

Práctica 1

Introducción a la Raspberry Pi y al mecanismo PWM para modular una señal digital (LED)

Grado en Ingeniería en Robótica Software

GSyC, Universidad Rey Juan Carlos



(CC) Julio Vega

1. Introducción

El mundo se está automatizando, con grandes cantidades de datos producidos y procesados para fines analíticos, de control y de conexión. La Raspberry Pi puede proporcionar una amplia gama de automatización y procesamiento de datos si se explota todo su potencial. Esta pequeña placa ofrece amplias funcionalidades y oportunidades para cambiar el mundo que nos rodea. Esta práctica es el primer paso para hacerlo...

Dada la gran capacidad de esta placa para interactuar con el mundo exterior, las principales aplicaciones que se pueden desarrollar usando Raspberry Pi son aquellas en las que se usan los diferentes sensores disponibles para esta y se convierten los datos vertidos por estos en datos útiles para analizar y controlar los dispositivos que vamos a experimentar durante el curso.

2. Descripción de la placa

Lo primero de todo es identificar los diferentes conectores y elementos de la placa. Para ello, observemos la imagen 1.

- Puertos GPIO: GPIO significa Entrada/Salida de Propósito General. Los vamos a usar mucho durante el curso, así que ya los veremos en detalle en la siguiente práctica. Lo mejor de estos puertos es que se les puede asignar una tarea específica según el GPIO específico del programa. Podremos leer valores de cualquier periférico, como sensores o actuadores, así como enviar los valores calculados en nuestros programas. También podremos conectar LEDs o pantallas LCD. Estos puertos, sin duda, marcan la gran diferencia entre la Raspberry Pi y cualquier otra placa microcontroladora, al dar a los desarrolladores la absoluta libertad de crear.
- Conector de salida de audio de 3,5 mm: si no se usa la conexión HDMI (que describimos a continuación), el audio se puede reproducir a través de altavoces o auriculares utilizando este conector estándar de 3,5 mm.
- USB: este es el conector más común, ampliamente utilizado en cualquier ordenador, y por lo tanto llamado Universal Serial Bus. Puedes conectar un pendrive, teclado, ratón e incluso un concentrador USB con alimentación externa para poder conectar más periféricos USB.
- Ethernet: esta es una conexión muy importante, además, claro está, de la conexión por Wi-Fi que ofrece este modelo, pues nos es de utilidad para tener un control remoto sobre la Raspberry Pi, así como para proporcionar conexión a Internet por cable. No siempre podemos conectar la placa a una pantalla dedicada, por lo que usamos el inicio de sesión remoto y así vemos todo el Escritorio y el interfaz de línea de comandos de la Pi en la pantalla de nuestro equipo.

