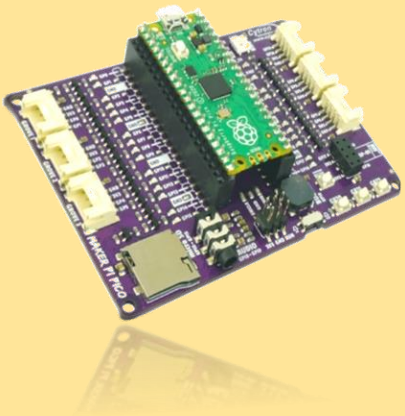
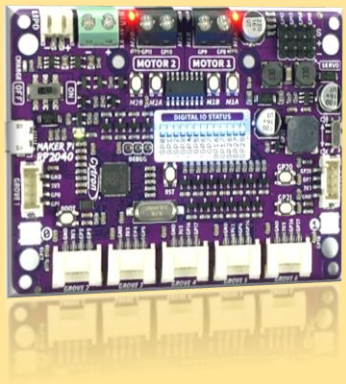
Raspberry Pi Pico

Programación con MicroPhyton (**Thonny**)

Utilización de las tarjetas **Cytron**



Maker **Pi PICO**

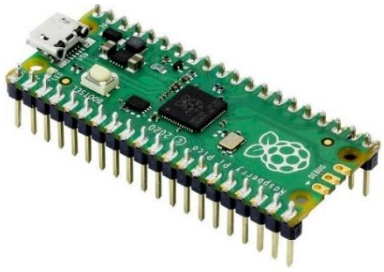


Maker **Pi RP2040**

Profesor: José Manuel Ruiz Gutiérrez

Marzo 2023

Indice



1. Introducción a este libro
2. Microcontrolador Raspberri Pi PICO
3. Tarjetas **Cytron**
4. Programando con **MicroPython Thonny**
   4.1. Proceso de creación de una aplicación
5. Estrctura basica de un programa. Uso de librerias
6. Botones y Leds
   6.1. Salida Digital: Blink (intermitente)
   6.2. Activación secuencial de Salidas
   6.3. Monoestable con monitorización de estado
   6.4. Medir tiempo entre dos eventos
   6.5. Contador binario
   6.6. Biestable
   6.7. Codificador Rotativo
   6.8. Generador de pulsos
   6.9. Semáforo
   6.10. Barra de LEDs MY9221 de Seeed Grove
7. Señales analogicas
   7.1. Leer un valor analógico desde un pin
   7.2. Blink con tiempo variable
   7.3. Salida analógica. PWM
   7.4. Efecto fadding de salida analógica
   7.5. Joysttick
8. Sonido
   8.1. Emitir un tono
   8.2. Alarma sirena
   8.3. Generación de una melodía
9. Jugando con Neopixel
   9.1. Trabajando con Neopixel
   9.2. Secuencias de iluminacion
   9.3. Arco de Iris
   9.4. Semaforos
10. Usando sensores
    10.1. Medida de Temperatura y humedad
    10.2. Termostato
    10.3. Fotoresistor: Medida de Luz
11. Detector de Infrarrojos
    11.1. Infrarrojos básico

    11.2. Infrarrojos Decodificador
12. Display OLED
    12.1. OLED Dibuja texto.
    12.2. OLED Señal Analógica.
    12.3. OLED Dibuja varias cosas.
    12.4. OLED Senoidal.
13. Usando un display LCD
    13.1. LCD Básico.
    13.2. LCD Lectura de valor analógico
    13.3. LCD Termostato.
14. Midiendo con ultrasonidos
    14.1. Medidor básico de ultrasonidos
    14.2. Medidor de ultrasonidos con emisión de tonos
15. Controlando Servos.
    15.1. Control de servo básico
    15.2. Control de servo mediante librería “Servo”.
    15.3. Servo control con Maker Pi RP2040
    15.4. Control servo con potenciómetro.
16. Controlando Motores
    16.1. Control de dos motores de cc. con la tarjeta Maker Pi RP2040
    16.2. Contrl de un Motor de cc. sin usar librería.
    16.3. Control de dos motores
    16.4. Control de velocidad de un motor con un potenciómetro
    16.5. Control de velocidsd y sentido de giro de un motor
17. Otros dispositivos
    17.1. Control de Display Numérico de 4 cifras.
    17.2. Control de matriz de LEDs Max7219
    17.3. Escaneo de puertos I2C

LINK INTERESANTES

1. Introducción a este libro

La aparición de la tarjeta Raspberry Pi Pico en todas sus versiones marca un punto de inflexión en la oferta de *Plataformas Hardware Educativas* orientadas al estudio y desarrollo de los microcontroladores en el ámbito educativo y en el mundo Maker.

La firma **Cytron** ha presentado al mercado una serie de diseños basados en esta nueva versión de Raspberry Pi que ha despertado el interés de la comunidad de usuarios de estas plataformas. Los desarrollos mas importantes disponibles en esta oferta son básicamente dos: Tarjeta MAKER PI PICO y Tarjeta MAKER PI RP2040.

En este libro se van a usar ambas tarjetas para desarrollar los más de 60 ejemplos que se abordan como aplicaciones básicas basadas en los procesadores Raspberry Pi PICO.

Para la programación de las plataformas basadas en este nuevo procesador se pueden seleccionar diversas herramientas basadas en los estándares MicroPython y CircuitPython así como el IDE Arduino. Por otra parte, existe una herramienta de programación de libre difusión basada en el estándar Snap! Denominada MicroBlocks que ofrece unas potentes posibilidades para iniciarse en la programación grafica muy adecuadas para usuarios que no controlen aun el lenguaje de programación Python o C.

