáles son los parámetros por defecto con los que se crea la señal analógica en la Raspberry Pi Pico. El valor por defecto de la frecuencia es cero hercios.

>>> LED.freq()

0

Para consultar el valor del ciclo de trabajo usamos duty\_u16() (el método duty() no está disponible para la Pico), que devuelve el valor del ciclo de trabajo con una resolución de 16 bits, es decir, como un entero entre 0 y 65535. Su valor por defecto en la Raspberry Pi Pico es cero.

>>> LED.duty\_u16()

0

A la vista de estos valores está claro por qué no se encendió el led: al llamar al constructor se ha generado en el pin una señal analógica con una frecuencia de 0 Hz y un ciclo de trabajo del 0 % (o sea, una señal de cero voltios).

7.7 Encendiendo un led con PWM

Apliquemos lo visto hasta ahora para encender el led con una intensidad del 100 % utilizando una señal analógica. En primer lugar debemos hacernos la siguiente pregunta: ¿cuáles van a ser los valores de la frecuencia y el ciclo de trabajo de la señal?

Cuando se trata de encender un led, el valor de la frecuencia de la señal es bastante irrelevante (siempre que no se escoja un valor muy bajo, como veremos más adelante); podemos establecer 5000 Hz como un valor adecuado. En cuanto al ciclo de trabajo, si nuestra intención es que la intensidad sea del 100 % debemos escoger el valor máximo: 1023 si lo establecemos con duty o 65535 si usamos duty\_u16.

Hay varias maneras de proceder: por un lado, se puede crear el objeto PWM con los valores por defecto y después se modifican estos valores; por otro, se pueden indicar en el propio constructor los valores deseados de los parámetros. Sea cual sea el método que elijamos, empezaremos siempre por importar los módulos necesarios.

>>> from machine import Pin, PWM

Método 1.Se especifican los valores de la frecuencia y el ciclo de trabajo en el constructor, en el momento de crear el objeto PWM. Para ello se indican los argumentos freq con el valor 5000 y duty\_u16 con el valor 65535.

>>> LED = PWM(Pin(18), freq=5000, duty\_u16=65535)

El ciclo de trabajo también se podría indicar con duty=1023, aunque este argumento no está disponible para la Raspberry Pi Pico.

Método 2. Se crea el objeto PWM con los argumentos por defecto y, a continuación, se establecen la frecuencia y el ciclo de trabajo con el método init().

>>> LED = PWM(Pin(18))

>>> LED.init(freq=5000, duty\_u16=65535)

Método 3. Alternativamente, en lugar de usar init() los valores de los parámetros se pueden modificar con los métodos freq() y duty() o duty\_u16().

>>> LED = PWM(Pin(18))

>>> LED.freq(5000)

>>> LED.duty\_u16(65535)

Método 4. Cualquier combinación de los métodos anteriores que encontremos más adecuada.

Todas las maneras anteriores de encender el led son equivalentes, aunque dependiendo de la aplicación puede ser más adecuada una u otra.

7.8 ¿Apagado o desactivado?

Encendamos el led con PWM.

>>> from machine import Pin, PWM

>>> LED = PWM(Pin(18), freq=5000, duty\_u16=65535)

Ahora vamos a apagarlo. También hay diferentes maneras de hacerlo.

Con el ciclo de trabajo

Una manera de apagar el led es poniendo a cero el ciclo de trabajo de la señal analógica. Un ciclo de trabajo de cero quiere decir que, dentro de un ciclo, la señal está en estado encendido el 0 % del tiempo. O sea, que está apagada el 100 % del tiempo y por eso el led se apaga.

>>> LED.duty\_u16(0)

Desactivando la señal

La misión del método deinit() es desactivar la salida PWM en el pin.

>>> LED.deinit()

Como la señal se ha desactivado, el led se apaga.

¿Ciclo de trabajo o **deinit()**?

