Conexión a Internet con Raspberry Pi Pico W

Conectar Raspberry Pi Pico W online con C/C++ o MicroPython

Colofón

fecha de construcción: 2024-05-02
versión de compilación: 576cee3-clean

Acerca del SDK

A lo largo del texto "el SDK" se refiere a nuestro SDK de Raspberry Pi Pico. Más detalles sobre el SDK pueden ser encontrados en el libro **Raspberry Pi Pico C/C++ SDK**. El código fuente incluido en la documentación es Copyright © 2020-2023 Raspberry Pi Ltd (anteriormente Raspberry Pi (Trading) Ltd.) y licenciado bajo la licencia 3- Clause BSD.

Aviso legal

LOS DATOS TÉCNICOS Y DE CONFIABILIDAD DE LOS PRODUCTOS RASPBERRY PI (INCLUYENDO LAS HOJAS DE DATOS), YA QUE SON MODIFICADOS DE VEZ EN CUANDO, (“RECURSOS”) SON PROPORCIONADOS POR RASPBERRY PI LTD (“RPL”) "TAL COMO ESTÁN" Y CUALQUIER GARANTÍA EXPRESA O IMPLÍCITA, INCLUYENDO, PERO NO LIMITADO A, LAS GARANTÍAS IMPLÍCITAS DE COMERCIABILIDAD E IDONEIDAD PARA UN PROPÓSITO PARTICULAR. EN LA MEDIDA MÁXIMA PERMITIDA POR LA LEY APLICABLE, EN NINGÚN CASO LA RPL SERÁ RESPONSABLE DE NINGÚN DAÑO DIRECTO, INDIRECTO, INCIDENTAL, ESPECIAL, EJEMPLAR O CONSECUENTE (INCLUYENDO, PERO NO LIMITADO A, LA ADQUISICIÓN DE BIENES O SERVICIOS SUSTITUTOS; PÉRDIDA DE USO, DATOS, O GANANCIAS; O INTERRUPCIÓN DEL NEGOCIO) NINGUNA CAUSA Y EN NINGUNA TEORÍA DE RESPONSABILIDAD, YA SEA POR CONTRATO, RESPONSABILIDAD ESTRICTA O AGRAVIO (INCLUYENDO NEGLIGENCIA O DE OTRA MANERA) QUE SURJA DE CUALQUIER MANERA DEL USO DE LOS RECURSOS, INCLUSO SI SE ADVIERTE DE LA POSIBILIDAD DE TAL DAÑO.

La RPL se reserva el derecho de realizar mejoras, actualizaciones, correcciones o cualquier otra modificación a los RECURSOS o a cualquier producto descrito en ellos en cualquier momento y sin previo aviso.

Los RECURSOS están destinados a usuarios expertos con niveles adecuados de conocimientos de diseño. Los usuarios son los únicos responsables de su selección y uso de los RECURSOS y de cualquier aplicación de los productos descritos en ellos. El usuario acepta indemnizar y eximir a la RPL de toda responsabilidad, costos, daños u otras pérdidas que surjan de su uso de los RECURSOS.

La RPL le cede a los usuarios permiso para utilizar los RECURSOS únicamente junto con los productos Raspberry Pi. Cualquier otro uso de los RECURSOS está prohibido. No se permite ninguna licencia a ningún otra RPL ni a ningún otro derecho de propiedad intelectual de terceros.

ACTIVIDADES DE ALTO RIESGO. Los productos Raspberry Pi no están diseñados, fabricados y destinados para ser usados en entornos peligrosos que requieran medidas de seguridad específicas, como en la operación de instalaciones nucleares, sistemas de comunicación o navegación de aeronaves, control de tráfico aéreo, sistemas de armas o aplicaciones críticas para la seguridad (incluyendo sistemas de soporte vital y otros dispositivos médicos), en los que la falla de los productos podría provocar directamente la muerte, lesiones personales o daños físicos o ambientales graves (“Actividades de alto riesgo”). La RPL renuncia específicamente a cualquier garantía expresa o implícita de idoneidad para actividades de alto riesgo y no acepta ninguna responsabilidad por el uso o la inclusión de productos Raspberry Pi en ACTIVIDADES DE ALTO RIESGO.

Los productos Raspberry Pi se proporcionan sujetos a los Términos estándar de la RPL. La provisión de RECURSOS por parte de la RPL no amplía ni modifica de ningún modo los Términos estándar, incluyendo pero no limitado a las exenciones de responsabilidad y garantías expresadas en ellos.

Tabla de contenido

Capítulo 1. Acerca de Raspberry Pi Pico W

Raspberry Pi Pico W es una placa de microcontrolador basada en el chip microcontrolador Raspberry Pi RP2040.

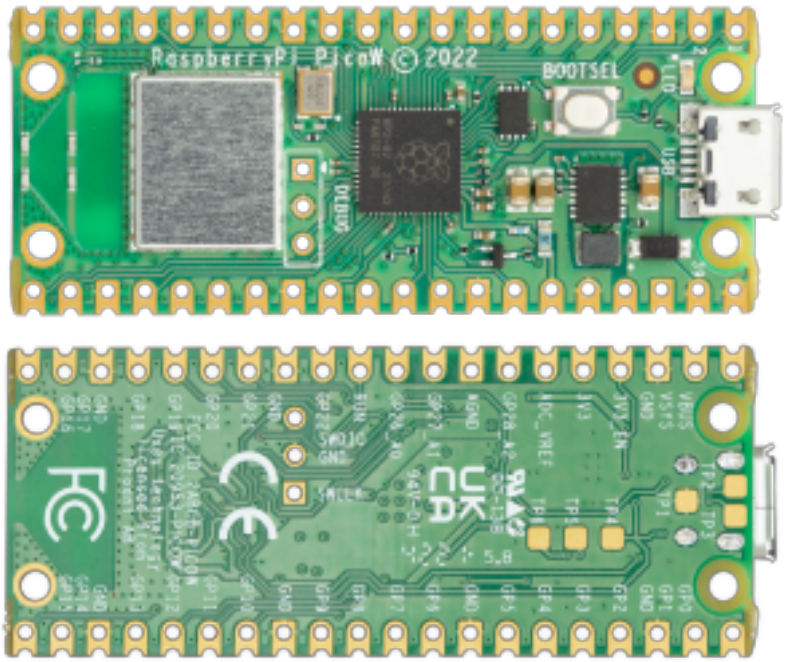


Figura 1. El Raspberry Pi Pico W Placa Rev3.

El Raspberry Pi Pico W ha sido diseñado para ser una plataforma de desarrollo flexible pero de bajo costo para RP2040, con una interfaz inalámbrica de 2,4 GHz y las siguientes características clave:

- Microcontrolador RP2040 con 2MB de memoria flash
- Interfaces inalámbricas integradas de banda única de 2,4 GHz (802.11n)
- Puerto micro USB B para alimentación y datos (y para reprogramar el flash)
- PCB estilo 'DIP' de 40 pines, 21 mm x 51 mm, 1 mm de espesor con pines de orificio pasante de 0,1"

  también con almenas en los bordes

  - Expone 26 E/S (GPIO) multifuncionales de propósito general de 3,3 V
  - 23 GPIO son sólo digitales, y tres también son compatibles con ADC
  - Se puede montar en superficie como módulo

Aparte de la incorporación de redes inalámbricas, Raspberry Pi Pico W es muy similar a Raspberry Pi Pico y, como todas las placas basadas en RP2040, comparte el mismo entorno de desarrollo. Si no ha utilizado anteriormente una placa basada en RP2040, puede comenzar leyendo Empezando con Raspberry Pi Pico si tiene la intención de utilizar nuestro C SDK, o SDK de Raspberry Pi Pico Python si está pensando en usar MicroPython.

NOTA

Los detalles completos de Raspberry Pi Pico W se pueden encontrar en la Hoja de datos de Raspberry Pi Pico W.

NOTA

Por defecto `libcyw43` tiene licencia para uso no comercial, pero los usuarios de Pico W y cualquier otra persona que construya su producto alrededor de RP2040 y CYW43439, se benefician de una licencia de uso comercial gratuita.

Capítulo 2. Conectarse a Internet con C SDK

Se ha agregado soporte inalámbrico para Raspberry Pi Pico W al SDK de C/C++.

NOTA

Si no ha utilizado anteriormente una placa basada en RP2040, puede comenzar leyendo Empezando con Raspberry Pi Pico, mientras que se pueden encontrar más detalles sobre el SDK junto con la documentación de nivel API en el libro Raspberry Pi Pico C/C++ SDK.

2.1. Instalación del SDK y ejemplos

Para obtener instrucciones completas sobre cómo comenzar con el SDK, consulte el libro Empezando con Raspberry Pi Pico.

```
 $ git clon https://github.com/raspberrypi/pico-sdk.git --branch master
$ cd pico-sdk
$ actualización del submódulo git --init
$cd..
 $ git clon https://github.com/raspberrypi/pico-examples.git --branch master
```

ADVERTENCIA

Si no ha inicializado el `tinyusb` submódulo en su `pico-sdk` finalizar la compra, luego el USB CDC serial y otras funciones USB y el código de ejemplo no funcionarán ya que el SDK no contendrá ninguna funcionalidad USB. Del mismo modo, si no ha inicializado el `cyw43-driver` y `lwip` submódulos en su pago, entonces la funcionalidad relacionada con la red no estará habilitada.

2.2. Construyendo un ejemplo de SDK

Para crear los ejemplos de SDK y otros códigos inalámbricos, debe especificar el SSID y la contraseña de su red, de esta manera:

```
$ cd pico-examples
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake -DPICO_BOARD=pico_w -DWIFI_SSID="Your Network" -DWIFI_PASSWORD="Your Password" ..
Using PICO_SDK_PATH from environment ('../../pico-sdk')
PICO_SDK_PATH is /home/pi/pico/pico-sdk
  .
  .
  .
-- Build files have been written to: /home/pi/pico/pico-examples/build
$
```

NOTA

Las banderas de la línea de comando `-DWIFI_SSID="Your Network" -DWIFI_PASSWORD="Your Password"` son utilizados por el `pico-examples` para configurar el SSID y la contraseña de la llamada a `cyw43_arch_wifi_connect_xxx()` para conectarse a su red inalámbrica.

Luego, para crear un ejemplo básico para Raspberry Pi Pico W que buscará redes inalámbricas cercanas, puede hacer lo siguiente:

```
$ cd pico-examples/pico_w/wifi/wifi_scan
$ make
PICO_SDK_PATH is /home/pi/pico-sdk
PICO platform is rp2040.
Build type is Release
PICO target board is pico_w.
  .
  .
  .
[100%] Built target picow_scan_test_background
$
```

Junto con otros objetivos, ahora hemos creado un binario llamado `picow_scan_test_background.uf2`, que se puede arrastrar al dispositivo de almacenamiento masivo USB RP2040.

Este binario buscará redes inalámbricas utilizando el chip inalámbrico de Raspberry Pi Pico W.

El método más rápido para cargar software en una placa basada en RP2040 por primera vez es montarla como un dispositivo de almacenamiento masivo USB. Hacer esto le permite arrastrar un archivo al tablero para programar el flash. Continúe y conecte la Raspberry Pi Pico W a su Raspberry Pi usando un cable micro-USB, asegurándose de mantener presionado el botón BOOTSEL mientras lo hace, para forzarlo al modo de almacenamiento masivo USB.

Si está ejecutando Raspberry Pi Desktop, Raspberry Pi Pico W debería montarse automáticamente como un dispositivo de almacenamiento masivo USB. Desde aquí, puede arrastrar y soltar el archivo UF2 en el dispositivo de almacenamiento masivo. RP2040 se reiniciará, se desmontará como dispositivo de almacenamiento masivo y comenzará a ejecutar el código actualizado.

De forma predeterminada, el código informará sus resultados a través de serie UART.