En este libro vamos a realizar las practicas con las tarjetas de **Cytron** usando el software de programación Thonny y el firmware propio para escribir el código en **MicroPython**

Recomiendo el uso de este libro especialmente a los estudiantes de *Educación Secundaria, Formación Profesional y Bachillerato* y todos cuantos quieran iniciarse en el uso de este poderoso microcontrolador.

Para finalizar quiero agradecer a **Cytron** el haberme brindado la posibilidad de manejar su documentación, usar sus imágenes y disponer de las tarjetas. Especialmente mi agradecimiento a **Cheryl Ng**. *Head of rero EDUteam*. **Cytron Technologies** **Malasia**

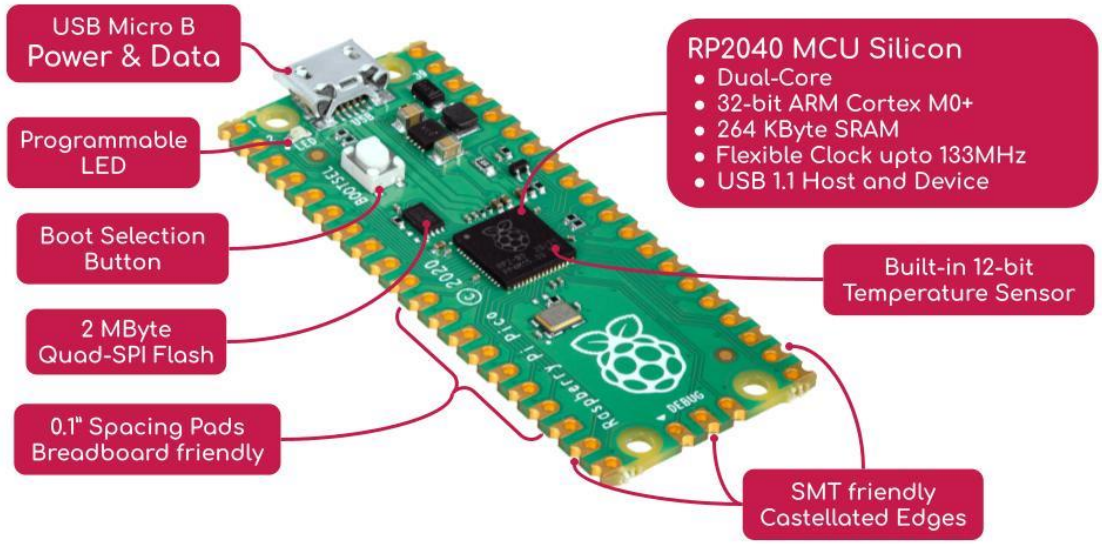
Profesor: José Manuel Ruiz Gutiérrez

Marzo 2023

2. Microcontrolador Raspberri Pi PICO

Descripción

La 1ª placa MCU de Raspberry Pi - Pico



Lanzada el 21 de enero de 2021, la Raspberry Pi Pico es la 1ª placa de desarrollo de microcontroladores de la Fundación Raspberry Pi. También se basa en el 1er Microcontrolador IC / Silicon - RP2040, diseñado y producido por ingenieros del equipo de Raspberry Pi también.

¿Qué es Raspberry Pi Pico?

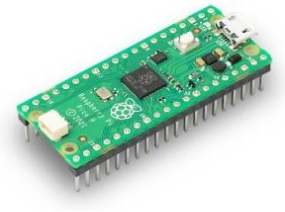
Una placa de desarrollo de microcontroladores

¿Cuáles son las diferencias entre la Raspberry Pi 4 Modelo B y esta Raspberry Pi Pico? Bueno, básicamente, la Raspberry Pi 4 Modelo B es una placa base o una computadora de placa única que puede usar para jugar, trabajar, grabar datos, navegar por Internet, ver películas, como un reproductor multimedia y muchos más. Raspberry Pi Pico no está diseñado para reemplazar a la Raspberry Pi 4 Modelo B (o placa similar), es más para proyectos de computación física donde controla cualquier cosa, desde pequeños componentes electrónicos, LED, motores; Leer información de sensores o comunicarse con otros microcontroladores.

Probablemente ya tenemos muchos microcontroladores en nuestra casa. Por ejemplo, una lavadora es controlada por un microcontrolador; Lo más probable es que nuestro reloj también lo sea; También hay uno en un microondas. Por supuesto, todos estos microcontroladores ya tienen sus programas para las aplicaciones.

Esta Raspberry Pi Pico es una placa de microcontrolador que puede realizar computación física y se puede reprogramar fácilmente a través de una conexión USB.

Se



Raspberry Pi Pico con cabezales **PRE-SOLDADOS**

Raspberry Pi Pico sin cabecera, SMD amigable

dispone de dos modelos de Raspberry Pi Pico

Programación con **MicroPython**

Como Python es el lenguaje de programación oficial de Raspberry Pi OS, **MicroPython** es elegido para ser uno de los lenguajes de programación de Raspberry Pi Pico. **MicroPython** es una implementación eficiente y eficiente del lenguaje de programación Python 3 que incluye un pequeño subconjunto de la biblioteca estándar de Python y está optimizada para ejecutarse en microcontroladores y en entornos restringidos.