Para escoger un método o el otro para apagar el led debemos recordar que, una vez inicializado el pin PWM, el hardware del dispositivo se encarga automáticamente de generar la señal de manera continua. En consecuencia, aunque apaguemos el led poniendo a cero el ciclo de trabajo, la señal analógica todavía se sigue generando. A su vez, la finalidad del método deinit() es desactivar la salida PWM; es decir, provoca que los *timers* dejen de generar la señal analógica. Veamos cuáles son las consecuencias prácticas que tiene esto.

Apaguemos el led haciendo que el ciclo de trabajo de la señal sea cero. Si después de apagarlo queremos volver a encenderlo no hay ningún problema: ponemos el ciclo de trabajo de vuelta a 65535 y listo, ya tenemos el led encendido.

>>> LED.duty\_u16(0)

>>> LED.duty\_u16(65535)

Apaguémoslo ahora aplicando el método deinit() al pin PWM. Si ahora queremos volver a encender el led tenemos un problema.

>>> LED.deinit()

>>> LED.duty\_u16(65535)

RuntimeError: PWM inactive

El motivo del mensaje de error debería estar claro: la señal PWM a la que pretendemos aplicar el método duty() ya no existe, porque la desactivamos con deinit(). Para volver a encender el led, por tanto, hay que volver a activar el pin usando el método init().

>>> LED.init(freq=5000, duty\_u16=65535)

En las distintas aplicaciones debemos tener en cuenta las diferencias entre una manera y otra de poner a cero la señal analógica para escoger la más adecuada.

7.9 La frecuencia

Este apartado y el siguiente los vamos a dedicar a jugar con los valores de la frecuencia y el ciclo de trabajo de la señal, ya que estas son las dos magnitudes que definen las señales analógicas con las que trabajaremos.

Empecemos por la frecuencia. Como vimos con anterioridad, la frecuencia de una señal es el número de ciclos por segundo que tiene la señal. Dicho de una manera poco rigurosa, es lo rápido que se enciende (se pone a 3.3 V) y apaga (se pone a 0 V) el pin periódicamente para conseguir el voltaje deseado. ¿Cómo afecta la frecuencia a la señal generada?

Vamos a inicializar el pin PWM del led con una frecuencia de 5000 Hz y un ciclo de trabajo del 50 % (para este experimento es importante que el ciclo de trabajo no sea del 100 %).

>>> from machine import Pin, PWM

>>> LED = PWM(Pin(18), freq=5000, duty\_u16=32768)

Utilizamos el método freq() para cambiar el valor de la frecuencia de la señal y observamos qué sucede en el led.

>>> LED.freq(1000)

Con una frecuencia de 1000 Hz no se observa ningún cambio en el led. ¿Qué sucede con otros valores? Si probamos valores mayores seguiremos sin observar ninguna diferencia. Pero si utilizamos valores bajos, empieza a notarse algo.

>>> LED.freq(20)

Si se establece la frecuencia en, por ejemplo, 20 Hz vemos que el led parpadea; de hecho, el ciclo encendido-apagado se repite exactamente veinte veces en un segundo. Si hemos entendido la modulación por ancho de pulso esto no nos sorprenderá; en realidad, con frecuencias más elevadas el led también está parpadeando, aunque lo hace tan rápido que a nuestros ojos no les da tiempo a notarlo.

La conclusión de este pequeño experimento es que, si estamos manejando un led con PWM, el valor de la frecuencia es irrelevante, siempre que no sea demasiado bajo. En otras aplicaciones, sin embargo, este valor tendrá una importancia fundamental, como veremos en el próximo capítulo.

7.10 El ciclo de trabajo

A estas alturas ya tenemos claro que el ciclo de trabajo controla la intensidad con que se enciende el led. Volvamos a la señal de 5000 Hz y comprobemos los valores extremos del ciclo de trabajo. Empecemos con el led apagado.

>>> from machine import Pin, PWM

>>> LED = PWM(Pin(18), freq=5000, duty\_u16=0)

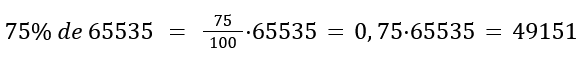
Ahora encendámoslo con la intensidad máxima, que se consigue pasando al método duty\_u16() el valor 65535 (o, si se usa duty(), el valor 1023).