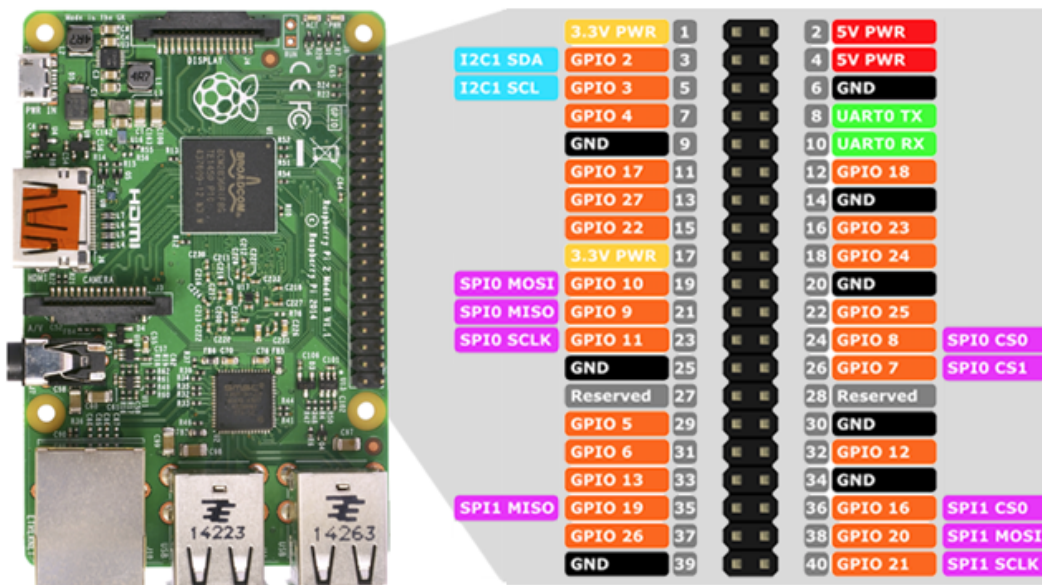


Figura 1: Elementos e interfaz GPIO en Raspberry Pi

- Conector de cámara CSI: la placa no incluye la cámara, pero sí incorpora un conector CSI para esta, la PiCamera, que se puede adquirir por separado. La cámara incluye un cable flexible de 15 cm (cuanto más largo es, hay más posibilidad de perder calidad en la imagen). La cámara tiene 8 megapíxeles y se puede usar, gracias a este puerto, para grabar vídeos de alta definición, así como fotografías fijas y, como no, desarrollar algoritmos de visión computacional.
- Conector HDMI: la interfaz multimedia de alta definición (HDMI) es un interfaz de audio/vídeo compacta utilizada para transferir datos multimedia sin comprimir. Gracias a este, puedes conectar una pantalla que disponga de esta moderna conexión para ver la imagen en alta definición. Si se usa este conector, ya no es necesario conectar los altavoces a la toma de audio.
- Micro USB: a través de este puerto, la placa recibe la energía necesaria para funcionar. Necesita de un voltaje de entrada de 5V y un mínimo de intensidad de 2'5A, aunque este último valor depende de los dispositivos que estén conectados a la Raspberry. Cabe añadir que, como se puede apreciar, la placa no tiene botón de encendido; la placa se enciende simplemente enchufando el adaptador de corriente.
- Ranura para microSD: la microSD es importante porque es donde la Raspberry tiene instalado su sistema operativo. También es donde almacenará todos nuestros documentos, programas, etc. Es el disco duro de la placa. En cuanto a la RAM, viene integrada en la placa.

3. Instalación del Sistema Operativo

Para instalar un sistema operativo necesitamos una tarjeta microSD donde instalarlo. Una vez que tenemos la microSD, vamos a proceder a descargar e instalar una imagen de sistema operativo válido para Raspberry Pi. Existen numerosas distribuciones. Nosotros vamos a usar la más estandarizada para esta placa: Raspberry Pi OS. Para ello, seguimos estos pasos:

1. Descargar del siguiente enlace la imagen que incluye la parte de escritorio (*Raspberry Pi OS with desktop and recommended software*):
<https://www.raspberrypi.org/software/operating-systems/> (3G).
2. Mientras, vamos formateando la microSD. Para ello, usamos la herramienta Discos de Linux. Le damos formato *Fat32*.

3. Una vez se haya descargado la imagen, volvemos a usar **Discos** con la opción de *Restaurar imagen*. Seleccionamos la que acabamos de descargar y empezamos su restauración en la microSD.