IMPORTANTE

Los pines UART predeterminados se configuran por placa utilizando archivos de configuración de placa. El pin predeterminado Raspberry Pi Pico W UART TX (saliendo de Pico W) es el pin GP0, y el pin UART RX (entrando a Pico W) es el pin GP1.

Para ver el texto, deberá habilitar las comunicaciones serie UART en el host Raspberry Pi. Para hacerlo, ejecute `raspi config`:

```
$ sudo raspi-config
```

E ir a `Interfacing Options → Serial` y seleccione "No" cuando se le pregunte "¿Le gustaría que se pueda acceder a un shell de inicio de sesión a través de serie?", luego "Sí" cuando se le pregunte "¿Le gustaría que el hardware del puerto serie esté habilitado?" Debería ver algo como Figura 2.

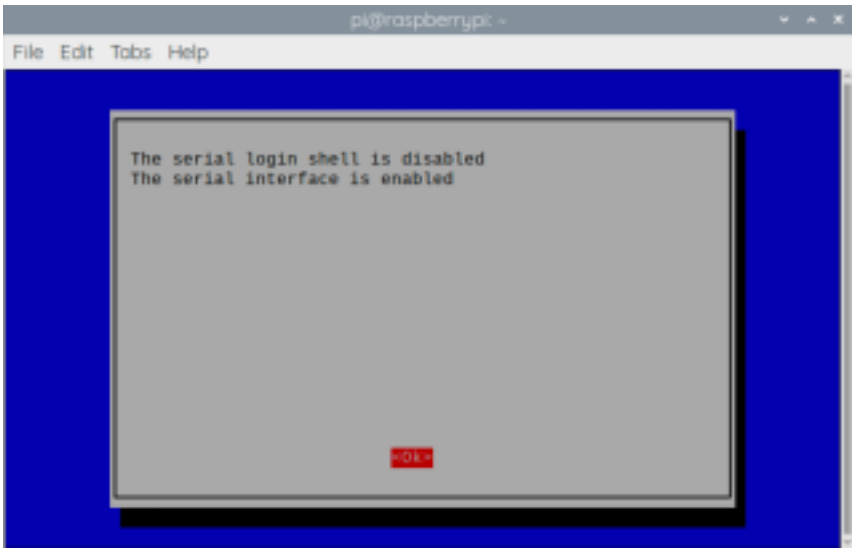


Figura 2. Habilitación de una serie UART usando raspi config sobre el Raspberry Pi.

Partida raspi config debe elegir "Sí" y reiniciar tu Raspberry Pi para habilitar el puerto serie.

Luego debe conectar la Raspberry Pi y la Raspberry Pi Pico W junto con el siguiente mapeo:

Raspberry Pi	Raspberry Pi Pico W
TIERRA (Pin 14)	TIERRA (Pin 3)
GPIO15 (UART_RX0, clavija 10)	GP0 (UART0_TX, clavija 1)
GPIO14 (UART_TX0, clavija 8)	GP1 (UART0_RX, clavija 2)

Ver figura 3.

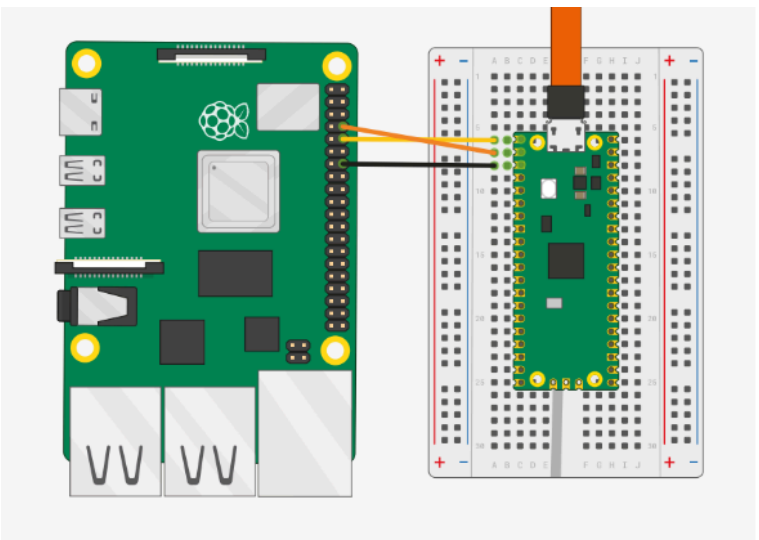


Figura 3. Una Raspberry Pi 4 y la Raspberry Pi Pico con UART0 conectados entre sí.

Una vez las dos placas están conectadas entre sí, debe instalar minicom si aún no lo has hecho:

```
$ sudo apt install minicom
```

y abre el puerto serie:

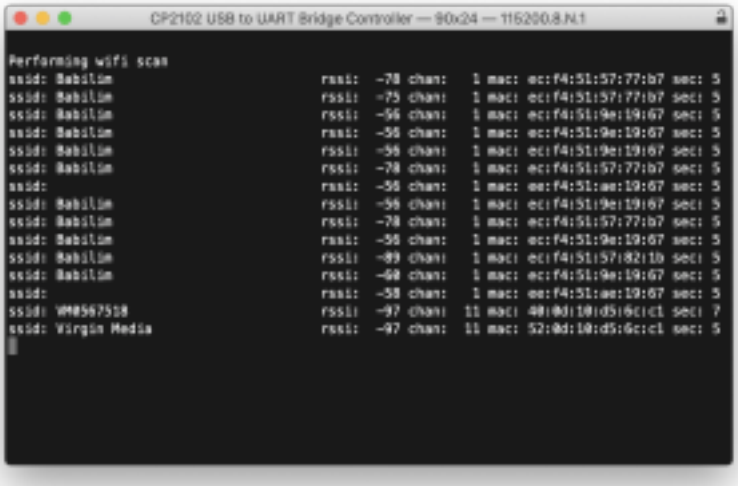
```
$ minicom -b 115200 -o -D /dev/serial0
```

Deberías ver los resultados de nuestro escaneo inalámbrico impresos en la consola, consulta Figura 4.

CONSEJO

Para salir de minicom, utilice CTRL-A seguido por X.

Figura 4. Resultados de nuestro escaneo inalámbrico en la consola



2.3. Creando tu propio proyecto

Continúe y cree un directorio para albergar su proyecto de prueba junto al pico-sdk directorio,

```
$ ls -la
total 16
drwxr-xr-x 7 aa staff 224 6 Apr 10:41 ./
drwx------@ 27 aa staff 864 6 Apr 10:41 ../
drwxr-xr-x 10 aa staff 320 6 Apr 09:29 pico-examples/
drwxr-xr-x 13 aa staff 416 6 Apr 09:22 pico-sdk/
$ mkdir test
$ cd test
```

y luego crear un prueba.c archivo en el directorio,

```
1 #incluye <stdio.h>
 2
 3 #incluye "pico/stdlib.h"
 4 #incluye "pico/cyw43_arch.h"
 5
 6 char ssid[] = "A Network";①
 7 char pass[] = "A Password";②
 8
 9 int main() {
10 stdio_init_all();
11
```

```
12 if (cyw43_arch_init_with_country(CYW43_COUNTRY_UK)) {
13 printf("failed to initialise\n");
14 return 1;
15 }
16 printf("initialised\n");
17
18 cyw43_arch_enable_sta_mode();
19
20 if (cyw43_arch_wifi_connect_timeout_ms(ssid, pass, CYW43_AUTH_WPA2_AES_PSK, 10000)) { 21
printf("failed to connect\n");
22 return 1;
23 }
24 printf("connected\n");
25 }
```

1. Reemplace A Network con el nombre SSID de su red.
2. Reemplace A Password con la contraseña de dicha red.

junto con el archivo CMakeLists.txt,

```
cmake_minimum_required(VERSION 3.13)

include(pico_sdk_import.cmake)

project(test_project C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
pico_sdk_init()

add_executable(test
  test.c
)

pico_enable_stdio_usb(test 1)
pico_enable_stdio_uart(test 1)

pico_add_extra_outputs(test)

target_include_directories(test PRIVATE ${CMAKE_CURRENT_LIST_DIR} )

target_link_libraries(test pico_cyw43_arch_lwip_threadsafe_background pico_stdlib)
```

Then copy the pico_sdk_import.cmake file from the external folder in your pico-sdk installation to your test project

folder, `$ cp ../pico-sdk/external/pico_sdk_import.cmake .`

along with the lwipopts.h file needed by the lwIP stack.

```
$ cp ../pico-examples/pico_w/wifi/lwipopts_examples_common.h lwipopts.h
```

You should now have something that looks like this,

```
$ ls -la
total 32
drwxr-xr-x 6 aa staff 192B 29 Jun 18:11 ./
drwxr-xr-x 7 aa staff 224B 29 Jun 16:57 ../
-rw-r--r--@ 1 aa staff 379B 29 Jun 18:10 CMakeLists.txt
-rw-rw-r--@ 1 aa staff 3.3K 15 Jun 00:34 lwipopts.h
-rw-rw-r--@ 1 aa staff 3.1K 15 Jun 00:34 pico_sdk_import.cmake
-rw-r--r--@ 1 aa staff 427B 29 Jun 17:03 test.c
```

and can build it as we did before with our previous example in the last section.

```
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake -DPICO_BOARD=pico_w ..
$ make
```

Luego, desconecte su Raspberry Pi Pico W de su ordenador si ya está enchufada. Luego presione y mantenga presionado el botón BOOTSEL mientras lo vuelve a conectar a su ordenador. Luego arrastre y suelte el `test.uf2` binario en el volumen de almacenamiento masivo RPI-RP2 que se montará en su escritorio.

Abra el puerto serie:

```
$$ minicom -b 115200 -o -D /dev/serial0
```

y debería ver un mensaje que indica que el Pico W se ha conectado a su red inalámbrica.

2.3.1. Ir más lejos

Puede encontrar más información sobre C SDK en el libro Raspberry Pi Pico C/C++ SDK. Si bien se puede encontrar información sobre lwIP en el sitio web del proyecto. El código de ejemplo se puede encontrar como parte del pico-examples Repositorio Github. Para obtener ejemplos de C adicionales para Raspberry Pi Pico W, consulte ejemplos de redes.

2.4. ¿En qué hardware estoy ejecutando?

No existe ningún método directo en el SDK de C al que se pueda llamar para permitir que el software descubra si se está ejecutando en una Raspberry Pi Pico o en una Pico W. Sin embargo, es posible descubrir indirectamente el tipo de hardware subyacente. Si la placa se alimenta mediante USB o `VSYS`, entonces `3v3_ES` no está bajo externamente, con GPIO25 bajo, ADC3 estará alrededor de 0V para Raspberry Pi Pico W y aproximadamente 1/3 de `VSYS` Para Raspberry Pi Pico.

Creando un `test.c` archivo,

```
1 #incluye <stdio.h>
2
3 #incluye "pico/stdlib.h"
4 #incluye "hardware/gpio.h"
```

```
 5 #incluye "hardware/adc.h"
 6
 7 int main() {
 8 stdio_init_all();
 9
10 adc_init();
11 adc_gpio_init(29);
12 adc_select_input(3);
13 const float conversion_factor = 3.3f / (1 << 12);
14 uint16_t result = adc_read();
15 printf("ADC3 value: 0x%03x, voltage: %f V\n", result, result * conversion_factor); 16
17 gpio_init(25);
18 gpio_set_dir(25, GPIO_IN);
19 uint value = gpio_get(25);
20 printf("GP25 value: %i", value);
21 }
```

alongside the following CMakeLists.txt file,

```
cmake_minimum_required(VERSION 3.13)

include(pico_sdk_import.cmake)

project(test_project C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
pico_sdk_init()

add_executable(test
  test.c
)

target_link_libraries(test pico_stdlib hardware_adc hardware_gpio)

pico_enable_stdio_usb(test 1)
pico_enable_stdio_uart(test 1)

pico_add_extra_outputs(test)
```

en un directorio de proyectos se proporciona esto para Raspberry Pi Pico W,

```
ADC3 value: 0x01c, voltage: 0.022559 V
GP25 value: 0
```

y esto para una placa Raspberry Pi Pico original,

```
ADC3 value: 0x2cd, voltage: 0.577661 V
GP25 value: 0
```

Capítulo 3. Conectarse a Internet con MicroPython

Se ha agregado soporte inalámbrico para Raspberry Pi Pico W a MicroPython. Un binario prediseñado, que se puede descargar desde la sección MicroPython a través del sitio web de la sitio web documentación, debería servir para la mayoría de los casos de uso y viene con la librería `micropython lib` pre-integrada en el binario.