>>> LED.duty\_u16(65535)

Claramente, una intensidad del 50 % se obtendrá con la mitad del valor máximo (32768 si se usa duty\_u16() o 512 si se usa duty()).

>>> LED.duty\_u16(32768)

¿Y qué valor ponemos para otras intensidades? Necesitamos “traducir” el porcentaje de intensidad al rango 0-65535. Afortunadamente, los valores de duty\_u16 correspondientes a un determinado ciclo de trabajo se obtienen con una simple proporción. Dado que 65535 es el 100 %, una intensidad de, por ejemplo, el 75 % equivale a un valor de duty\_16 del 75 % de 65535, es decir:

Hemos redondeado el resultado porque este parámetro solo puede tomar valores enteros.

>>> LED.duty\_u16(49151)

Por otro lado, si usamos duty, que nos da una resolución de 10 bits en lugar de los 16 del ejemplo anterior, la intensidad del 75 % equivaldría a un valor de duty del 75 % de 1023, es decir, 75/100·1023=0.75·1023=767.

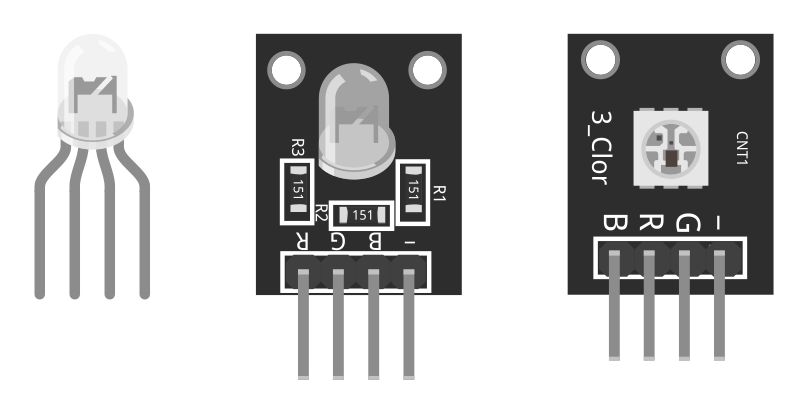
Procederíamos de la misma manera para cualquier otro valor de la intensidad. En la práctica nunca tendremos que calcular “a mano” estos valores, sino que haremos las operaciones indicadas en el propio programa.

7.11 El led RGB

Para ilustrar la técnica PWM hemos utilizado un led “normal”, como los que usamos en capítulos anteriores. Pero el objetivo de este capítulo es crear diferentes colores usando un led RGB, así que introduzcamos este dispositivo.

Un led RGB es, en realidad, tres ledes. En su interior tiene tres materiales semiconductores diferentes, cada uno de los cuales emite, respectivamente, luz roja, verde y azul. Mezclando estos tres colores en diferentes proporciones se pueden obtener millones de colores diferentes.

Los ledes RGB están disponibles en diferentes presentaciones. Los más habituales son los que tienen un encapsulado similar al de los ledes monocromos de los capítulos anteriores, con dos diferencias: la carcasa es transparente, y en lugar de tener dos terminales tienen cuatro. En muchas ocasiones vienen montados en un módulo para facilitar las conexiones. Hay también otros módulos en los que el led RGB se presenta en una carcasa del tipo SMD (del inglés *Surface Mount Device* o dispositivo de montaje en superficie); en ellos el led es plano y ocupa menos espacio.



Diferentes presentaciones de los ledes RGB. El de la derecha en un led del tipo SMD.