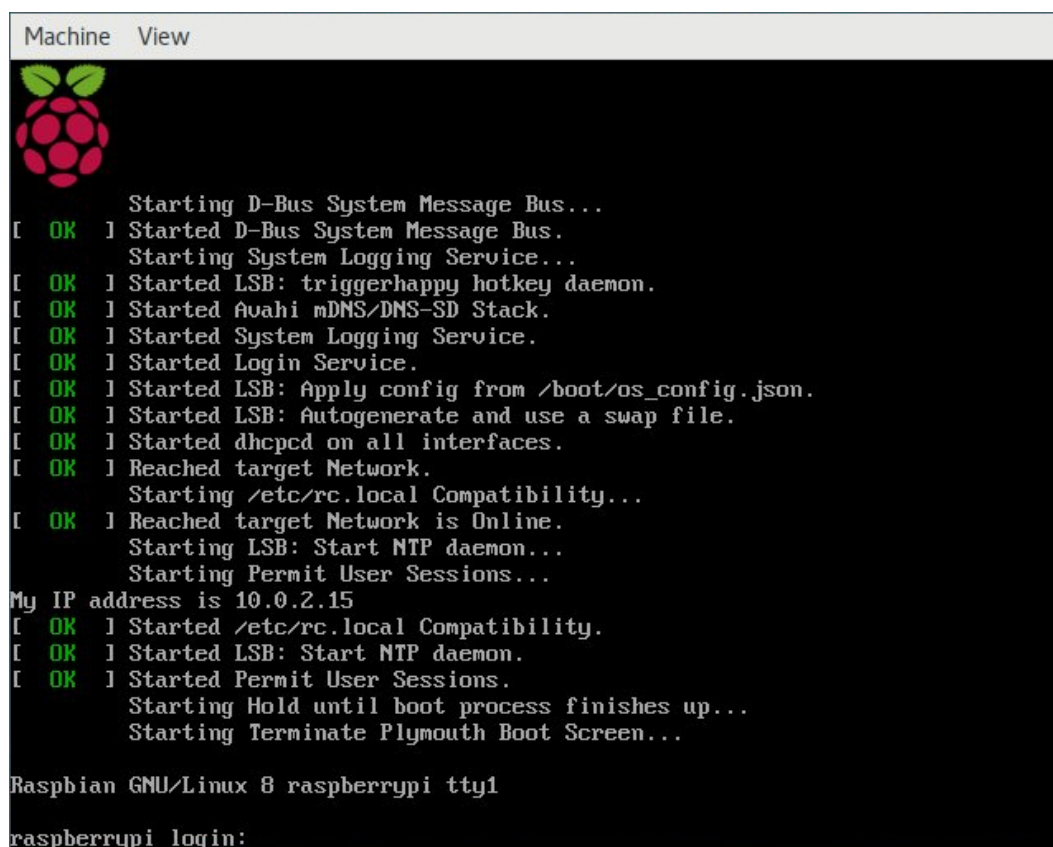
The image shows a terminal window titled 'Machine View' with the Raspberry Pi logo in the top left corner. The terminal displays the boot sequence of the Raspberry Pi OS. It starts with 'Starting D-Bus System Message Bus...' followed by several status messages in green text: '[OK] Started D-Bus System Message Bus.', 'Starting System Logging Service...', '[OK] Started LSB: triggerhappy hotkey daemon.', '[OK] Started Avahi mDNS/DNS-SD Stack.', '[OK] Started System Logging Service.', '[OK] Started Login Service.', '[OK] Started LSB: Apply config from /boot/os_config.json.', '[OK] Started LSB: Autogenerate and use a swap file.', '[OK] Started dhcpcd on all interfaces.', '[OK] Reached target Network.', 'Starting /etc/rc.local Compatibility...', '[OK] Reached target Network is Online.', 'Starting LSB: Start NTP daemon...', 'Starting Permit User Sessions...', 'My IP address is 10.0.2.15', '[OK] Started /etc/rc.local Compatibility.', '[OK] Started LSB: Start NTP daemon.', '[OK] Started Permit User Sessions.', 'Starting Hold until boot process finishes up...', and 'Starting Terminate Plymouth Boot Screen...'. At the bottom, it shows 'Raspbian GNU/Linux 8 raspberrypi tty1' and 'raspberrypi login: _'.

Figura 2: Arranque de Raspberry Pi OS

Cuando acabe la restauración, podemos insertar la microSD en la Raspberry para que pueda arrancar. Para ello, conectamos el cargador y debería aparecer la Figura 2.

Y, si todo ha ido bien, y dado que hemos instalado la versión con escritorio, nos debería aparecer directamente la pantalla de la Figura 3 con nuestro sistema ya funcionando.

Por último, comentar que, por defecto, Raspberry Pi OS establece los siguientes datos de acceso:

- Usuario: pi (que es *root*, por cierto).
- Contraseña: raspberry.

4. Introducción a la técnica PWM

Para empezar a trastear con la placa vamos a aprender a conectar y controlar un diodo emisor de luz (LED) usando la Raspberry Pi. Lo primero que hemos de tener en cuenta es que todos los pines GPIO de la Raspberry Pi son digitales. Por ello, a priori, la conexión de un LED a cualquier pin GPIO, solo nos va a permitir encenderlo o apagarlo. Ya veremos más en profundidad la electrónica que hay detrás de los puertos GPIO.