NOTA

Si no ha utilizado anteriormente una placa basada en RP2040, puede comenzar leyendo el libro SDK de Raspberry Pi Pico Python.

3.1. Obteniendo MicroPython para Raspberry Pi Pico W

Binario prediseñado

Un binario prediseñado del último firmware de MicroPython está disponible en sección MicroPython del sitio de documentación.

La forma más rápida de obtener MicroPython es descargar el binario de versión prediseñado desde la página web de Documentación. Si no puede o no quiere utilizar la versión prediseñada (por ejemplo, si desea desarrollar un módulo C para MicroPython), puede seguir las instrucciones en Apéndice A> para obtener el código fuente de MicroPython, que puede utilizar para crear su propio firmware binario de MicroPython.

3.2. Instalación de MicroPython en Raspberry Pi Pico W

Raspberry Pi Pico W tiene un modo BOOTSEL para programar firmware a través del puerto USB. Si mantiene presionado el botón BOOTSEL cuando encienda su placa, ésta entrará en un modo especial donde aparecerá como un dispositivo de almacenamiento masivo USB. Primero asegúrese de que su Raspberry Pi Pico W no esté enchufado a ninguna fuente de alimentación: desconecte el cable micro USB si está enchufado y desconecte cualquier otro cable que pueda estar suministrando energía a la placa, por ejemplo. a través del pin VSYS o VBUS. Ahora mantenga presionado el botón BOOTSEL y conecte el cable micro USB (que en ideal tiene el otro extremo conectado a su ordenador).

Debería aparecer una unidad llamada RPI-RP2. Continúe y arrastre MicroPython `firmware.uf2` archivo en esta unidad. Esto programa el firmware MicroPython en la memoria flash de su Raspberry Pi Pico W.

Debería tomar unos segundos programar el archivo UF2 en la memoria flash. La placa se reiniciará automáticamente cuando termine, lo que hará que la unidad RPI-RP2 desaparezca y se inicie en MicroPython.

Cuando MicroPython arranque por primera vez, se sentará y esperará a que usted se conecte y le diga qué hacer. Puede crear un `.py` archivo desde tu ordenador a la pizarra, pero una forma más inmediata de interactuar con él es a través de lo que se llama *leer el bucle evaluar-imprimir* REPL.

Hay dos formas de conectarse a este REPL; para que pueda comunicarse con el firmware MicroPython de su placa a través de USB o a través del puerto serie UART en los GPIO Raspberry Pi Pico W.

NOTA

El puerto MicroPython para RP2040 no expone un REPL a través de un puerto UART de forma predeterminada; consulte SDK de Raspberry Pi Pico Python para obtener más detalles sobre cómo configurar MicroPython para permitirle conectarse al REPL a través de UART.

3.3. Conexión de un Raspberry Pi a través de USB

El firmware MicroPython está equipado con un puerto serie USB virtual al que se accede a través del conector micro USB en Raspberry Pi Pico W. Su ordenador debería notar este puerto serie y nombrarlo como un dispositivo de caracteres, muy probablemente /dev/ttyACM0.

CONSEJO

Puede ejecutar la orden ls /dev/tty* para enumerar sus puertos serie. Puede que haya bastantes, pero el serial USB de MicroPython comenzará con /dev/ttyACM. En caso de duda, desconecta el conector micro USB y observa cuál desaparece. Si no ve nada, puede intentar reiniciar su Raspberry Pi.

puede instalar minicom para acceder al puerto serie:

```
$ sudo apt install minicom
```

y luego abrirlo como tal:

```
$ minicom -o -D /dev/ttyACM0
```

Donde el -D /dev/ttyACM0 está apuntando minicom en el puerto serie USB de MicroPython, y el -o flag significa esencialmente "simplemente hazlo". No hay necesidad de preocuparse por la velocidad, ya que se trata de un puerto serie virtual.

Presione la tecla Intro varias veces en la terminal donde abrió minicom debería ver esto:

```
>>>
```

Esto es *inmediato*. MicroPython quiere que escribas algo y le digas qué hacer.

Si presiona CTRL-D en su teclado mientras el minicom terminal está enfocado, debería ver un mensaje similar a este:

```
MPY: soft reboot
MicroPython v1.18-524-g22474d25d on 2022-05-25; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```

Esta combinación de teclas le indica a MicroPython que se reinicie. Puede hacer esto en cualquier momento. Cuando se reinicie, MicroPython imprimirá un mensaje que indica exactamente qué versión de firmware está ejecutando y cuándo se creó. Su número de versión será diferente al que se muestra aquí.

NOTA

Si está trabajando en un Mac de Apple, siempre que utilice una versión reciente de macOS como Catalina, los controladores ya deberían estar cargados. De lo contrario, consulte el sitio web del fabricante para Controladores de chips FTDI. Luego debería usar el programa Terminal para conectarse a Serial-over-USB (USB CDC). El puerto serie se mostrará como /dev/tty.usbmodem con un número añadido al final.

Si aún no tiene instalado el programa Terminal, puede instalarlo minicom usando HomeBrew:

```
$ brew install minicom
```

y conéctelo a la placa como se muestra a continuación.

```
$ minicom -b 115200 -o -D /dev/tty.usbmodem0000000000001
```

Otras aplicaciones de Terminal como CoolTerm o Serial también pueden ser usadas.

3.3.1. Utilizar un entorno de desarrollo integrado (IDE)

El puerto MicroPython para Raspberry Pi Pico W y otras placas basadas en RP2040 funciona con entornos de desarrollo de uso común. Thonny es el editor recomendado. Los paquetes Thonny están disponibles para Linux, MS Windows y macOS. Después de la instalación, el uso del entorno de desarrollo de Thonny es el mismo en las tres plataformas. La última versión de Thonny se puede descargar desde thonny.org.

Para obtener detalles completos sobre cómo utilizar el editor Thonny, consulte la sección sobre utilizando un entorno de desarrollo en el libro SDK de Raspberry Pi Pico Python.

3.3.2. Acceso mpremote a través del puerto serie

Se sugiere utilizar la herramienta mpremote para acceder al dispositivo a través del puerto serie.

```
$ pip install mpremote
$ mpremote connect list
/dev/cu.Bluetooth-Incoming-Port None 0000:0000 None None
/dev/cu.usbmodem22201 e660583883807e27 2e8a:0005 MicroPython Board in FS mode
$ mpremote connect port:/dev/cu.usbmodem22201
Connected to MicroPython at /dev/cu.usbmodem22201
Use Ctrl-] to exit this shell
>>>
```

Con esto puede ejecutar un script desde tu máquina local directamente en Raspberry Pi Pico W.

```
$ mpremote connect port:/dev/cu.usbmodem22201
$ mpremote run hello_world.py
```

NOTA

Para más información sobre mpremote ver la documentación.

3.4. El LED a bordo

A diferencia del Raspberry Pi Pico original, el LED integrado en Pico W no está conectado a un pin en el RP2040, sino a un pin GPIO en el chip inalámbrico. MicroPython se ha modificado en consecuencia. Esto significa que ahora puede hacer:

```
>>> import machine
>>> led = machine.Pin("LED", machine.Pin.OUT)
>>> led.off()
>>> led.on()
```

o incluso:

```
>>> led.toggle()
```

para cambiar el estado actual. Sin embargo, si ahora nos fijamos en el objeto led:

```
>>> led
Pin(WL_GPIO0, mode=OUT)
>>>
```

También puede hacer lo siguiente:

```
>>> led = machine.Pin("LED", machine.Pin.OUT, value=1)
```

que configurará el led objeto, asócielo con el LED integrado **y** enciende el LED.

NOTA

Los detalles completos de Raspberry Pi Pico W se pueden encontrar en la Hoja de datos de Raspberry Pi Pico W. WL_GPIO1 está conectado al PS/SYNC pin en el RT6154A para permitir la selección de diferentes modos de funcionamiento, mientras WL_GPIO2 Se puede utilizar para monitorear USB. VBUS.

3.5. Instalación de módulos

Puede usar la herramienta upip para instalar módulos que no están presentes en la instalación predeterminada de MicroPython.

```
>>> import upip
>>> upip.install("micropython-pystone_lowmem")
>>> import pystone_lowmem
>>> pystone_lowmem.main()
Pystone(1.2) time for 500 passes = 402ms
This machine benchmarks at 1243 pystones/second
```

```
>>>
```

3.6. Conexión a una red inalámbrica

Estamos usando la librería network para hablar con el hardware inalámbrico:

```
import network
import time

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('Wireless Network', 'The Password')

while not wlan.isconnected() and wlan.status() >= 0:
print("Waiting to connect:")
time.sleep(1)

print(wlan.ifconfig())
```

Aunque lo más correcto es esperar a que la conexión se realice correctamente o falle en su código y manejar cualquier error de conexión que pueda ocurrir.

```
import time
import network

ssid = 'Wireless Network'
password = 'The Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

# Espere un fallo o conexión
max_wait = 10
while max_wait > 0:
if wlan.status() < 0 or wlan.status() >= 3:
break
max_wait -= 1
print('waiting for connection...')
time.sleep(1)

# Maneja error de conexión
if wlan.status() != 3:
raise RuntimeError('network connection failed')
else:
print('connected')
status = wlan.ifconfig()
print( 'ip = ' + status[0] )
```

También puede desconectarse y luego conectarse a una red inalámbrica diferente.

```
# Conéctese a otra red inalámbrica
wlan.disconnect();
wlan.connect('Other Network', 'The Other Password')
```

Para más información sobre la librería network.WLAN ver la documentación de la librería.

3.6.1. Códigos de estado de conexión

Los valores devueltos por el wlan.status() Las llamadas se definen en el controlador inalámbrico CYW43 y se pasan directamente al código de usuario.

```
// Valor devuelto de cyw43_wifi_link_status
#define CYW43_LINK_DOWN (0)
#define CYW43_LINK_JOIN (1)
#define CYW43_LINK_NOIP (2)
#define CYW43_LINK_UP (3)
#define CYW43_LINK_FAIL (-1)
#define CYW43_LINK_NONET (-2)
#define CYW43_LINK_BADAUTH (-3)
```

3.6.2. Configurando el país

De forma predeterminada, la configuración de país para la red inalámbrica no está configurada. Esto significa que el conductor utilizará una configuración segura mundial predeterminada, lo que puede significar que algunos canales no estén disponibles.

```
>>> import rp2
>>> rp2.country()
'\x00\x00'
>>>
```

Esto puede causar problemas en algunas redes inalámbricas. Si descubre que su Raspberry Pi Pico W no se conecta a su red inalámbrica, puede intentar configurar el código de país, por ejemplo.

```
>>> rp2.country('GB')
```

3.6.3. Modo de ahorro de energía

De forma predeterminada, el chip inalámbrico activará el modo de ahorro de energía cuando esté inactivo, lo que podría hacer que responda menos. Si está ejecutando un servidor o necesita más capacidad de respuesta, puede cambiar esto alternando el modo de energía.

```
import network

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.config(pm = 0xa11140)
```

3.7. La dirección MAC

La MAC se almacena en el chip inalámbrico OTP.