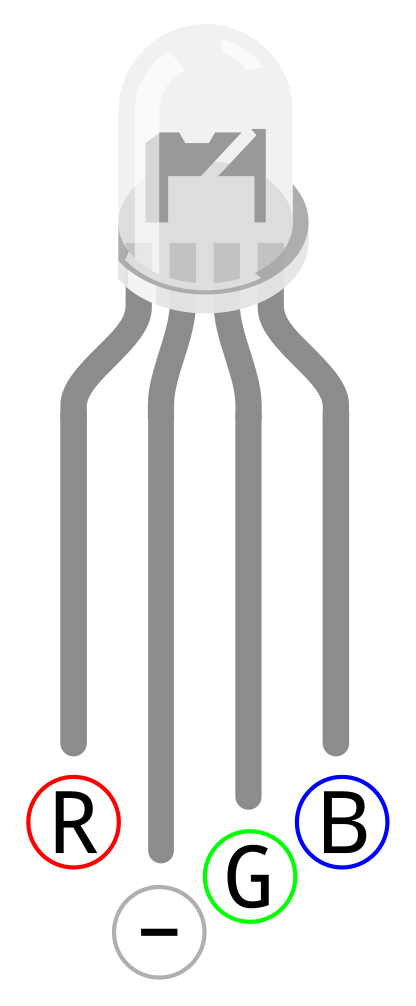
Vaya por delante que algunos led RGB no destacan por su calidad y los colores obtenidos no son espectaculares; pudiendo escoger, usaremos un led SMD. Además, si lo rodeamos de un recubrimiento translúcido obtendremos resultados más satisfactorios.

Las cuatro patas del led

Como el led RGB está formado por tres ledes, en lugar de tener dos terminales tiene cuatro: uno para cada uno de los colores rojo, verde y azul, y el cuarto es la conexión negativa que comparten los tres (lo más probable es que el led sea de cátodo común, y es lo que asumiremos en la explicación).

¿Cuál es la función de cada una de las patas del led, y cómo se distinguen? Si el led viene montado en un módulo, la función de cada una viene identificada con una letra: “R” para controlar el led rojo, “G” para el led verde, “B” para el azul y “–” es el cátodo, que irá conectado a tierra (GND en el microcontrolador). Si el led no está montado en un módulo, para identificarlas lo primero que hay que hacer es localizar la pata más larga:

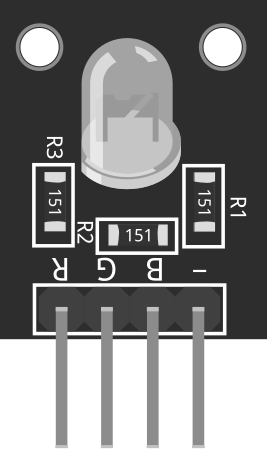
* La pata más larga es el cátodo (–), es decir, la conexión negativa.
* A un lado de la pata más larga hay una sola pata. Esta es la que controla el led rojo (R).
* Al otro lado de la pata larga hay dos patas. De estas, la que está más cerca del cátodo es la que controla el led verde (G).
* La pata que queda (la más alejada del cátodo) es la correspondiente al led azul (B).



Pines en un led RGB de cátodo común: rojo (R), verde (G), azul (B) y cátodo (–).

¿Con o sin resistencia?

De los capítulos anteriores sabemos que para utilizar un led debemos siempre añadir una resistencia en serie. Como el led RGB es un “tres por uno”, necesitaremos tres resistencias. Sin embargo, en muchas ocasiones los módulos en los que se montan estos ledes ya traen incorporadas las resistencias necesarias; así, cuando montemos el circuito no tendremos que preocuparnos de ponerlas. Aunque los proyectos que te presentamos pueden realizarse con cualquiera de los modelos, recomendamos utilizar el que tiene las resistencias integradas. ¡A veces las resistencias no hacen buen contacto en la placa de montaje!



Módulo RGB con las resistencias integradas (son los elementos rotulados como R1, R2 y R3).