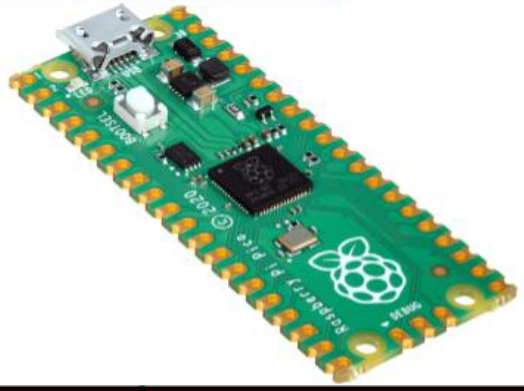
La programación de la carga de **MicroPython** en Raspberry Pi Pico es fácil. Simplemente conecte el Pico a cualquier computadora (incluido Raspberry Pi SBC) a través de USB, luego **arrastre y suelte** el archivo en él. ¡Sí! ¡Así de fácil! Y Raspberry Pi Foundation ha reunido un archivo UF2 descargable para permitirle instalar **MicroPython** más fácilmente. Visite la página de introducción de Raspberry Pi para descargar el archivo necesario. O puede obtener una copia impresa de "Comience con **MicroPython** en Raspberry Pi Pico" y comience su viaje de creación digital.

El sistema operativo oficial, Raspberry Pi OS viene preinstalado con **Thonny** Python IDE que está listo para que comience a escribir código **MicroPython** para Pico. Si está utilizando otro sistema operativo (Windows, macOS u otra distribución de Linux), visite https://**Thonny**.org/ para descargar el IDE e instalarlo.

En la siguiente pagina se muestra tabla con las características del procesador Raspberry utilizando otro sistema operativo (Windows, macOS u otra distribución de Linux), visite https://**Thonny**.org/ para descargar el IDE e instalarlo.

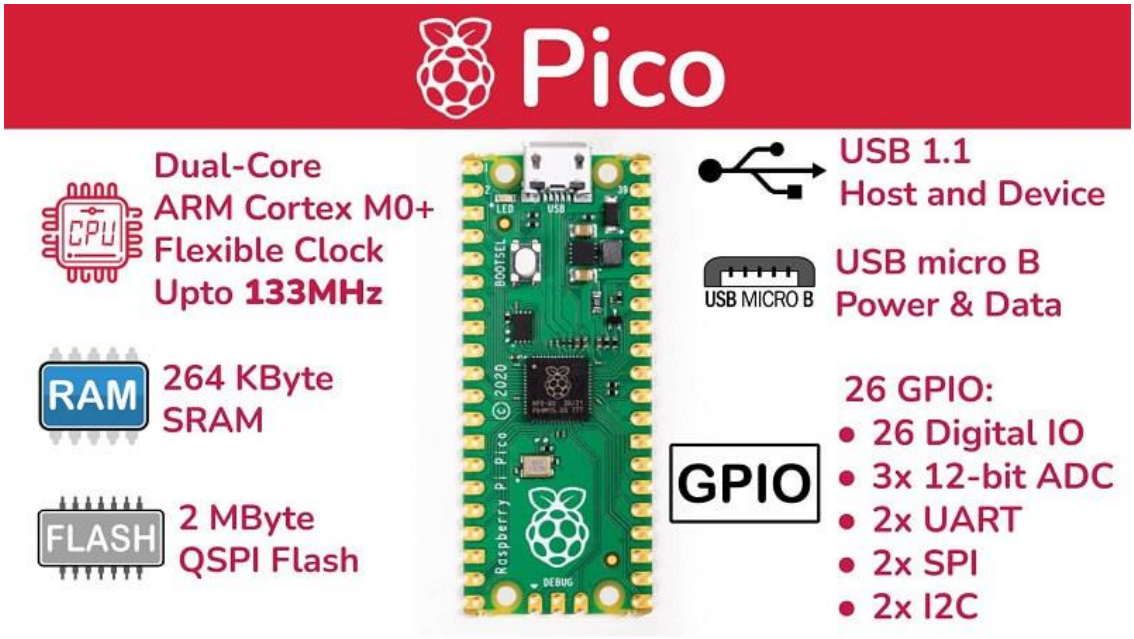
En la siguiente página se muestra tabla con las características del procesador Raspberry Pi Pico

Raspberry Pi Pico



Features/Specs	Raspberry Pi Pico
Release Date	**21st January 2021**
Microcontroller	**RP2040**
Cores	**Dual-Core**
Core Architecture	32-bit ARM **Cortex-M0+**
CPU Clock	Flexible clock upto **133MHz**
RAM Size	**264 KByte SRAM**
Flash Size	**2 MByte Q-SPI Flash**
Programming Language	**MicroPython, C, C++**
Board Power Input	**5VDC** via USB Micro B
Alternative Board Power	**2 - 5VDC** via VSYS Pin (Pin 39)
MCU Voltage	**3.3VDC**
GPIO Voltage	**3.3VDC**
USB Interface	USB 1.1 **Device and Host**
Program Loading	USB Micro B, **USB Mass Storage**
GPIO	**26 x Digital** Input/Output (Total)
ADC	**3 x 12-bit** 500ksps
Temperature Sensor	**Built-in, 12-bit**
UART	**2 x UART**
I^2C	**2 x I^2C**
SPI	**2 x SPI**
PWM	**16 x PWM**
Timer	1 x Timer with 4 x Alarm
Real Time Counter	1 x Real Time Counter
PIO	2 x Programmable High Speed IO
On Board LED	1 x Programmable LED (GP25)
On Board Button	1 x BOOTSEL Button
Breakout of PCB	40 x 0.1" (100mil) Standard Header Pads 40 x 0.1" (100) **Castellations Edge** (SMT Friendly)
Debug Port	3-pin ARM Serial Wire Debug (SWD) Port