```
import network
import ubinascii

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
mac = ubinascii.hexlify(network.WLAN().config('mac'),':').decode()
print(mac)

# Otras cosas que puede probar
print(wlan.config('channel'))
print(wlan.config('essid'))
print(wlan.config('txpower'))
```

📌 **NOTA** Tenemos que configurar la conexión inalámbrica activa (que carga el firmware) antes de poder obtener la dirección MAC.

3.8. Realizar solicitudes HTTP

Puede adoptar un enfoque de bajo nivel para las solicitudes HTTP utilizando sockets sin formato, o un enfoque de alto nivel utilizando la librería urequests.

3.8.1. HTTP con sockets

```
# Conéctese a una red
import network

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('Wireless Network', 'The Password')

# Debería estar conectado y poseer una dirección de IP
wlan.status() # 3 == éxito
wlan.ifconfig()

# Obenga la dirección de IP para google.com
import socket
ai = socket.getaddrinfo("google.com", 80)
addr = ai[0][-1]

# Crea un socket y una solicitud HTTP
s = socket.socket()
s.connect(addr)
s.send(b"GET / HTTP/1.0\r\n\r\n")

# Imprima la respuesta
print(s.recv(512))
```

3.8.2. HTTP con urequests

Es mucho más sencillo utilizar la librería urequests para establecer una conexión HTTP.

```
# Conéctese a una red
import network
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('Wireless Network', 'The Password')

# Crea una solicitud GET
import urequests
r = urequests.get("http://www.google.com")
print(r.content)
r.close()
```

Support has been added for redirects.

```
import urequests
r = urequests.get("http://www.raspberrypi.com")
print(r.status_code) # directa a https
r.close()
```

NOTA

HTTPS funciona, pero debe tener en cuenta que la verificación SSL está actualmente deshabilitada.

La librería urequests viene con soporte JSON limitado.

```
>>> r = urequests.get("http://date.jsontest.com")
>>> r.json()
{'milliseconds_since_epoch': 1652188199441, 'date': '05-10-2022', 'time': '01:09:59
PM'} >>>>
```

Para más información sobre urequests ver la documentación de la librería.

IMPORTANTE

Debe cerrar el objeto de respuesta devuelto después de realizar una solicitud utilizando la librería urequests usando response.close(). Si no lo hace, el objeto no se recolectará como basura y, si la solicitud se realiza dentro de un bucle, esto provocará rápidamente un bloqueo.

3.8.3. Garantizar conexiones robustas

Este ejemplo parcial ilustra un enfoque más sólido para conectarse a una red utilizando urequests.

```
import time
import network
import urequests as requests

ssid = 'A Network'
```

```
password = 'The Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

# Espere fallo o conexión
max_wait = 10
while max_wait > 0:
if wlan.status() < 0 or wlan.status() >= 3:
break
max_wait -= 1
print('waiting for connection...')
time.sleep(1)

# Maneja error de conexión
if wlan.status() != 3:
raise RuntimeError('network connection failed')
else:
print('connected')
status = wlan.ifconfig()
print( 'ip = ' + status[0] )

while True:

# Aquí, quizá podría medir algo con algún sensor?

# ...y luego defina los headers y payload
headers = ...
payload = ...

# E intente ejecutarlo con el bloque try/except
try:
print("sending...")
response = requests.post("A REMOTE END POINT", headers=headers, data=payload)
print("sent (" + str(response.status_code) + "), status = " + str(wlan.status()) )
response.close()
except:
print("could not connect (status =" + str(wlan.status()) + ")")
if wlan.status() < 0 or wlan.status() >= 3:
print("trying to reconnect...")
wlan.disconnect()
wlan.connect(ssid, password)
if wlan.status() == 3:
print('connected')
else:
print('failed')

time.sleep(5)
```

Aquí manejamos la posibilidad de que perdamos la conexión a nuestra red inalámbrica y luego buscaremos volver a conectarnos.

3.9. Construyendo servidores HTTP

Puede crear servidores web sincrónicos o asincrónicos.

3.9.1. Un servidor sencillo para páginas estáticas.

Puede usar la librería socket para construir un servidor web simple.

```
import network
import socket
import time

from machine import Pin

led = Pin(15, Pin.OUT)

ssid = 'A Network'
password = 'A Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

html = """<!DOCTYPE html>
<html>
<head> <title>Pico W</title> </head>
<body> <h1>Pico W</h1>
<p>Hello World</p>
</body>
</html>
"""

# Espere fallo o conexión
max_wait = 10
while max_wait > 0:
if wlan.status() < 0 or wlan.status() >= 3:
break
max_wait -= 1
print('waiting for connection...')
time.sleep(1)

# Maneja error de conexión
if wlan.status() != 3:
raise RuntimeError('network connection failed')
else:
print('connected')
status = wlan.ifconfig()
print( 'ip = ' + status[0] )

# Abra el socket
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

s = socket.socket()
s.bind(addr)
s.listen(1)

print('listening on', addr)

# Preste atención a conexiones
while True:
try:
```

```
54 cl, addr = s.accept()
55 print('client connected from', addr)
56 cl_file = cl.makefile('rwb', 0)
57 while True:
58 line = cl_file.readline()
```



```
59 if not line or line == b'\r\n':
60 break
61 response = html
       62 cl.send('HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n')
63 cl.send(response)
64 cl.close()
65
66 except OSError as e:
67 cl.close()
68 print('connection closed')
```

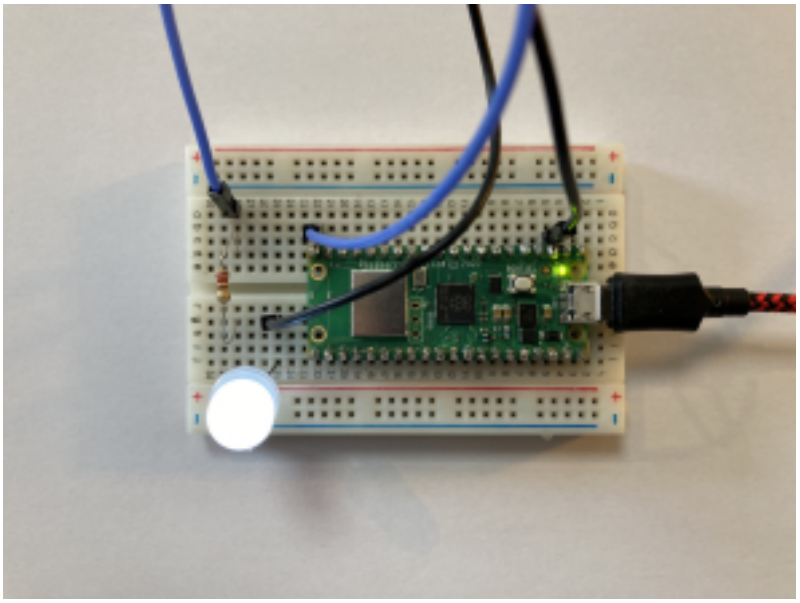
NOTA

Este ejemplo es sincrónico; para un manejo de solicitudes más sólido, debe implementar el servidor para manejar las solicitudes de forma asincrónica.

3.9.2. Controlar un LED a través de un servidor web

Yendo más allá, podemos implementar un servidor web RESTful que nos permitirá controlar un LED.

Figura 5. El Raspberry Pi Pico W con un LED en GP15.



Conectando un LED a GP15 podemos encender y apagar el LED usando HTTP GET. Podemos hacer esto yendo a http://192.168.1.X/light/on para encender el LED y http://192.168.1.X/light/off apagar el LED, en nuestro navegador web; dónde 192.168.1.X es la dirección IP de nuestro Pico W, la cual quedará impresa en la consola luego de que se conecte a la red.

```
1 import network
2 import socket
3 import time
4
5 from machine import Pin
6
7 led = Pin(15, Pin.OUT)
8
```

```
ssid = 'A Network'
password = 'A Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

html = """<!DOCTYPE html>
<html>
<head> <title>Pico W</title> </head>
<body> <h1>Pico W</h1>
<p>%s</p>
</body>
</html>
"""

# Espere fallo o conexión
max_wait = 10
while max_wait > 0:
if wlan.status() < 0 or wlan.status() >= 3:
break
max_wait -= 1
print('waiting for connection...')
time.sleep(1)

# Maneja error de conexión
if wlan.status() != 3:
raise RuntimeError('network connection failed')
else:
print('connected')
status = wlan.ifconfig()
print( 'ip = ' + status[0] )

# Abra el socket
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

s = socket.socket()
s.bind(addr)
s.listen(1)

print('listening on', addr)

# Listen for connections
while True:
try:
cl, addr = s.accept()
print('client connected from', addr)
request = cl.recv(1024)
print(request)

request = str(request)
led_on = request.find('/light/on')
led_off = request.find('/light/off')
print( 'led on = ' + str(led_on))
print( 'led off = ' + str(led_off))

if led_on == 6:
print("led on")
led.value(1)
stateis = "LED is ON"

if led_off == 6:
print("led off")
led.value(0)
```

```
73 stateis = "LED is OFF"
74
75 response = html % stateis
76
77 cl.send('HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n')
```



```
78 cl.send(response)
79 cl.close()
80
81 except OSError as e:
82 cl.close()
83 print('connection closed')
```

Al ejecutar el código, podemos ver la respuesta en nuestro navegador.

Figura 6. Lo que vemos en nuestro navegador web al conectarse a nuestro servidor web Pico W

3.9.3. Un servidor web asíncrono

Podemos usar el módulo uasyncio para implementar el mismo servidor, pero en este caso manejará las solicitudes HTTP de forma asíncrona en lugar de bloquearlas.

```
 1 import network
 2 import socket
 3 import time
 4
 5 from machine import Pin
 6 import uasyncio as asyncio
 7
 8 led = Pin(15, Pin.OUT)
 9 onboard = Pin("LED", Pin.OUT, value=0)
10
11 ssid = 'A Network'
12 password = 'A Password'
13
14 html = """<!DOCTYPE html>
15 <html>
16 <head> <title>Pico W</title> </head>
17 <body> <h1>Pico W</h1>
18 <p>%s</p>
19 </body>
20 </html>
21 """
22
```

```
wlan = network.WLAN(network.STA_IF)

def connect_to_network():
wlan.active(True)
wlan.config(pm = 0xa11140) # Apague el modo de bajo consumo
wlan.connect(ssid, password)

max_wait = 10
while max_wait > 0:
if wlan.status() < 0 or wlan.status() >= 3:
break
max_wait -= 1
print('waiting for connection...')
time.sleep(1)

if wlan.status() != 3:
raise RuntimeError('network connection failed')
else:
print('connected')
status = wlan.ifconfig()
print('ip = ' + status[0])

async def serve_client(reader, writer):
print("Client connected")
request_line = await reader.readline()
print("Request:", request_line)
# Los headers de solicitudes HTTP no nos interesan, salten los
while await reader.readline() != b"\r\n":
pass

request = str(request_line)
led_on = request.find('/light/on')
led_off = request.find('/light/off')
print( 'led on = ' + str(led_on))
print( 'led off = ' + str(led_off))

stateis = ""
if led_on == 6:
print("led on")
led.value(1)
stateis = "LED is ON"

if led_off == 6:
print("led off")
led.value(0)
stateis = "LED is OFF"

response = html % stateis
writer.write('HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n')
writer.write(response)

await writer.drain()
await writer.wait_closed()
print("Client disconnected")

async def main():
print('Connecting to Network...')
connect_to_network()

print('Setting up webserver...')
asyncio.create_task(asyncio.start_server(serve_client, "0.0.0.0", 80))
while True:
onboard.on()
print("heartbeat")
```

```
87 await asyncio.sleep(0.25)
88 onboard.off()
89 await asyncio.sleep(5)
```



```
90
91 try:
92 asyncio.run(main())
93 finally:
94 asyncio.new_event_loop()
```

3.10. ejecutando iperf

puede instalar iperf utilizando la herramienta upip:

```
>>> import network
>>> wlan = network.WLAN(network.STA_IF)
>>> wlan.active(True)
>>> wlan.connect('Wireless Network', 'The Password')
>>> import upip
>>> upip.install("uiperf3")
```

y comenzar un iperf3 cliente.

NOTA

El iperf El servidor debe estar ejecutándose en otra máquina.