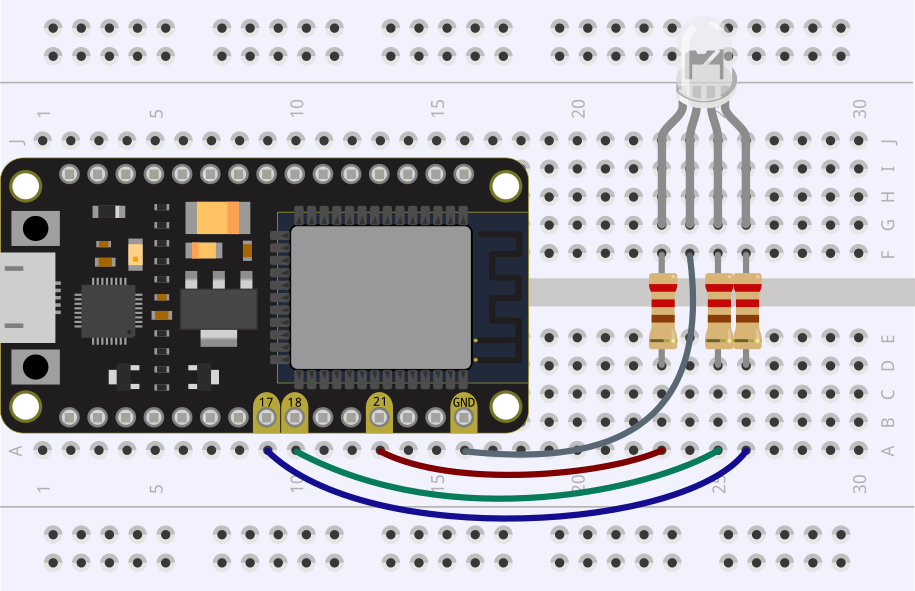
Sea cual sea el led, antes de proceder al montaje debemos localizar la conexión negativa (el cátodo) y cada uno de los terminales rojo, verde y azul. Comprobaremos también si necesitamos resistencias o si, por contra, ya vienen integradas en el módulo.

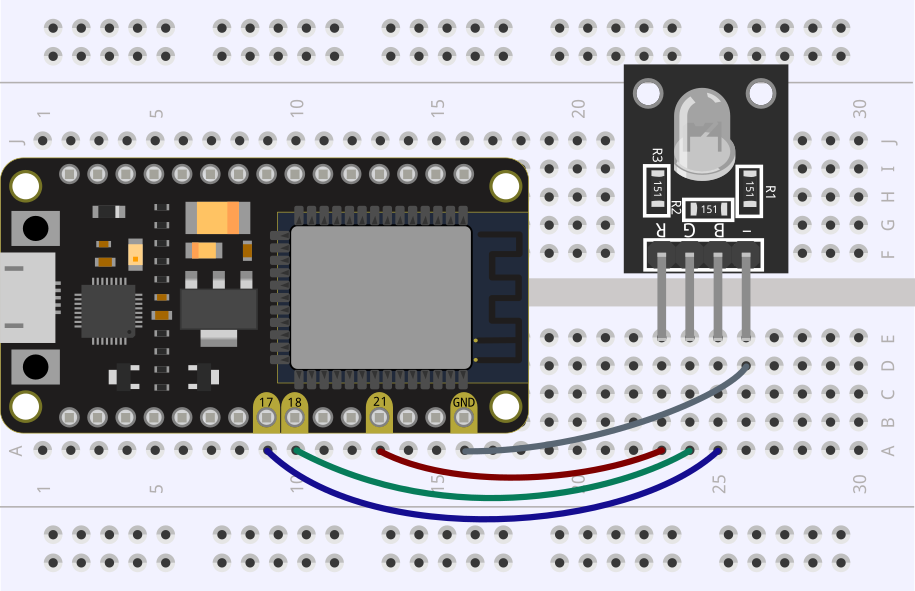
7.12 El circuito

Cuando programemos el led RGB manejaremos los tres colores primarios por separado. Por ello, cada uno de los terminales R, G y B debe ir conectado a un GPIO diferente en la placa, y el cátodo irá conectado a GND. Usaremos los pines indicados en la tabla inferior aunque, como siempre, se pueden escoger otros.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Led | GPIO | Arduino | ESP32 | Raspberry Pi Pico |
| Azul (B) | GPIO17 | D8 | TX2 / G17 | 22 |
| Verde (G) | GPIO18 | D9 | D18 / G18 | 24 |
| Rojo (R) | GPIO21 | D10 | D21 / G21 | 27 |
| Cátodo (–) | GND | GND | GND | GND |

Por otro lado, como ahora tenemos “tres” ledes, tendremos que utilizar tres resistencias de 220 Ω, una para cada uno de los colores; sin embargo, si el módulo RGB ya trae las resistencias integradas no es necesario añadirlas.