Lo más destacado del procesador:

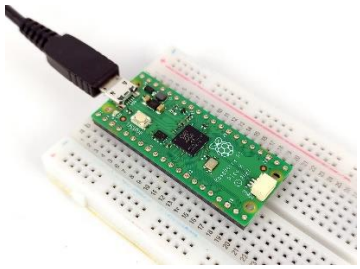


La 1ª plataforma MCU por la Fundación Raspberry Pi

¡1ª plataforma de microcontrolador con el primer silicio MCU diseñado internamente, **RP2040**! Sin embargo, el equipo de Raspberry Pi nunca deja de sorprendernos con su trabajo. Esta pequeña placa viene con un procesador de **doble núcleo de** 32 bits, ARM Cortex M0 + es un superconjunto optimizado del Cortex-M0. El Cortex-M0+ tiene compatibilidad completa del conjunto de instrucciones con el Cortex-M0, lo que permite el uso del mismo compilador y herramientas de depuración. La tubería Cortex-M0+ se redujo de 3 a 2 etapas, lo que reduce el uso de energía. Además del Cortex M0+ de doble núcleo, Raspberry Pi Pico viene con una velocidad de reloj reconfigurable, con el PLL (Phase-Locked Loop) en chip, que permite que el MCU se cronometre a una velocidad máxima de **133MHz**, por supuesto, es necesaria una configuración.

Compacto y listo para ser montado en el producto

Raspberry Pi Pico no solo es súper asequible, sino que también está listo para integrarse en cualquier producto listo para usar. Si elige la versión sin cabezales presoldados, está listo para SMT (Surface Mount Technology). Raspberry Pi Pico se extiende a 40 pines 21x51 DIP (paquete dual en línea), PCB de 1 mm de grosor con pines de orificio pasante de 0.1 "(100 mil). Los pines se extienden aún más al borde de la PCB con una placa de circuito almenada. Esto permite que se suelde a otra placa de PCB sin la necesidad de pines de cabezal adicionales, lo que lo convierte en un producto terminado más pequeño y compacto. ¡Genial!

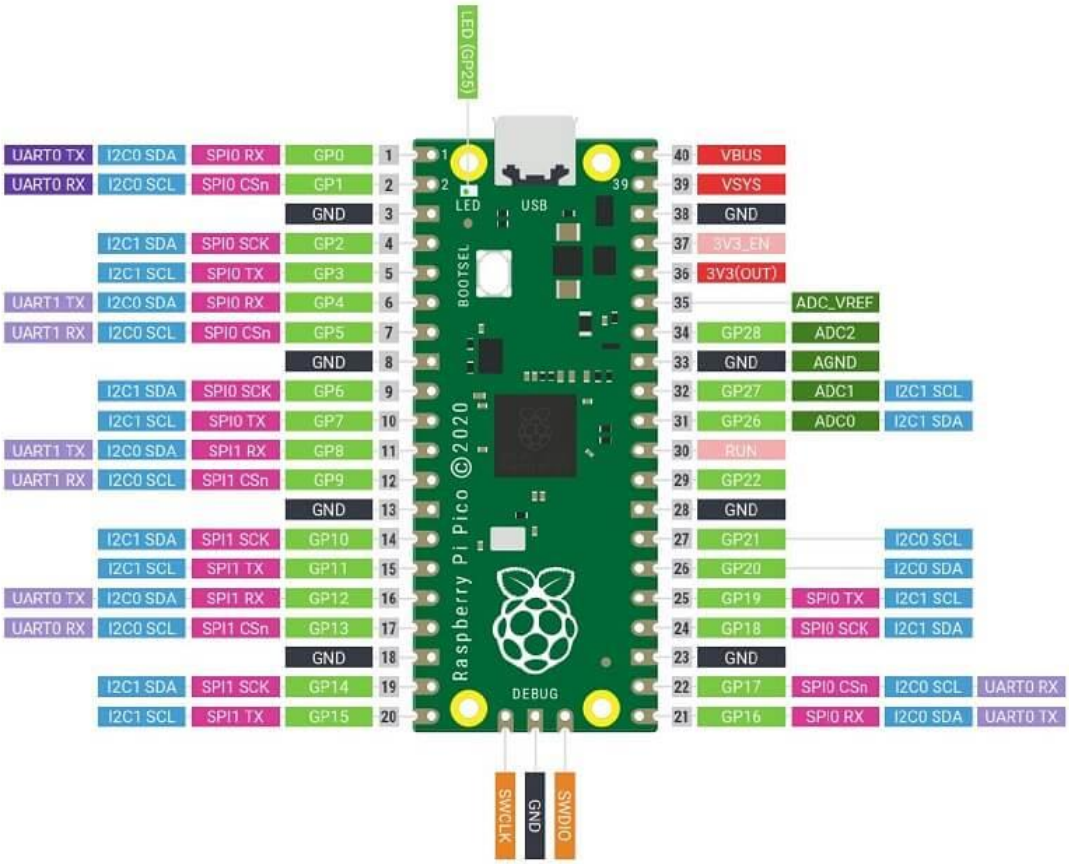


Mientras tanto, la versión con cabezales pre-soldados es amigable con la placa de pruebas, por lo que los estudiantes, fabricantes e ingenieros pueden usar la Raspberry Pi Pico en una placa de prueba o cualquier placa PCB estándar para desarrollo o creación de prototipos.