```
>>> import uiperf3
>>> uiperf3.client('10.3.15.xx)

CLIENT MODE: TCP sending
Connecting to ('10.3.15.234', 5201)
Interval Transfer Bitrate
  0.00-1.00 sec 48.4 KBytes 397 Kbits/sec
  1.00-2.00 sec 48.4 KBytes 397 Kbits/sec
  2.00-3.00 sec 80.5 KBytes 659 Kbits/sec
  3.00-4.00 sec 100 KBytes 819 Kbits/sec
  4.00-5.00 sec 103 KBytes 845 Kbits/sec
  5.00-6.00 sec 22.7 KBytes 186 Kbits/sec
  6.00-7.00 sec 0.00 Bytes 0.00 bits/sec
  7.00-8.00 sec 0.00 Bytes 0.00 bits/sec
  8.00-9.00 sec 45.3 KBytes 371 Kbits/sec
  9.00-10.00 sec 89.1 KBytes 729 Kbits/sec
 10.00-10.01 sec 0.00 Bytes 0.00 bits/sec
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  0.00-10.01 sec 538 KBytes 440 Kbits/sec sender
>>>
```

3.11. ¿En qué hardware estoy ejecutando?

No existe un método directo para que el software escrito en MicroPython descubra si se está ejecutando en una Raspberry Pi Pico o en una Pico W observando el hardware. Sin embargo, puede saberlo indirectamente mirando si la funcionalidad de red está incluida en su firmware MicroPython particular:

```
1 import network
2 if hasattr(network, "WLAN"):
3 # la placa tiene funciones de WLAN
```

Alternativamente, puede inspeccionar la versión del firmware de MicroPython para verificar si fue compilada para Raspberry Pi Pico o para Pico W usando el módulo sys.

```
>>> import sys
>> sys.implementation
(name='micropython', version=(1, 19, 1), _machine='Raspberry Pi Pico W with RP2040', _mpy=4102)
```

Entonces si 'Pico W' in sys.implementation._machine se puede utilizar para detectar si su firmware fue compilado

para Pico W.

Capítulo 4. Acerca de Bluetooth

La interfaz Bluetooth integrada de Raspberry Pi Pico W es compatible con las funciones Bluetooth LE Central y Periféricos, junto con compatibilidad con Bluetooth Classic, y es configurable para que pueda habilitar tanto LE como Classic al mismo tiempo, o cualquiera de ellos individualmente.

NOTA

Detalles completos de protocolos y perfiles Bluetooth compatibles están disponibles en la cocina azul BTStack Repositorio Github. Además de los términos de la licencia BTstack estándar, es una licencia suplementaria que cubre el uso comercial de BTstack con Raspberry Pi Pico W o Raspberry Pi Pico WH.

4.1. Más sobre Bluetooth LE

Bluetooth LE divide el mundo en dispositivos periféricos y centrales. Los dispositivos periféricos son algo así como sensores: suelen ser pequeños, de baja potencia y con recursos limitados. Los dispositivos centrales son, por ejemplo, teléfonos móviles o ordenadores portátiles, aunque a menudo también funcionan en modo periférico.

NOTA

La especificación Bluetooth LE es un desastre en expansión de documentos entrelazados, eso abarca miles de páginas; el documento de estándares básicos tiene más de 2.700 páginas por sí solo. Proceda con precaución.

Los periféricos pueden funcionar en dos modos: ya sea mediante transmisión o cuando se conectan directamente a un dispositivo central. El mecanismo de transmisión es una de las grandes diferencias entre Bluetooth LE y Bluetooth "clásico". Al usarlo, el periférico puede enviar datos a cualquier dispositivo dentro del alcance.

Esto significa que un dispositivo periférico Bluetooth LE no necesariamente necesita estar emparejado (en Bluetooth LE hablamos de él como "conectado", en lugar de "emparejado" como lo hicimos con Bluetooth 2.1) a un dispositivo central para transferir datos. En modo emisión, el periférico enviará periódicamente paquetes publicitarios, disponibles para cualquiera que los esté buscando, para los dispositivos que actúan como "observadores".

El paquete de publicidad estándar describe el dispositivo de transmisión y sus capacidades, pero también es capaz de incluir información personalizada (datos de sensores, por ejemplo) que quizás desee transmitir.

Transmitir datos desde su periférico es una buena opción si está construyendo algo como una estación meteorológica, donde los datos no son confidenciales. Sin embargo, no existe ninguna seguridad durante la transmisión, por lo que para los datos personales, el dispositivo central debe conectarse al periférico.

Las conexiones son exclusivas. Esto significa que un periférico no se puede conectar a más de un dispositivo central a la vez. Cuando un dispositivo central se conecta a un periférico, el periférico dejará de anunciarse. Otros dispositivos no podrán verlo ni conectarse a él hasta que finalice la primera conexión. Mientras que un periférico solo se puede conectar a un dispositivo central, un dispositivo central se puede conectar a más de un periférico al mismo tiempo.

Si necesita intercambiar datos entre el periférico y el dispositivo central, deberá establecer una conexión entre los dos dispositivos.

4.1.1. Protocolos y perfiles

Además de los protocolos que componen el estándar Bluetooth LE, la especificación ofrece lo que se denomina "perfiles". Estos son los modos de funcionamiento básicos que necesitan todos los dispositivos Bluetooth LE, por ejemplo, el perfil de acceso genérico (GAP) y el perfil de atributo genérico (GATT), o perfiles que cubren casos de uso específicos como el perfil de frecuencia cardíaca.

4.1.2. El GAP

El perfil de acceso genérico (GAP) es el perfil que define las funciones de los dispositivos, incluidas las funciones periféricas y centrales que mencionamos en la última sección, junto con la publicidad y el descubrimiento.

Hay dos formas de anunciar datos utilizando GAP: datos publicitarios y paquetes de respuesta de escaneo. Si bien ambos paquetes utilizan el mismo formato de carga útil y constan de hasta 31 bytes de datos, sólo el paquete de datos publicitarios es obligatorio. Se envía en un intervalo publicitario preestablecido; cuanto más largo sea el intervalo, menos energía se utilizará. Al recibirlos, los dispositivos de escucha pueden solicitar el paquete de respuesta de escaneo con datos adicionales, si existen. Por ejemplo, el uso de datos de publicidad personalizados en los paquetes de transmisión es la forma en que se implementan los estándares de baliza Bluetooth.

Una vez que se haya establecido una conexión con el periférico, utilizará los servicios y características de GATT para comunicarse con el dispositivo periférico, y la publicidad se detendrá hasta que se finalice la conexión.

4.1.3. El GATT

El perfil de atributos genéricos (GATT) define cómo Bluetooth LE transfiere datos de un lado a otro entre dispositivos periféricos y centrales. Define perfiles, que son colecciones de servicios. Cada servicio tiene características que contienen datos.

Los roles cambian al pasar del GAP al GATT. El GATT define dos roles: cliente y servidor.

Puede parecer contradictorio, pero los dispositivos periféricos se conocen como Servidores GATT, mientras que los dispositivos centrales (generalmente más potentes) son Clientes GATT. Piénsalo de esta manera. El servidor tiene datos y el cliente quiere datos. Todas las conexiones entre los dispositivos las inicia el cliente.

Después de conectar el cliente, podemos obtener una lista de servicios ofrecidos por el servidor. Antes de conectar, el dispositivo central tiene una lista de servicios potencialmente incompleta a partir de los datos publicitarios.

4.1.4. Servicios y características

Los servicios se utilizan para dividir los datos en fragmentos asociados lógicamente y constan de una colección de características. Las características son los contenedores que contienen los datos asociados a un servicio. Tanto los servicios como las características se identifican mediante un identificador único, conocido como UUID. Ver Sección 4.1.5.

Las características contienen al menos dos atributos: una declaración de característica que contiene metadatos sobre los datos y el valor de característica que contiene los datos en sí. La característica también puede contener descriptores adicionales para ampliar los metadatos. Juntos, la declaración, el valor y cualquier descriptor opcional forman un paquete que constituye una característica.

Las características se pueden definir como lectura o escritura. El cliente lee las características mediante una solicitud de lectura, siendo el valor devuelto de la solicitud el valor de la característica. Los valores característicos se pueden escribir mediante una solicitud de escritura. El servidor devuelve una confirmación después de escribir el valor. Hay una propiedad de escritura adicional llamada comando de escritura. Cuando se escribe un valor de característica con un comando de escritura, el servidor no envía ninguna respuesta al cliente. El comando de escritura a veces se denomina escritura sin respuesta.

Se notifican e indican dos propiedades adicionales. Ambas son comunicaciones iniciadas por el servidor. Un cliente se suscribe para recibir una notificación cuando cambie el valor de una característica. Cuando se produce un cambio, el servidor notifica al cliente enviando el nuevo valor. Una indicación es similar a una notificación, excepto que el cliente debe acusar recibo de la indicación.

Las características pueden tener múltiples propiedades. Por ejemplo, una característica podría permitir leer, escribir, escribir comandos y notificar.

4.1.5. UUID

Bluetooth utiliza identificadores únicos universales (UUID) para muchas cosas, incluidos servicios y características. Servicios Bluetooth que han sido aprobados por el Grupo de Interés Especial de Bluetooth se les asignan UUID de 16 bits. Todos los demás servicios y características deben utilizar UUID de 128 bits. Los UUID de 128 bits se pueden generar con herramientas como uuidgen, p.

```
$uuidgen
437121E5-A6F0-43F9-8F8F-4AB73D6CC3EB
```

Está bien reutilizar los UUID si existen servicios y características que satisfagan sus necesidades. Si está creando sus propios servicios, utilice UUID de 128 bits. Ver el sitio para desarrolladores de Bluetooth para más información.

Capítulo 5. Trabajar con Bluetooth y C SDK

Se ha agregado compatibilidad con Bluetooth para Raspberry Pi Pico W al SDK de C/C++.

Las instrucciones abreviadas para instalar el SDK y ejemplos se pueden encontrar en la Sección 2.1. Para obtener instrucciones completas sobre cómo comenzar con el SDK, consulte el libro Empezando con Raspberry Pi Pico.

NOTA

Si no ha utilizado anteriormente una placa basada en RP2040, puede comenzar leyendo Empezando con Raspberry Pi Pico, mientras que se pueden encontrar más detalles sobre el SDK, junto con la documentación de nivel API, en el libro Raspberry Pi Pico C/C++ SDK.

ADVERTENCIA

Si no ha inicializado el `tinyusb` submódulo en su `pico-sdk` Al finalizar la compra, el USB CDC serial y otras funciones USB y el código de ejemplo no funcionarán, ya que el SDK no contendrá ninguna funcionalidad USB. Del mismo modo, si no ha inicializado el `cyw43-driver` y `lwip` submódulos en su proceso de pago, las funciones relacionadas con la red y el bluetooth no estarán habilitadas.

5.1. Un ejemplo de servicio Bluetooth

A ejemplo de Bluetooth independiente, sin toda la infraestructura de construcción de ejemplo común, está disponible en la `pico-examples` Repositorio de GitHub.

NOTA

El código de ejemplo independiente se encuentra en el `pico-examples` Repositorio de GitHub.

Detalles completos de protocolos y perfiles Bluetooth compatibles son cocina azul BTStack Repositorio Github.

5.1.1. Creación de un periférico de servicio de temperatura

El ejemplo independiente implementa un servicio de temperatura.