De modo que, para poder controlar el brillo del LED como si de una señal analógica se tratara, hemos de emplear algún artificio. Y ese artificio se llama PWM, cuyas siglas provienen de *Pulse Width Modulation*

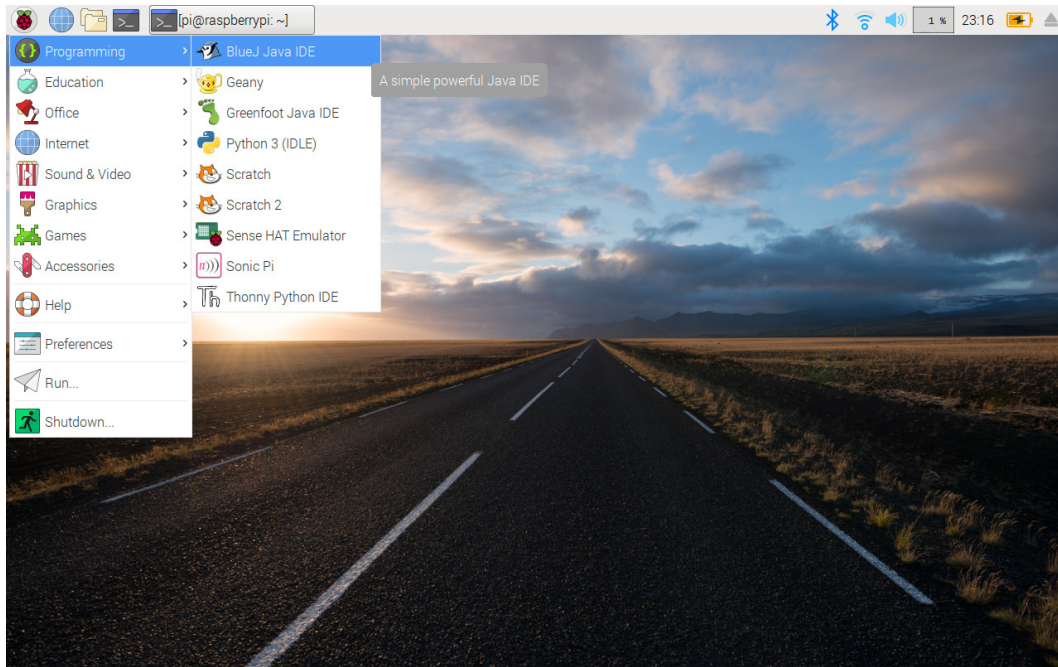


Figura 3: Aspecto del interfaz de escritorio de Raspberry Pi OS

o, en castellano, modulación de ancho de pulso. Si no empleamos este mecanismo, solo podríamos hacer dos cosas: encender (estado del pin HIGH) o apagar (LOW) el LED. Gracias al PWM podremos simular voltajes analógicos a través de los pines digitales. De modo que, considerando los pines GPIO de la Raspberry —que son de 3,3V— si por ejemplo queremos inyectar una señal de 1,65V, hemos de encender y apagar el pin de forma que este esté encendido la mitad del tiempo, y la otra mitad apagado. Asimismo, si queremos simular una salida analógica de 1,1V, deberíamos tomar tiempos para que la señal de 3,3V estuviera encendida el 33 % del tiempo. Como vemos, este enfoque es muy práctico para muchas aplicaciones como la que pretendemos: controlar el brillo de un LED.

Para hacer uso de este concepto, en Raspberry hemos de considerar la tensión como una señal con una frecuencia y un ciclo de trabajo. Así por ejemplo si consideramos una señal con una frecuencia de 100 Hz, tendría un período de 10 milisegundos o, en otras palabras, la señal se repite cada 10 milisegundos. Si la señal tuviera un ciclo de trabajo del 100 %, sería *Alta* el 100 % del tiempo y *Baja* el 0 % del tiempo. Si tuviera un ciclo de trabajo del 50 %, sería *Alta* el 50 % del tiempo (0.5×10 milisegundos = 5 milisegundos) y *Baja* el 50 % del tiempo (0.5×10 milisegundos = 5 milisegundos). Por lo que obtenemos que, con una señal de 100 Hz de frecuencia, durante un periodo total de 10 milisegundos, la señal sería *Alta* 5ms y *Baja* otros tantos.