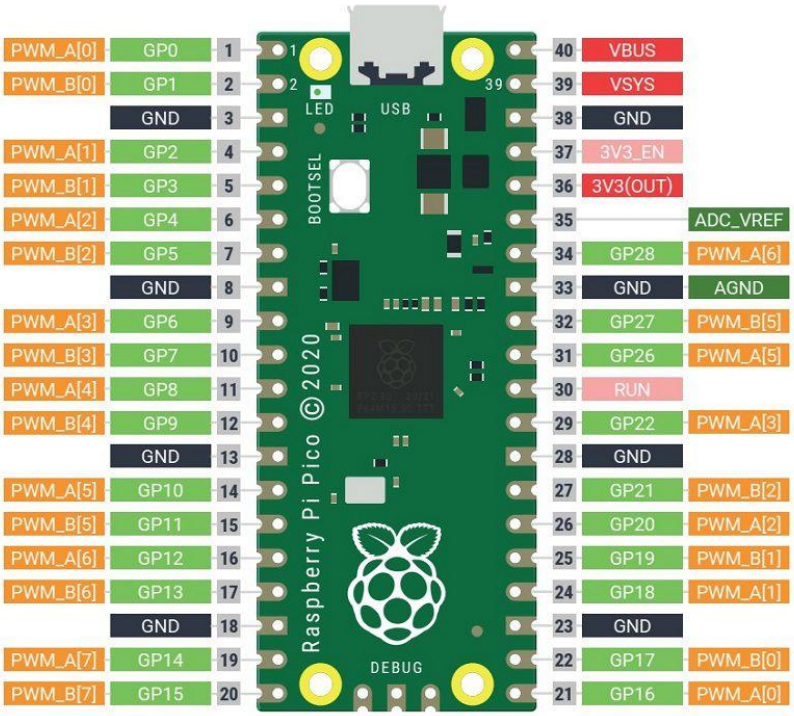
USB Micro B para alimentación y datos

Raspberry Pi Pico incorpora el famoso y comúnmente utilizado receptor USB Micro B tanto para alimentación como para datos. Simplemente obtenga un cable USB Micro B que normalmente viene con un teléfono Andriod o un banco de energía para alimentarlo y cargar el programa en él. No se necesita ningún adaptador USB a serie adicional. ¡Pulcro!

MCU rico en periféricos



GPIO, ADC, UART, SPI, I2C de Raspberry Pi Pico

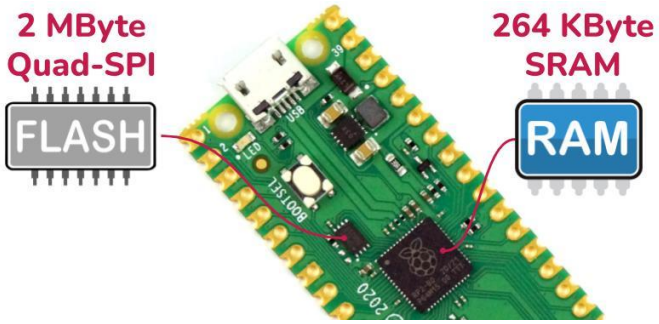


PWM Pins de Raspberry Pi Pico

Con 26 GPIO (3.3V) desglosados para aplicaciones, tiene más pines GPIO que Arduino UNO, Arduino NNO o incluso Arduino MKR Zero. Entre estos 26 GPIO, 3 se pueden configurar como ADC de 12 bits con 500ksps (kilo de muestra por segundo), 2 x UART, 2 x SPI, 2 x I^2C y hasta 16 x PWM pin. Internamente, también viene con 1 x temporizador con 4 alarmas y 1 x contador en tiempo real. Sin olvidar los periféricos duales de E/S programables (PIO) que son E/S de alta velocidad flexibles y programables por el usuario. Puede emular interfaces como tarjetas SD y VGA.

Nota: El Raspberry Pi Pico GPIO se ejecuta en **3.3VDC.** El voltaje máximo que los pines de E/S pueden tolerar es de 3.3V. La aplicación de voltajes superiores a 3.3V a cualquier pin de E/S podría dañar la placa.

Gran tamaño de RAM y flash

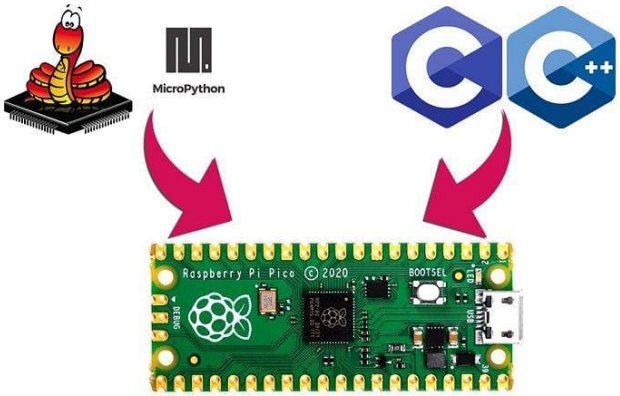


Con **2 MByte** de QSPI Flash externo y **264 KByte** de SRAM en **Raspberry Pi Pico**, nunca le preguntará por no tener suficiente memoria :) Además, el gran tamaño de RAM y Flash también permite que **Raspberry Pi Pico** sea compatible con lenguajes de programación superiores como **MicroPython** o incluso **Javascript**.