```
1 PRIMARY_SERVICE, GAP_SERVICE
2 CHARACTERISTIC, GAP_DEVICE_NAME, READ, "picow_temp"
3
4 PRIMARY_SERVICE, GATT_SERVICE
5 CHARACTERISTIC, GATT_DATABASE_HASH, READ,
6
7 PRIMARY_SERVICE, ORG_BLUETOOTH_SERVICE_ENVIRONMENTAL_SENSING
8 CHARACTERISTIC, ORG_BLUETOOTH_CHARACTERISTIC_TEMPERATURE, READ | NOTIFY | INDICATE |
```

`DYNAMIC`, To build this example you should:

```
$ git clone https://github.com/raspberrypi/pico-sdk.git --branch master
$ cd pico-sdk
$ git submodule update --init
$ cd ..
$ git clone https://github.com/raspberrypi/pico-examples.git --branch master
$ cd pico-examples
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake -DPICO_BOARD=pico_w ..
Using PICO_SDK_PATH from environment ('../../pico-sdk')
PICO_SDK_PATH is /home/pi/pico/pico-sdk
  .
  .
  .
-- Build files have been written to: /home/pi/pico/pico-examples/build
$ cd pico_w/bt/standalone
$ make
```

Junto con otros objetivos, ahora hemos creado dos binarios llamados picow_ble_temp_sensor.uf2 y picow_ble_temp_reader.uf2, que se puede arrastrar al dispositivo de almacenamiento masivo USB RP2040.

El método más rápido para cargar software en una placa basada en RP2040 por primera vez es montarla como un dispositivo de almacenamiento masivo USB. Hacer esto le permite arrastrar un archivo a la placa para programar la memoria flash. Continúe y conecte la Raspberry Pi Pico W a su Raspberry Pi usando un cable micro-USB, asegurándose de mantener presionado el botón BOOTSEL mientras lo hace, para forzarlo al modo de almacenamiento masivo USB.

Si está ejecutando Raspberry Pi Desktop, Raspberry Pi Pico W debería montarse automáticamente como un dispositivo de almacenamiento masivo USB. Desde aquí, se puede arrastrar y soltar el archivo UF2 en el dispositivo de almacenamiento masivo. RP2040 se reiniciará, se desmontará como dispositivo de almacenamiento masivo y comenzará a ejecutar el código actualizado.

Si se conecta al dispositivo Bluetooth mediante una aplicación de escáner en su teléfono (consulte Figura 7), debería ver una entrada de servicio en la sección "Detección ambiental". Debajo de esto, encontrará una sección de temperatura. Debería poder leer la temperatura o suscribirse para recibir notificaciones cuando cambie el valor de la temperatura.

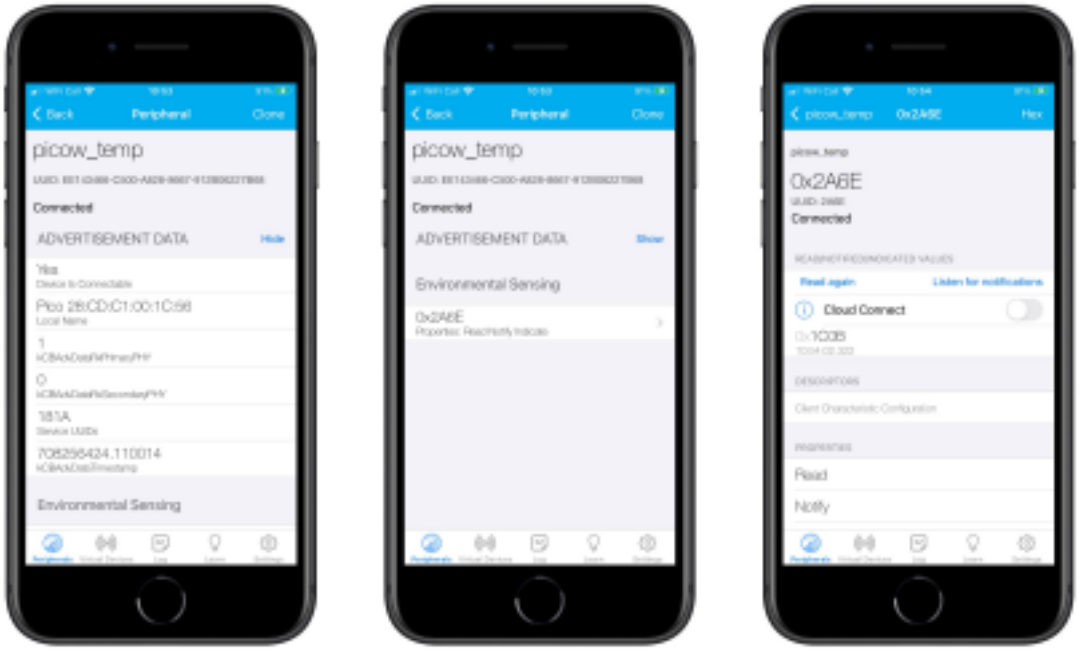


Figura 7. El picow_temp periférico en el Punch Through Azul claro aplicación; datos publicitarios (izquierda), servicios (medio), y servicio de temperatura (bien).

La temperatura se mostrará como un número hexadecimal, p. 0x4B0B. Este número es un little-endian de dos bytes

representación de la temperatura multiplicada por 100. Para recuperar el valor en grados centígrados, deberá invertir esta representación a big-endian (por ejemplo, 0x4B0B se convierte en 0x0B4B), convertir el valor a decimal (por ejemplo, 0x0B4B se convierte en 2891) y luego dividir por 100 para obtener el valor en centígrados (aquí sería 28,91°C).

5.2. Disponibilidad de otro código de ejemplo

Más código de ejemplo está disponible en pico-examples Repositorio de GitHub. Estos ejemplos son para Pico W y solo se construyen cuando PICO_BOARD=pico_w se pasa a CMake.

NOTA

Los ejemplos en el pico-examples El repositorio se toma de la Cocina Azul. pila de ejemplos bluetooth.

De forma predeterminada, los ejemplos de Bluetooth solo están integrados en un "modo" (background, poll o freertos), siendo el valor predeterminado fondo. Esto se puede cambiar pasando un modo a CMake al compilar en la línea de comando, por ejemplo. -DBTSTACK_EXAMPLE_TYPE=encuesta.

NOTA

Las versiones de FreeRTOS solo se pueden crear si FREERTOS_KERNEL_PATH se define.

Capítulo 6. Trabajar con Bluetooth en MicroPython

IMPORTANTE

Asegúrese de tener instalada la última versión de MicroPython, con la compatibilidad con Bluetooth habilitada. Hasta que se haya actualizado la funcionalidad Bluetooth, un binario prediseñado estará disponible en la Sección MicroPython del sitio de documentación.

Se ha agregado compatibilidad con Bluetooth para Raspberry Pi Pico W a MicroPython. Un binario prediseñado, que se puede descargar desde la sección MicroPython de este documento web, debería servir para la mayoría de los casos de uso y viene con la librería de micropython pre-integrada en el binario.

Binario prediseñado

Un binario prediseñado del último firmware de MicroPython está disponible en sección MicroPython del Sitio de documentación de Raspberry Pi. Ver Sección 3.2 para obtener instrucciones sobre la instalación.

NOTA

Si no ha utilizado anteriormente una placa basada en RP2040, comience leyendo el libro SDK de Raspberry Pi Pico Python.

NOTA

Puede encontrar más información sobre el uso de Bluetooth desde MicroPython en línea en Documentación de MicroPython.

6.1. Publicidad de un servicio Bluetooth

Podemos crear un servicio personalizado en MicroPython y anunciarlo usando el siguiente código:

Ejemplos de Pico MicroPython: https://github.com/raspberrypi/pico-micropython-examples/blob/master/bluetooth/ble_advertising.py

```
# Herramientas para generar carga útiles de publicidad

from micropython import const
import struct
import bluetooth

# Carga útiles de publicidad son repetidas por paquetes de la siguiente forma:
# 1 byte de longitud de datos (N + 1)
# 1 tipo de byte (mire los constantes abajo)
# N bytes de datos de tipos específicos

_ADV_TYPE_FLAGS = const(0x01)
_ADV_TYPE_NAME = const(0x09)
_ADV_TYPE_UUID16_COMPLETE = const(0x3)
_ADV_TYPE_UUID32_COMPLETE = const(0x5)
_ADV_TYPE_UUID128_COMPLETE = const(0x7)
_ADV_TYPE_UUID16_MORE = const(0x2)
_ADV_TYPE_UUID32_MORE = const(0x4)
_ADV_TYPE_UUID128_MORE = const(0x6)
_ADV_TYPE_APPEARANCE = const(0x19)
```

```
21
22
23 # Genere una carga útil (payload) para pasarla a gap_advertise(adv_data=...).
24 def advertising_payload(limited_disc=False, br_edr=False, name=None, services=None,
  appearance=0):
25 payload = bytearray()
26
27 def _append(adv_type, value):
28 nonlocal payload
       29 payload += struct.pack("BB", len(value) + 1, adv_type) + value
30
31 _append(
32 _ADV_TYPE_FLAGS,
33 struct.pack("B", (0x01 if limited_disc else 0x02) + (0x18 if br_edr else 0x04)), 34 )
35
36 if name:
37 _append(_ADV_TYPE_NAME, name)
38
39 if services:
40 for uuid in services:
41 b = bytes(uuid)
42 if len(b) == 2:
43 _append(_ADV_TYPE_UUID16_COMPLETE, b)
44 elif len(b) == 4:
45 _append(_ADV_TYPE_UUID32_COMPLETE, b)
46 elif len(b) == 16:
47 _append(_ADV_TYPE_UUID128_COMPLETE, b)
48
49 # Mire org.bluetooth.characteristic.gap.appearance.xml
50 if appearance:
       51 _append(_ADV_TYPE_APPEARANCE, struct.pack("<h", appearance))
52
53 return payload
54
55
56 def decode_field(payload, adv_type):
57 i = 0
58 result = []
59 while i + 1 < len(payload):
60 if payload[i + 1] == adv_type:
61 result.append(payload[i + 2 : i + payload[i] + 1])
62 i += 1 + payload[i]
63 return result
64
65
66 def decode_name(payload):
67 n = decode_field(payload, _ADV_TYPE_NAME)
68 return str(n[0], "utf-8") if n else ""
69
70
71 def decode_services(payload):
72 services = []
73 for u in decode_field(payload, _ADV_TYPE_UUID16_COMPLETE):
    74 services.append(bluetooth.UUID(struct.unpack("<h", u)[0]))
75 for u in decode_field(payload, _ADV_TYPE_UUID32_COMPLETE):
    76 services.append(bluetooth.UUID(struct.unpack("<d", u)[0]))
77 for u in decode_field(payload, _ADV_TYPE_UUID128_COMPLETE):
78 services.append(bluetooth.UUID(u))
79 return services
80
81
82 def demo():
83 payload = advertising_payload(
```

```
84 name="micropython",
       85 services=[bluetooth.UUID(0x181A), bluetooth.UUID("6E400001-B5A3-F393-E0A9-
  E50E24DCCA9E")],
86 )
87 print(payload)
88 print(decode_name(payload))
89 print(decode_services(payload))
90
91
92 if __name__ == "__main__":
93 demo()
```

Detalles completos de protocolos y perfiles Bluetooth compatibles son cocina azul BTStack Repositorio de GitHub.

6.2. Un ejemplo de servicio Bluetooth

Podemos utilizar el código publicitario periférico genérico en Sección 6.1 para ayudar a implementar un servicio de ejemplo.

6.2.1. Creación de un periférico de servicio de temperatura

Este ejemplo demuestra un periférico de sensor de temperatura simple. El valor local del sensor se actualiza cada diez segundos y cualquier dispositivo central conectado será notificado del cambio.

Ejemplos de Pico MicroPython: https://github.com/raspberrypi/pico-micropython-examples/blob/master/bluetooth/picow_ble_temp_sensor.py