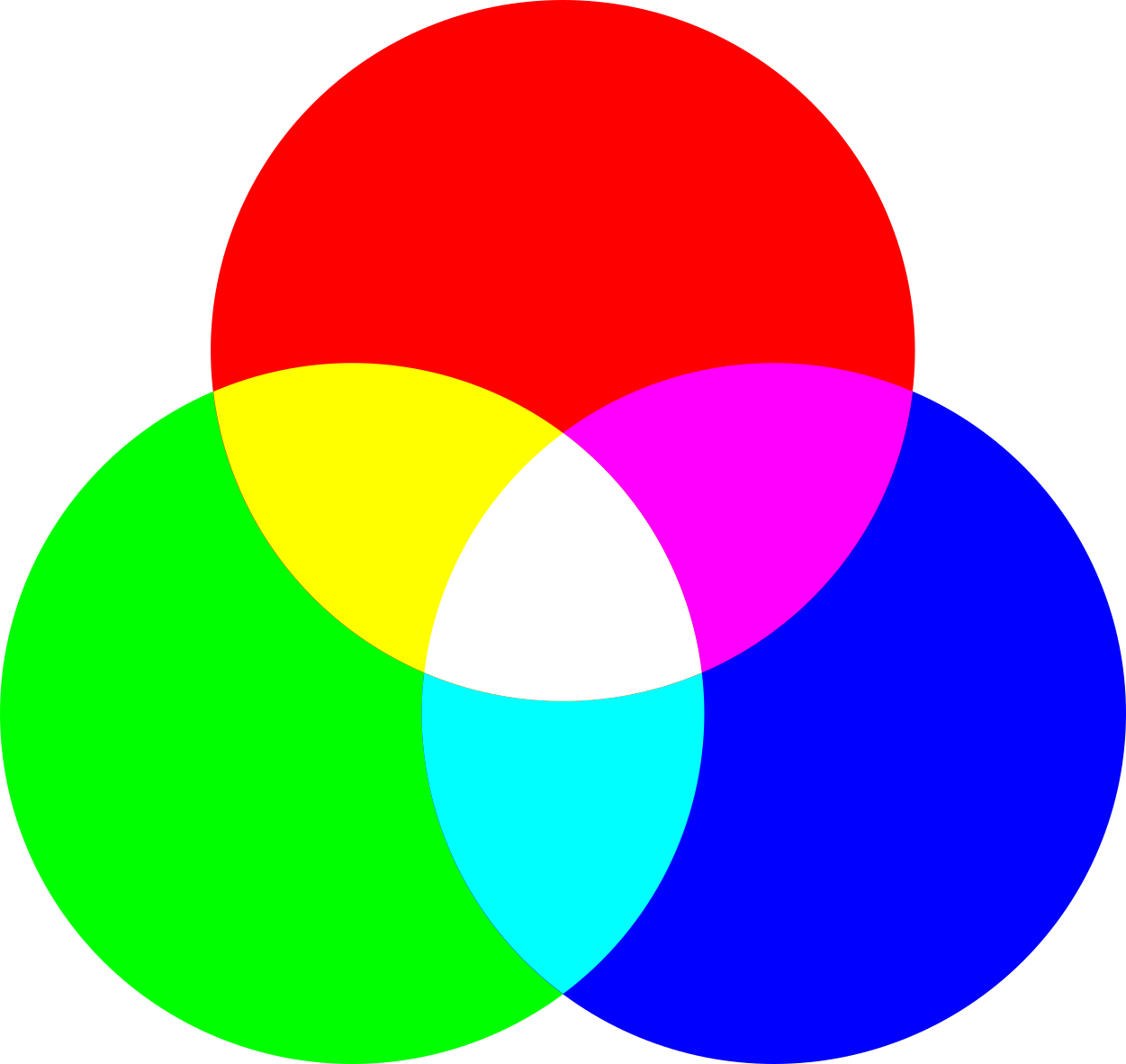




7.13 Los colores primarios y secundarios de la luz

Por fin, después de muchos conceptos nuevos –y no demasiado intuitivos– estamos en condiciones de afrontar el objetivo que nos marcamos en este capítulo: crear luz de diferentes colores. Rojo, verde y azul son los colores primarios de la luz. Si se mezclan los tres en la misma proporción se obtiene luz blanca. Pero, ¿y si se mezclan solo dos de los tres colores primarios? En tal caso lo que se obtiene son los llamados colores secundarios de la luz: rojo y verde da amarillo; verde y azul da cian, y rojo y azul da magenta (siempre que ambos estén en la misma proporción).