Método de carga del programa de arrastrar y soltar

Con el receptor USB Micro B listo como conexión física a un ordenador y el USB 1.1 PHY en el RP2040 (MCU), la Raspberry Pi Pico ofrece un método de carga de programas simple y directo. Es como copiar archivos de una unidad a otra. ¡El Pico aparece como almacenamiento masivo USB cuando se conecta a la computadora a través del puerto USB! ¡Se convierte en una unidad USB! Escriba el código y arrastre el archivo a esa unidad USB. Después de que el archivo se haya copiado por completo, Pico se reiniciará y ejecutará el programa :) Fácil, ¿verdad?

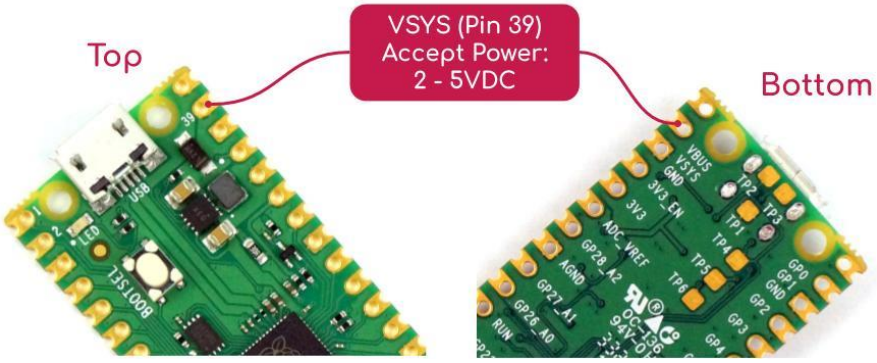
Soporta **MicroPython**, C y C++



Python es uno de los lenguajes de programación más famosos y poderosos de la actualidad. Se está utilizando en muchas aplicaciones de alto nivel, como IA (inteligencia artificial), DL (aprendizaje profundo) y desarrollo web e Internet, y más. Python se utiliza con éxito en miles de aplicaciones empresariales del mundo real en todo el mundo, incluidos muchos sistemas grandes y de misión crítica. **MicroPython** es una implementación eficiente y eficiente del lenguaje de programación Python 3 que incluye un pequeño subconjunto de la biblioteca estándar de Python y está optimizada para ejecutarse en microcontroladores y en entornos restringidos. Te encantará.

Además de **MicroPython**, Raspberry Pi Pico también es compatible con C y C ++ Programming Langauge. Consulte el SDK de C/C++ para obtener más información. Todos estos lenguajes de programación se cargan en Raspberry Pi Pico a través de USB Mass Storage que permite el método simple de arrastrar y soltar (como copiar un archivo a otra unidad).

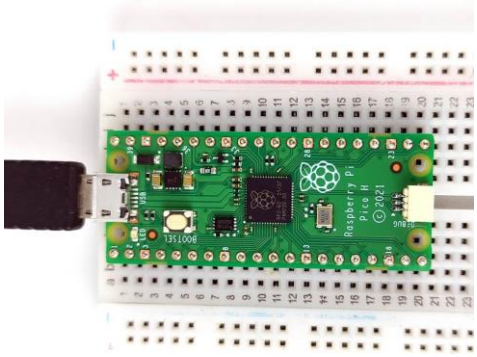
Entrada de energía flexible



El receptor USB Micro B es la entrada de alimentación principal para que la Raspberry Pi Pico "funcione", simplemente conecte el cable USB a cualquier puerto USB y suministrará la energía necesaria para que el MCU ejecute el programa. En el caso de que no desee utilizar el puerto USB, por ejemplo, un producto alimentado por batería o personalizado; No se preocupe, Raspberry Pi Pico viene con una fuente de alimentación de modo de conmutación (SMPS) flexible a bordo que es capaz de aceptar **entradas de 2 a 5VDC** y la convierte en una fuente estable de 3.3V para que funcione el MCU RP2040.

¡Simplemente increíble! El pin es **VSYS** (Pin 39). Con un amplio rango de voltaje, Raspberry Pi Pico puede ser alimentado por USB, 2 x pilas AA, 2 pilas NiMH AA, ¡1 x 18650 Li-ion o 1 x pila LiPo de celda!

Raspberry Pi Pico se inserta en una placa de pruebas, hay dos filas de orificios de puente a ambos lados de la PCB para la creación de prototipos.



¿Qué necesitará para comenzar?

Dado que Raspberry Pi Pico es la 1ª plataforma MCU de la Fundación Raspberry Pi, todos son nuevos en ella. ¡No temas! Es una placa de microcontrolador bastante amigable para principiantes. Aquí están nuestras recomendaciones para el principiante, fabricante e ingeniero:

1. Obtenga el Raspberry Pi Pico con cabezales pre-soldados si desea una experiencia sin problemas para comenzar.
2. Un cable USB Micro B, al menos necesitará este cable para alimentar y cargar el programa en Raspberry Pi Pico.
3. Breadboard 8.5x5.5cm (400 agujeros), puede ayudar a sostener la Raspberry Pi Pico y puede extender el GPIO para la creación de prototipos.
4. Official Get Started with **MicroPython** on Raspberry Pi Pico-Color Printed, una guía completa de Raspberry Pi para comenzar con esta nueva plataforma MCU.
5. Maker Pi Pico es una placa de desarrollo para principiantes que extendió todo el GPIO de Pico a cabezales, conectores Grove, ranura para tarjetas MicroSD, zócalo ESP-01 (WiFi), indicadores LED, botones integrados, LED RGB (Neo-Pixel).
6. Raspberry Pi Pico Basic Kit sin Pico, un kit electrónico basado en la guía oficial, **sin el Pico y Serial LCD y RGB LED**
7. Raspberry Pi Pico Basic Kit - con Pico, un kit electrónico completo basado en la guía oficial, **sin el LCD serie y el LED RGB**

Funciones

- **1ª placa de desarrollo de microcontroladores** de la Fundación Raspberry Pi
- **1st Silicon (IC), MCU RP2040** diseñado desde cero por ingenieros de Raspberry Pi Foundation
- **Procesador ARM Cortex M0+ de doble núcleo y 32 bits**

- Reloj flexible, **configurable máximo a 133MHz**
- Preparado con receptor USB Micro B para alimentación y datos
- **Soporte USB 1.1 host y dispositivo**
- Conectado al puerto USB y aparecerá como **almacenamiento masivo USB** de forma predeterminada, no se necesita ningún controlador
- Soporta **CircuitPython, MicroPython, C y C++,** lenguaje **de programación Arduino IDE**
- Método de carga del programa de arrastrar y soltar, al igual que mover archivos en el Explorador de Windows
- Viene en PCB de 40 pines de 1 mm de grosor estilo 'DIP' 21x51 con **pines de orificio pasante de 0.1 ", amigable con la placa de pruebas**
- Con las **castellaciones de borde** PCB, está listo para ser montado en otra PCB sin un pin de cabezal adicional, compatible con SMD
- Dos opciones:
    - Viene con pines de cabezal PRE-SOLDADOS, amigable con la placa de pruebas
    - Viene sin cabecera, compatible con SMD
- Rico periférico:
    - Salida extendida 26 **E/S multifunción de 3,3 V de uso general (GPIO)**
    - 23 GPIO son solo digitales
    - **3 ADC de 12 bits capaz de 500Ksps**, convertidor analógico a digital
    - 2 x UART (Receptor/transmisor asíncrono universal)
    - 2 x SPI (interfaz periférica serie)
    - 2 x I2C (Inter IC)
    - **16 x PWM** (modulación de ancho de pulso)
    - 1 x temporizador con 4 alarmas
    - 1 x contador en tiempo real
    - **2 x E/S programables (PIO)** que pueden emular interfaces de alta velocidad como tarjeta SD o VGA
    - Sensor de temperatura ADC de 12 bits integrado
- Puerto de depuración de cable serie (SWD) ARM de 3 pines
- **LED programable integrado, GP25**
- Arquitectura de fuente de alimentación simple pero altamente flexible
    - Admite alimentación USB, fuente externa (2 - 5VDC) o incluso energía de la batería.
- SDK completo, ejemplos de software y documentación
- Oficialmente de la Fundación Raspberry Pi
- Compatible con cualquier ordenador con puerto USB, Windows, macOS, Linux
- Funciona sin problemas con Raspberry Pi 4 Model B, o Raspberry Pi 400 y Raspberry Pi OS
- Dimensiones: 51mm x 21mm x 1mm

Recursos

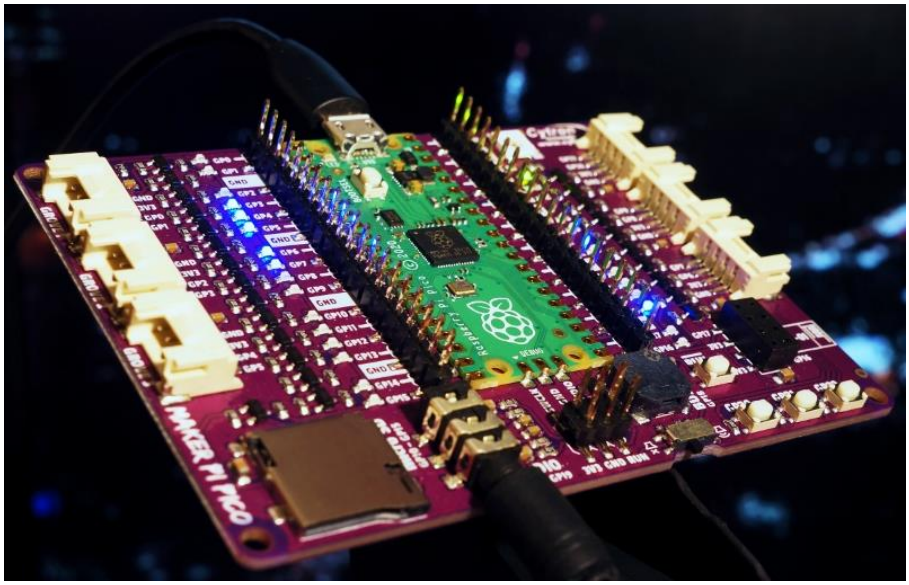
- Introducción a Raspberry Pi Pico (URL)
- Introducción a Raspberry Pi Pico (pdf), Desarrollo de C/C++ con Pico y otras placas de microcontroladores basadas en RP2040
- Hoja de datos de Raspberry Pi Pico (pdf), una placa de microcontrolador basada en RP2040

- SDK de Pico Python (pdf), Un entorno de **MicroPython** para el microcontrolador RP2040
- SDK de Pico C/C++ (pdf), Bibliotecas y herramientas para el desarrollo de C/C++ en el microcontrolador RP2040
- RP2040 Hoja de datos (pdf), Un microcontrolador por Raspberry Pi
- Un nuevo retador en la plataforma MCU: Raspberry Pi Pico, un artículo sobre Raspberry Pi Pico y RP2040
- Raspberry Pi Pico Vs Arduino UNO R3, un artículo para comparar las especificaciones de RPi Pico y Arduino UNO R3

3. Tarjetas **Cytron**

Tarjeta Maker Pi Pico

Es un producto certificado **Powered by Raspberry Pi**.