```
 1 # Este ejemplo muestra una temperatura simple periférica.
 2 #
 3 # El valor local del sensor se actualiza, notificará
 4 # cualquier central conectada cada 10 segundos.
 5
 6 import bluetooth
 7 import random
 8 import struct
 9 import time
10 import machine
11 import ubinascii
12 from ble_advertising import advertising_payload
13 from micropython import const
14 from machine import Pin
15
16 _IRQ_CENTRAL_CONNECT = const(1)
17 _IRQ_CENTRAL_DISCONNECT = const(2)
18 _IRQ_GATTS_INDICATE_DONE = const(20)
19
20 _FLAG_READ = const(0x0002)
21 _FLAG_NOTIFY = const(0x0010)
22 _FLAG_INDICATE = const(0x0020)
23
24 # org.bluetooth.service.environmental_sensing
25 _ENV_SENSE_UUID = bluetooth.UUID(0x181A)
26 # org.bluetooth.characteristic.temperature
27 _TEMP_CHAR = (
28 bluetooth.UUID(0x2A6E),
29 _FLAG_READ | _FLAG_NOTIFY | _FLAG_INDICATE,
30 )
31 _ENV_SENSE_SERVICE = (
32 _ENV_SENSE_UUID,
```

```
 33 (_TEMP_CHAR,),
 34 )
 35
 36 # org.bluetooth.characteristic.gap.appearance.xml
 37 _ADV_APPEARANCE_GENERIC_THERMOMETER = const(768)
 38
 39 class BLETemperature:
 40 def __init__(self, ble, name=""):
 41 self._sensor_temp = machine.ADC(4)
 42 self._ble = ble
 43 self._ble.active(True)
 44 self._ble.irq(self._irq)
 45 ((self._handle,),) = self._ble.gatts_register_services((_ENV_SENSE_SERVICE,))  46
self._connections = set()
 47 if len(name) == 0:
 48 name = 'Pico %s' % ubinascii.hexlify(self._ble.config('mac')[1],':').decode
().upper()
 49 print('Sensor name %s' % name)
 50 self._payload = advertising_payload(
 51 name=name, services=[_ENV_SENSE_UUID]
 52 )
 53 self._advertise()
 54
 55 def _irq(self, event, data):
 56 # Revise las conexiones para que podamos notificar.
 57 if event == _IRQ_CENTRAL_CONNECT:
 58 conn_handle, _, _ = data
 59 self._connections.add(conn_handle)
 60 elif event == _IRQ_CENTRAL_DISCONNECT:
 61 conn_handle, _, _ = data
 62 self._connections.remove(conn_handle)
 63 # Comience la publicidad de nuevo para permitir una nueva conexión.
 64 self._advertise()
 65 elif event == _IRQ_GATTS_INDICATE_DONE:
 66 conn_handle, value_handle, status = data
 67
 68 def update_temperature(self, notify=False, indicate=False):
 69 # Escriba un valor local, preto para que lo lea una central.
 70 temp_deg_c = self._get_temp()
 71 print("write temp %.2f degc" % temp_deg_c);
 72 self._ble.gatts_write(self._handle, struct.pack("<h", int(temp_deg_c * 100)))  73 if
notify or indicate:
 74 for conn_handle in self._connections:
 75 if notify:
 76 # Notifique centrales conectadas.
                    77 self._ble.gatts_notify(conn_handle, self._handle)
 78 if indicate:
 79 # Indique centrales conectadas
                    80 self._ble.gatts_indicate(conn_handle, self._handle)
 81
 82 def _advertise(self, interval_us=500000):
        83 self._ble.gap_advertise(interval_us, adv_data=self._payload)
 84
 85 # ref https://github.com/raspberrypi/pico-micropython-
  examples/blob/master/adc/temperature.py
 86 def _get_temp(self):
 87 conversion_factor = 3.3 / (65535)
        88 reading = self._sensor_temp.read_u16() * conversion_factor
 89
 90 # El sensor de temperatura mide el voltaje (Vbe) de un diodo bipolar, conectado al canal ADC
quinto.
 91 # Normalmente, Vbe = 0.706V a 27ºC, con una inclinación de -1.721mV (0.001721) por  grado.
 92 return 27 - (reading - 0.706) / 0.001721
```

```
93
94 def demo():
95 ble = bluetooth.BLE()
96 temp = BLETemperature(ble)
97 counter = 0
98 led = Pin('LED', Pin.OUT)
99 while True:
100 if counter % 10 == 0:
      101 temp.update_temperature(notify=True, indicate=False)
102 led.toggle()
103 time.sleep_ms(1000)
104 counter += 1
105
106 if __name__ == "__main__":
107 demo()
```

NOTA

El sensor de temperatura RP2040 mide la V_{ser} voltaje de un diodo bipolar polarizado, conectado al quinto canal ADC. Normalmente, V_{ser} = 0,706V a 27°C, con una pendiente de -1,721mV (0,001721) por grado.

NOTA

Este servicio de ejemplo importa código de Sección 6.1.

Si se conecta al dispositivo Bluetooth mediante una aplicación de escáner en su teléfono (consulte Figura 8) debería ver una entrada de servicio en la sección "Detección ambiental". Debajo de esto, hay una sección de temperatura. Debería poder leer la temperatura o suscribirse para recibir notificaciones cuando cambie el valor de la temperatura.

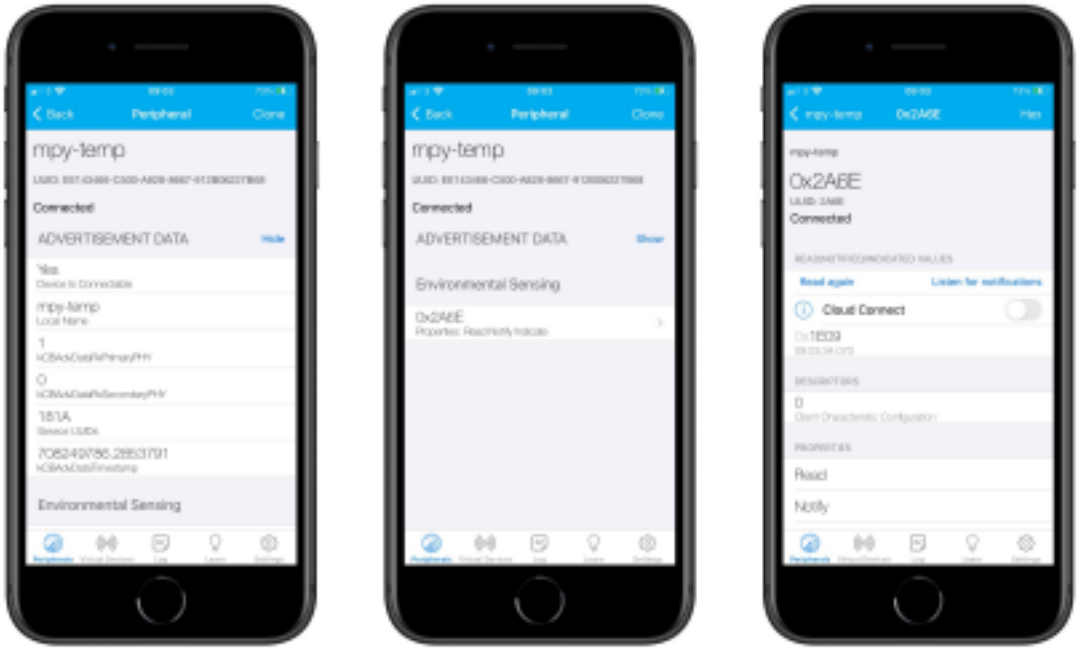


Figura 8. El mpy-temp periférico en el Perforar Azul claro aplicación: datos publicitarios (izquierda), servicios (medio), y servicio de temperatura (bien).

La temperatura se mostrará como un número hexadecimal, p. 0x4B0B. Este número es una representación little-endian de dos bytes de la temperatura multiplicada por 100. Para recuperar el valor en centígrados, necesitará convertir esta representación a big-endian (por ejemplo, 0x4B0B se convierte en 0x0B4B), convertir el valor a decimal (por ejemplo, 0x0B4B se convierte en 2891), y luego divida por 100 para obtener el valor en grados centígrados (aquí sería 28,91°C).

6.2.2. Implementación de un dispositivo central

Si bien es probable que el principal uso de BLE en Pico W sea como un periférico que ofrece servicios, también es posible utilizar su Pico W como dispositivo central. Aquí implementamos un dispositivo central que encuentra y se conecta a periféricos de temperatura (ver Sección 6.2.1).

Ejemplos de Pico MicroPython: https://github.com/raspberrypi/pico-micropython-examples/blob/master/bluetooth/picow_ble_temp_reader.py

```
# Este ejemplo encuentra y conecta a un sensor de temperatura BLE (e.g. ble_temperature.py)

import bluetooth
import random
import struct
import time
import micropython
from ble_advertising import decode_services, decode_name
from micropython import const
from machine import Pin

_IRQ_CENTRAL_CONNECT = const(1)
_IRQ_CENTRAL_DISCONNECT = const(2)
_IRQ_GATTS_WRITE = const(3)
_IRQ_GATTS_READ_REQUEST = const(4)
_IRQ_SCAN_RESULT = const(5)
_IRQ_SCAN_DONE = const(6)
_IRQ_PERIPHERAL_CONNECT = const(7)
_IRQ_PERIPHERAL_DISCONNECT = const(8)
_IRQ_GATTC_SERVICE_RESULT = const(9)
_IRQ_GATTC_SERVICE_DONE = const(10)
_IRQ_GATTC_CHARACTERISTIC_RESULT = const(11)
_IRQ_GATTC_CHARACTERISTIC_DONE = const(12)
_IRQ_GATTC_DESCRIPTOR_RESULT = const(13)
_IRQ_GATTC_DESCRIPTOR_DONE = const(14)
_IRQ_GATTC_READ_RESULT = const(15)
_IRQ_GATTC_READ_DONE = const(16)
_IRQ_GATTC_WRITE_DONE = const(17)
_IRQ_GATTC_NOTIFY = const(18)
_IRQ_GATTC_INDICATE = const(19)

_ADV_IND = const(0x00)
_ADV_DIRECT_IND = const(0x01)
_ADV_SCAN_IND = const(0x02)
_ADV_NONCONN_IND = const(0x03)

# org.bluetooth.service.environmental_sensing
_ENV_SENSE_UUID = bluetooth.UUID(0x181A)
# org.bluetooth.characteristic.temperature
_TEMP_UUID = bluetooth.UUID(0x2A6E)
_TEMP_CHAR = (
_TEMP_UUID,
bluetooth.FLAG_READ | bluetooth.FLAG_NOTIFY,
)
_ENV_SENSE_SERVICE = (
_ENV_SENSE_UUID,
(_TEMP_CHAR,),
)

class BLETemperatureCentral:
def __init__(self, ble):
self._ble = ble
self._ble.active(True)
```