Para comprobar que el led está bien conectado planteamos un reto: encender el led con los colores primarios y secundarios de la luz.



Colores primarios (rojo, verde y azul) y secundarios (cian, magenta y amarillo) de la luz.

Para resolver el problema empezamos por importar las clases Pin y PWM del módulo machine.

>>> from machine import Pin, PWM

Creamos sendos pines PWM, uno para cada color. Inicializamos la señal con una frecuencia de 5000 Hz (en ESP32 y Arduino este es el valor por defecto, así que no es necesario indicarlo explícitamente); el ciclo de trabajo lo ponemos a cero para que no se enciendan.

>>> LED\_ROJO = PWM(Pin(21), freq=5000, duty\_u16=0)

>>> LED\_VERDE = PWM(Pin(18), freq=5000, duty\_u16=0)

>>> LED\_AZUL = PWM(Pin(17), freq=5000, duty\_u16=0)

Para obtener cada color primario solo debemos encender el led correspondiente mientras los otros dos están apagados. Para encenderlo ponemos el valor máximo del ciclo de trabajo aplicando duty\_u16(65535) y para apagarlo lo ponemos a cero con duty\_u16(0) (o usamos duty(1023) y duty(0), respectivamente).

>>> LED\_ROJO.duty\_u16(65535); LED\_VERDE.duty\_u16(0); LED\_AZUL.duty\_u16(0)

>>> LED\_ROJO.duty\_u16(0); LED\_VERDE.duty\_u16(65535); LED\_AZUL.duty\_u16(0)

>>> LED\_ROJO.duty\_u16(0); LED\_VERDE.duty\_u16(0); LED\_AZUL.duty\_u16(65535)

Si alguno de los colores ha fallado se deben revisar las conexiones (especialmente si hemos colocado resistencias).

Análogamente, los colores secundarios los obtendremos encendiendo dos ledes y apagando el tercero.

>>> LED\_ROJO.duty\_u16(65535); LED\_VERDE.duty\_u16(65535); LED\_AZUL.duty\_u16(0)

>>> LED\_ROJO.duty\_u16(0); LED\_VERDE.duty\_u16(65535); LED\_AZUL.duty\_u16(65535)

>>> LED\_ROJO.duty\_u16(65535); LED\_VERDE.duty\_u16(0); LED\_AZUL.duty\_u16(65535)

Finalmente, si los encendemos todos obtenemos luz blanca (más o menos azulada).

>>> LED\_ROJO.duty\_u16(65535); LED\_VERDE.duty\_u16(65535); LED\_AZUL.duty\_u16(65535)

Cuando hayamos terminado de hacer las pruebas, para apagar el led procedemos a desactivar los pines PWM.

>>> LED\_ROJO.deinit(); LED\_VERDE.deinit(); LED\_AZUL.deinit()

Dado que en este ejercicio hemos encendido los ledes al 100 % de la intensidad lo podíamos haber resuelto usando señales digitales, como en los capítulos previos. Pero hemos preferido hacerlo así para ganar más familiaridad con las señales analógicas.

7.14 Azul turquesa

Cualquier color de la luz se puede obtener mezclando los colores primarios en las proporciones adecuadas. El color azul turquesa, por ejemplo, se obtiene mezclando un 6 % de luz roja, un 87 % de luz verde y un 69 % de luz azul (se pueden encontrar en multitud de referencias *online* los porcentajes de rojo, verde y azul necesarios para obtener los diferentes colores). El reto es ahora que el led se encienda con este bonito color.