5. Instalación del LED

Pasamos a instalar y usar el LED. Lo primero que debemos tener en cuenta es la polaridad de un LED. Si nos fijamos con el en la mano, vemos que hay un pin más corto que otro (Figura 4), y que además, en ese mismo lado el encapsulado del LED presenta una muesca o marca plana. Pues bien, este lado es el negativo (cátodo); por contra, el pin más largo (y sin muesca) es el ánodo o polo positivo.

Recordemos que el ánodo y cátodo de un dispositivo depende de si el dispositivo en cuestión consume o aporta energía. Siendo el ánodo el electrodo por el cual entra la energía al dispositivo; y el cátodo, por el que sale. Así, en una batería, en modo *aportar energía* (no en modo cargando), la energía sale del positivo y entra en el negativo; es por ello que el ánodo es el negativo y, el cátodo, el positivo. Por contra, en un diodo LED emisor de luz, que consume energía, la corriente entra por el positivo y sale por el negativo; por tanto, el ánodo es el positivo y, el cátodo, el negativo.

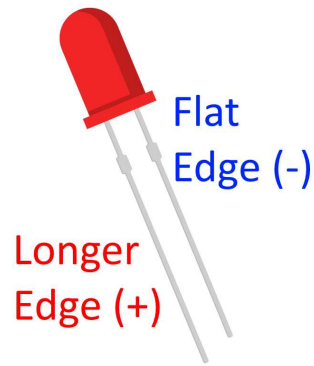


Figura 4: Conexión de un LED con la placa Raspberry Pi

En la Figura 5 vemos cómo quedaría la conexión del LED con la placa de Raspberry Pi. En este caso, hemos usado una resistencia, si bien se ha testado durante largos periodos la conexión de un LED sin esta y no se han detectado anomalías de funcionamiento.

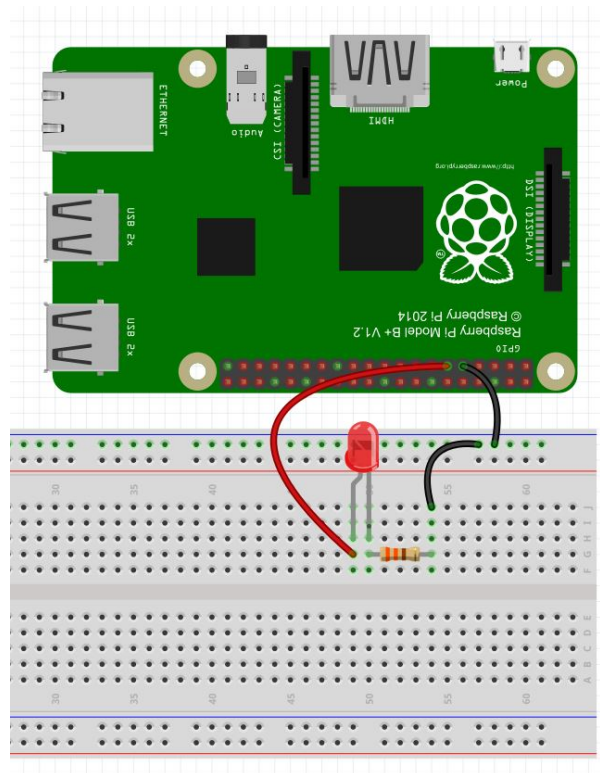


Figura 5: Conexión de un LED con la placa Raspberry Pi

Como se puede apreciar, estamos utilizando el pin físico 9 como toma de tierra (GND) y el pin físico 11 (o GPIO 17) como el pin de manejo (y alimentación 3,3V). Recuerda el diagrama de puertos GPIO que ya vimos en la Figura 1, y que más cómodamente puedes visualizar en cualquier momento en el Terminal de Raspberry Pi OS mediante el siguiente comando:

```
pinout
```


6. Manejo básico del LED mediante el Terminal

Podemos controlar los puertos GPIO sencillamente mediante comandos en el Terminal, concretamente, usando los ficheros de sistema existentes en la carpeta `/sys/class/gpio/` para tal efecto.