```
54 self._ble.irq(self._irq)
55 self._reset()
56 self._led = Pin('LED', Pin.OUT)
57
58 def _reset(self):
59 # Nombre y dirección en caché de un análisis exitoso.
60 self._name = None
61 self._addr_type = None
62 self._addr = None
63
64 # Valor en caché (si hemos captado alguno)
65 self._value = None
66
67 # Devoluciones de llamada para la finalización de diversas operaciones.
68 # Estos se restablecen a None después de ser invocados.
69 self._scan_callback = None
70 self._conn_callback = None
71 self._read_callback = None
72
73 # Devolución persistente cuando un nuevo conjunto de datos es notificado por el dispositivo
74 self._notify_callback = None
75
76 # Dispositivo conectado.
77 self._conn_handle = None
78 self._start_handle = None
79 self._end_handle = None
80 self._value_handle = None
81
82 def _irq(self, event, data):
83 if event == _IRQ_SCAN_RESULT:
84 addr_type, addr, adv_type, rssi, adv_data = data
85 if adv_type in (_ADV_IND, _ADV_DIRECT_IND):
86 type_list = decode_services(adv_data)
87 if _ENV_SENSE_UUID in type_list:
88 # Dispositivo posible encontrado, recuerde lo y pare de escanear.
89 self._addr_type = addr_type
90 self._addr = bytes(addr) # Nota: addr buffer es propiedad del que ejecuta, así que no hay que copiarlo.
91 self._name = decode_name(adv_data) or "?"
92 self._ble.gap_scan(None)
93
94 elif event == _IRQ_SCAN_DONE:
95 if self._scan_callback:
96 if self._addr:
97 # Dispositivo encontrado durante el análisis (y el análisis se detuvo).
98 self._scan_callback(self._addr_type, self._addr, self._name)
99 self._scan_callback = None
100 else:
101 # Análisis se detuvo.
102 self._scan_callback(None, None, None)
103
104 elif event == _IRQ_PERIPHERAL_CONNECT:
105 # Conexión exitosa
106 conn_handle, addr_type, addr = data
107 if addr_type == self._addr_type and addr == self._addr:
108 self._conn_handle = conn_handle
109 self._ble.gattc_discover_services(self._conn_handle)
110
111 elif event == _IRQ_PERIPHERAL_DISCONNECT:
112 # Desconexión (ya sea iniciada por nosotros o por el extremo remoto).
113 conn_handle, _, _ = data
114 if conn_handle == self._conn_handle:
115 # Si lo hemos iniciado nosotros, ya se restablecerá.
116 self._reset()
117
```

```
118 elif event == _IRQ_GATTC_SERVICE_RESULT:
119 # Conexión ha vuelto
120 conn_handle, start_handle, end_handle, uuid = data
            121 if conn_handle == self._conn_handle and uuid == _ENV_SENSE_UUID:
                122 self._start_handle, self._end_handle = start_handle, end_handle
123
124 elif event == _IRQ_GATTC_SERVICE_DONE:
125 # Servicio completo.
126 if self._start_handle and self._end_handle:
127 self._ble.gattc_discover_characteristics(
                    128 self._conn_handle, self._start_handle, self._end_handle
129 )
130 else:
                131 print("Failed to find environmental sensing service.")
132
133 elif event == _IRQ_GATTC_CHARACTERISTIC_RESULT:
134 # El dispositivo conectado ha devuelto alguna característica.
            135 conn_handle, def_handle, value_handle, properties, uuid = data
            136 if conn_handle == self._conn_handle and uuid == _TEMP_UUID:
137 self._value_handle = value_handle
138
139 elif event == _IRQ_GATTC_CHARACTERISTIC_DONE:
140 # Consulta de la característica ha sido completada.
141 if self._value_handle:
142 # Hemos descubierto y conectado otro dispositivo, eche la devolución. 143 if
self._conn_callback:
144 self._conn_callback()
145 else:
                146 print("Failed to find temperature characteristic.")
147
148 elif event == _IRQ_GATTC_READ_RESULT:
149 # Una lectura se ha completado con éxito.
150 conn_handle, value_handle, char_data = data
151 if conn_handle == self._conn_handle and value_handle == self._value_handle: 152
self._update_value(char_data)
153 if self._read_callback:
154 self._read_callback(self._value)
155 self._read_callback = None
156
157 elif event == _IRQ_GATTC_READ_DONE:
158 # Lectura completa.
159 conn_handle, value_handle, status = data
160
161 elif event == _IRQ_GATTC_NOTIFY:
                162 # El ble_temperature.py notifica periódicamente su valor.
163 conn_handle, value_handle, notify_data = data
164 if conn_handle == self._conn_handle and value_handle == self._value_handle: 165
self._update_value(notify_data)
166 if self._notify_callback:
167 self._notify_callback(self._value)
168
   169 # Devuelve true si hemos conectado con éxito y descubierto las características.
170 def is_connected(self):
        171 return self._conn_handle is not None and self._value_handle is not None
172
173 # Encuentre un dispositivo que anuncie el sensor ambiental.
174 def scan(self, callback=None):
175 self._addr_type = None
176 self._addr = None
177 self._scan_callback = callback
178 self._ble.gap_scan(2000, 30000, 30000)
179
180 # Conéctese al dispositivo en cuestión (sino use una dirección en caché de un análisis).
   181 def connect(self, addr_type=None, addr=None, callback=None):
```

```
182 self._addr_type = addr_type or self._addr_type
183 self._addr = addr or self._addr
184 self._conn_callback = callback
185 if self._addr_type is None or self._addr is None:
186 return False
187 self._ble.gap_connect(self._addr_type, self._addr)
188 return True
189
190 # Desconéctese del dispositivo actual.
191 def disconnect(self):
192 if not self._conn_handle:
193 return
194 self._ble.gap_disconnect(self._conn_handle)
195 self._reset()
196
197 # Emite una lectura (asincrónica), invocará una devolución de los datos.
198 def read(self, callback):
199 if not self.is_connected():
200 return
201 self._read_callback = callback
202 try:
            203 self._ble.gattc_read(self._conn_handle, self._value_handle)
204 except OSError as error:
205 print(error)
206
207 # Se invocará una devolución de los datos en cuanto nos notifique el dispositivo.
208 def on_notify(self, callback):
209 self._notify_callback = callback
210
211 def _update_value(self, data):
212 # Los datos están expresados en sint16 en grados Celsius con una resolución de 0,01
grados Celsius. 213 try:
214 self._value = struct.unpack("<h", data)[0] / 100
215 except OSError as error:
216 print(error)
217
218 def value(self):
219 return self._value
220
221 def sleep_ms_flash_led(self, flash_count, delay_ms):
222 self._led.off()
223 while(delay_ms > 0):
224 for i in range(flash_count):
225 self._led.on()
226 time.sleep_ms(100)
227 self._led.off()
228 time.sleep_ms(100)
229 delay_ms -= 200
230 time.sleep_ms(1000)
231 delay_ms -= 1000
232
233 def print_temp(result):
234 print("read temp: %.2f degc" % result)
235
236 def demo(ble, central):
237 not_found = False
238
239 def on_scan(addr_type, addr, name):
240 if addr_type is not None:
241 print("Found sensor: %s" % name)
242 central.connect()
243 else:
244 nonlocal not_found
245 not_found = True
```

```
246 print("No sensor found.")
247
248 central.scan(callback=on_scan)
249
250 # Espere una conexión...
251 while not central.is_connected():
252 time.sleep_ms(100)
253 if not_found:
254 return
255
256 print("Connected")
257
258 # Emitir lecturas explícitamente
259 while central.is_connected():
260 central.read(callback=print_temp)
261 sleep_ms_flash_led(central, 2, 2000)
262
263 print("Disconnected")
264
265 if __name__ == "__main__":
266 ble = bluetooth.BLE()
267 central = BLETemperatureCentral(ble)
268 while(True):
269 demo(ble, central)
270 sleep_ms_flash_led(central, 1, 10000)
```

NOTA

Este ejemplo importa código de Sección 6.1.

Copiar ble_publicidad.py y picow_ble_temp_reader.py a una segunda Raspberry Pi Pico W. Debería comenzar a funcionar y el LED integrado parpadea brevemente una vez repetidamente si no puede encontrar un dispositivo al que conectarse. Una vez que encuentre otro dispositivo ejecutando el picow_ble_temp_sensor.py, parpadeó repetidamente dos veces, rápidamente, cuando esté conectado y leyendo la temperatura a través de Bluetooth.

Apéndice A: Construyendo MicroPython desde el código fuente

Antes de poder continuar con la construcción de un MicroPython UF2 para Raspberry Pi Pico W desde el código fuente, debe instalar las dependencias normales para construir MicroPython. Véase la Sección 1.3 del libro SDK de Raspberry Pi Pico Python para más detalles.

Después debería clonar el micropython y micropython-lib repositorios.

```
$ mkdir pico_w
$ cd pico_w
$ git clone https://github.com/micropython/micropython.git --branch master
$ git clone https://github.com/micropython/micropython-lib.git --branch master
```

NOTA

Poniendo micropython-lib lado a lado con su pago de MicroPython significa que se incorpora automáticamente a su compilación de MicroPython y a las librerías en micropython-lib se "pre-agregara" a la lista de módulos disponibles de forma predeterminada en su dispositivo Pico W.

Luego construye MicroPython:

```
$ cd micropython
$ make -C ports/rp2 BOARD=PICO_W submodules
$ make -C mpy-cross
$ cd ports/rp2
$ make BOARD=PICO_W
```

Si todo ha ido bien, habrá un nuevo directorio llamado build-PICO_W (eso es ports/rp2/build-PICO_W en relación con el nivel superior micropython directorio), que contiene los nuevos archivos binarios del firmware. Arrastra y suelta el firmware.uf2 en la unidad RPI-RP2 que aparece una vez que su Raspberry Pi Pico W está en modo BOOTSEL.

Apéndice B: Historial de publicación de documentación

Liberar	Fecha	Descripción
1.0	30 de junio de 2022	• Versión inicial
Libros de datos Pico y Pico W combinados en un historial de lanzamientos unificado		
2.0	01 dic 2022	• Actualizaciones y correcciones menores. • Se agregó información de disponibilidad del RP2040. • Se agregaron condiciones de almacenamiento y características térmicas del RP2040. • Reemplace la documentación de la librería SDK con enlaces a la página en línea versión • Instrucciones de construcción y uso de Picoprobe actualizadas
2.1	03 de marzo de 2023	• Una gran cantidad de actualizaciones y correcciones menores. • Huella SMT de Pico W corregida • Actualizado para la versión 1.5.0 del SDK de Raspberry Pi Pico C • Erratas agregadas E15 • Se agregó documentación sobre el nuevo Pico instalador de Windows • Se agregó documentación sobre el Pico-W-Go extensión para Desarrollo de Python • Se agregó un ejemplo de red inalámbrica a Python. documentación • Se agregaron especificaciones de marcado de paquetes. • Se agregaron cifras de consumo de energía de referencia del RP2040 • Se agregó un diagrama de mantenimiento de antena a la hoja de datos de Pico W.
2.2	14 de junio de 2023	• Actualizaciones y correcciones menores. • Actualizado para la versión 1.5.1 del SDK de Raspberry Pi Pico C • Documentación sobre la compatibilidad con Bluetooth para Pico W

Tabla 1. Documentación historial de lanzamiento

Liberar	Fecha	Descripción
2.3	02 de febrero de 2024	• Numerosas actualizaciones y correcciones menores. • Actualizar la información del registro ROSC • Documentación de introducción actualizada para MS Windows y MacOS de Apple • Actualizaciones derivadas del lanzamiento de Raspberry Pi 5 • Se reintroduce la documentación actualizada de la librería SDK (antes retirado en 2.0 debido a conflictos XML) • Actualizado para incluir el nuevo número de pieza recomendado para cristales utilizados con RP2040 • Se agregó nueva información de plantilla de pegado para Pico y Pico W. • Otras actualizaciones de la documentación de respaldo
2.4	02 mayo 2024	• Numerosas actualizaciones y correcciones menores. • Correcciones de formato para la documentación de nivel de API del SDK de Pico C • Renombrado picosonda firmware para sonda de depuración • Se aclaró que la configuración de compilación de CMake utiliza variables de caché, no variables de configuración • Se corrigieron los nombres de parámetros incorrectos utilizados en el @asm_pio decorador y Máquina estatal inicializar ejemplos • MicroPython expandido cáscara ejemplos para incluir un completo guía sobre cómo cargar y ejecutar programas en su dispositivo • Se agregó un ejemplo que demuestra cómo restablecer un Pico desde la línea de comando usando OpenOCD • Documentación mejorada del complemento VS Code MicroPico para reflejar el nuevo nombre del complemento, la eliminación del servidor FTP integrado y algunas instrucciones de uso adicionales • Se agregó documentación sobre el código oficial Raspberry Pi Pico VS. extensión.

La última versión se puede encontrar en https://datasheets.raspberrypi.com/picow/connecting-to-the-internet-with-pico w.pdf.