Una posible manera de solucionar el reto es con el siguiente programa.

from machine import Pin, PWM

LED\_ROJO = PWM(Pin(21), freq=5000, duty\_u16=0)

LED\_VERDE = PWM(Pin(18), freq=5000, duty\_u16=0)

LED\_AZUL = PWM(Pin(17), freq=5000, duty\_u16=0)

r = 6

g = 87

b = 69

LED\_ROJO.duty\_u16(int(r/100\*65535))

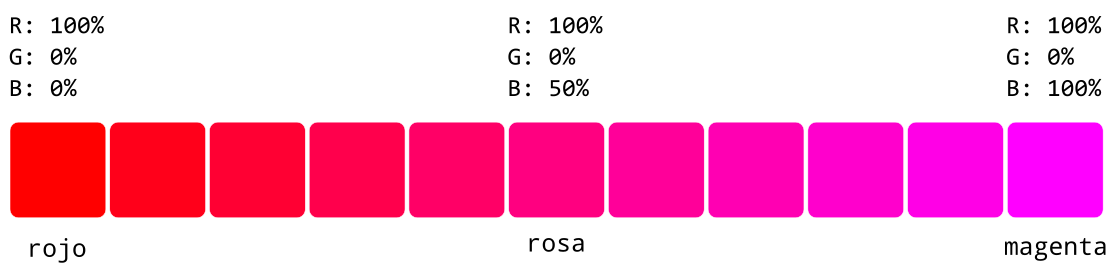
LED\_VERDE.duty\_u16(int(g/100\*65535))

LED\_AZUL.duty\_u16(int(b/100\*65535))

Las primeras instrucciones tienen poco que explicar: empezamos importando las clases Pin y PWM e inicializamos los pines PWM –uno para cada color primario–. A continuación creamos tres variables –r, g y b– con el porcentaje respectivo de rojo, verde y azul; de esta manera es muy fácil actualizar los valores para crear cualquier otro color. Por último, usando el método duty\_u16() en cada pin PWM adjudicamos el ciclo de trabajo correspondiente a cada color. Para ello, como vimos con anterioridad, dividimos el porcentaje entre cien y lo multiplicamos por 65535 para traducirlo al valor adecuado en el rango 0-65535. Dado que este número tiene que ser entero, utilizamos la función int() que, como su nombre indica, transforma un número real en un entero. Al ejecutar el programa el led se enciende en color turquesa.

7.15 Del rojo al magenta, pasando por el rosa

Añadiendo al rojo distintos porcentajes de azul se obtienen diferentes tonos de rosas, hasta llegar al magenta. Para finalizar, programemos el led para que pase, de manera continua, del rojo al magenta.



from machine import Pin, PWM

from time import sleep\_ms

LED\_ROJO = PWM(Pin(21), freq=5000, duty\_u16=65535)

LED\_VERDE = PWM(Pin(18), freq=5000, duty\_u16=0)

LED\_AZUL = PWM(Pin(17), freq=5000, duty\_u16=0)

for i in range(0, 65536, 100):

LED\_AZUL.duty\_u16(i)

sleep\_ms(5)

LED\_ROJO.deinit()

LED\_VERDE.deinit()

LED\_AZUL.deinit()

En esta ocasión hemos utilizado un bucle for para obtener el ciclo de trabajo. La función range(0, 65536, 100) devuelve una lista con los valores enteros desde 0 (incluido) hasta 65536 (no incluido) tomados de 100 en 100. Estos son los valores que pasamos a duty\_u16 para encender el led azul con la intensidad correspondiente.

En resumen

En este extenso capítulo hemos trabajado con señales analógicas. Hemos introducido la modulación por ancho de pulso, una técnica fundamental en el trabajo con microcontroladores a la que se recurre en multitud de aplicaciones. Para entender cómo funciona esta herramienta hemos analizado diferentes características de las señales periódicas, como son la frecuencia de la señal y el ciclo de trabajo. Usando la clase PWM del módulo machine hemos generado señales analógicas en el microcontrolador y las hemos usado para encender un led RGB con diferentes colores.