Lo primero que hemos de hacer es inicializar el fichero que manejará su correspondiente puerto GPIO. Por ejemplo, para manejar el pin físico 11, creamos el fichero correspondiente al GPIO 17 mediante el siguiente comando:

```
echo 17 > /sys/class/gpio/export
```

A continuación, especificamos si lo vamos a usar de salida o de entrada. En este caso, especificamos que va a ser de salida (out):

```
echo out > /sys/class/gpio/gpio17/direction
```

Y ya podemos usar el pin GPIO. Para enviar un pulso eléctrico y que, de este modo, se encienda el LED, lanzamos el siguiente comando:

```
echo 1 > /sys/class/gpio/gpio17/value
```

De forma similar, si queremos poner su estado a bajo o LOW, enviamos un 0:

```
echo 0 > /sys/class/gpio/gpio17/value
```

Y, por último, para resetear el comportamiento que hemos otorgado a este pin, borramos el fichero creado a tal efecto mediante el siguiente comando:

```
echo 17 > /sys/class/gpio/unexport
```

7. Programación del LED mediante PWM

Lo primero que debemos hacer siempre para poder usar los puertos GPIO de la placa Raspberry es importar la biblioteca RPi:

```
import RPi.GPIO as GPIO
```

A continuación lo que debemos hacer para evitar confusiones (muy importante) es indicar qué esquema de numeración de pin vamos a usar: numeración de pin físico (BOARD) o numeración según *Broadcom SOC channel* (BCM). En nuestro caso, por mera preferencia memorística, vamos a usar la primera opción:

```
GPIO.setmode (GPIO.BOARD)
```

En otro caso, deberíamos reemplazar BOARD por BCM en el comando anterior. El siguiente paso es indicarle que el pin que vamos a usar, el 11, va a ser de salida, pues no vamos a percibir nada del LED, sólo emitir. Para ello escribimos la siguiente instrucción:

```
GPIO.setup (11, GPIO.OUT)
```

En este punto ya podríamos escribir la señal en dicho pin en estado *Alto* o *Bajo*, pero nuestro objetivo es usar PWM para controlar el brillo del LED a nuestro antojo, para lo que necesitamos hacer una cosa más: crear un objeto PWM con el pin a usar y la frecuencia de trabajo como parámetros:

```
pwm = GPIO.PWM (11,100)
```

La frecuencia de 100 Hz es muy cómoda para trabajar con ella, pues el periodo resultante es de 10 ms, un número bastante redondo para hacer cálculos de forma sencilla.

El siguiente paso es establecer el ciclo de trabajo o *DutyCycle*. Este, por definición, es el porcentaje del período en que la señal está en estado *Alto*. Así, por ejemplo, si queremos aproximar una señal de 1.6 V, que es la mitad de los 3.3 V que aporta el pin de la Raspberry, necesitamos establecer un ciclo de trabajo del 50 %. Esto se hace mediante el siguiente comando:

```
pwm.start (50)
```

Ya con este comando se debería encender el LED con la mitad de brillo. Cambiando el ciclo de trabajo podemos cambiar este; por ejemplo, si queremos que la luz sea muy tenue, estableceremos un ciclo de trabajo del 1 % con el siguiente comando:

```
pwm.ChangeDutyCycle (1)
```

Por contra, si quiero que luzca en todo su esplendor, al brillo máximo, debería establecer un ciclo de trabajo del 100 %: `pwm.ChangeDutyCycle (100)`

También se puede cambiar la frecuencia de la señal PWM con `ChangeFrequency`. Así, por ejemplo, si quiero establecer una frecuencia de 1000 Hz, lo haré mediante la instrucción:

```
pwm.ChangeFrequency (1000)
```

Aunque en este caso no apreciaré un cambio en el brillo de la luz, sino simplemente va más rápido, pero esto no afecta el tiempo fraccional en que la señal está encendida y apagada, por lo tanto, el brillo del LED no cambia.

Para que estas instrucciones comandadas tengan efecto, hemos de darles tiempo para que se ejecuten. Esto lo conseguimos añadiendo cualquier instrucción que fuerce a parar la ejecución del programa; por ejemplo, la siguiente:

```
input("Ejecutando hasta que pulse una tecla...")
```

Por último, antes de acabar la ejecución del programa, hemos de desactivar el PWM con:

```
pwm.stop ()
```

Y también hacemos lo correspondiente con los puertos GPIO:

```
GPIO.cleanup ()
```