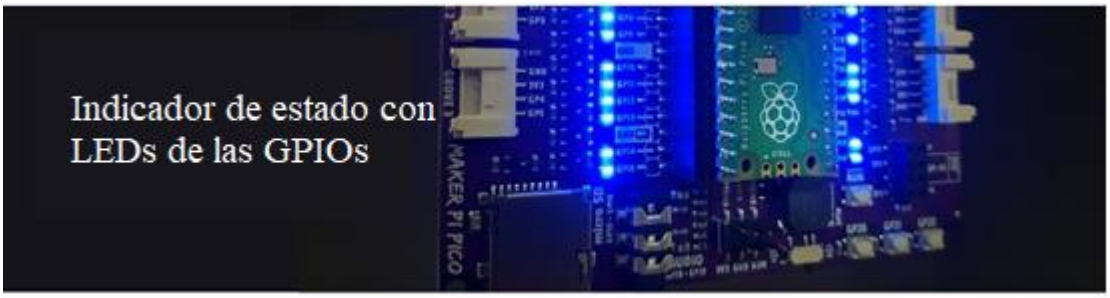


Crédito de la foto: Kevin J. Walters

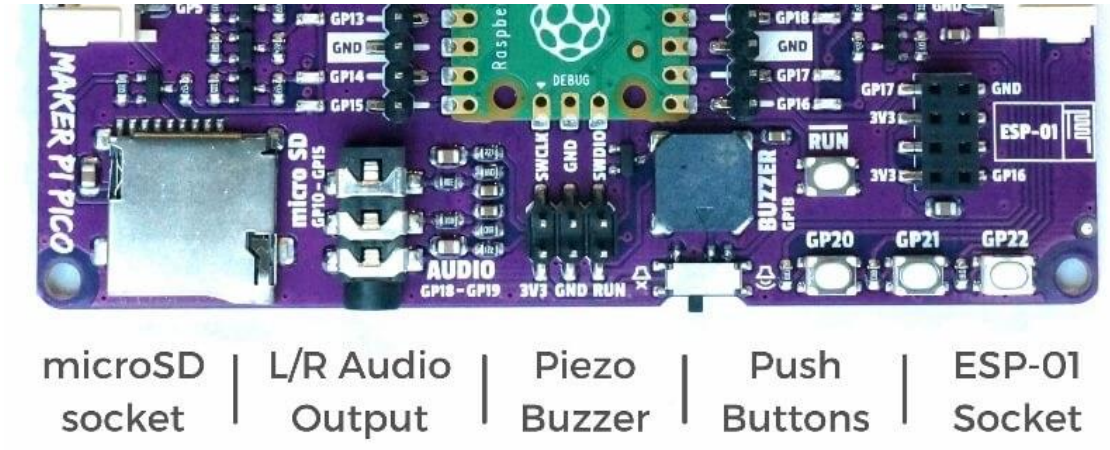
Maker Pi Pico y Maker Pi Pico Base incorporan el botón de reinicio más buscado para Raspberry Pi Pico o Pico W **y** le da acceso a todos los pines GPIO en dos cabezales de pines de 20 vías, con etiquetas claras. Cada GPIO se combina con un indicador LED para realizar pruebas de código y solucionar problemas de manera conveniente. La capa inferior de esta placa incluso viene con un diagrama de pines completo que muestra la función de cada pin.

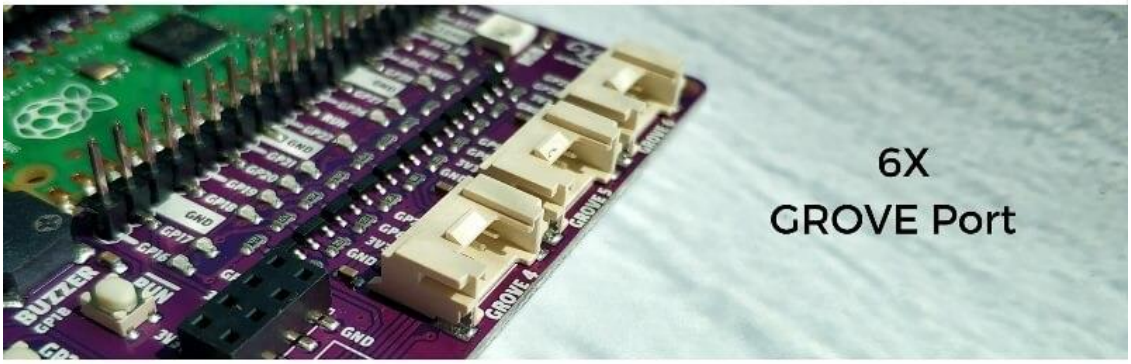
Nota Para los fabricantes internacionales, consulte a nuestros **socios internacionales** que llevan Maker Pi Pico Base.

Sus características más destacables

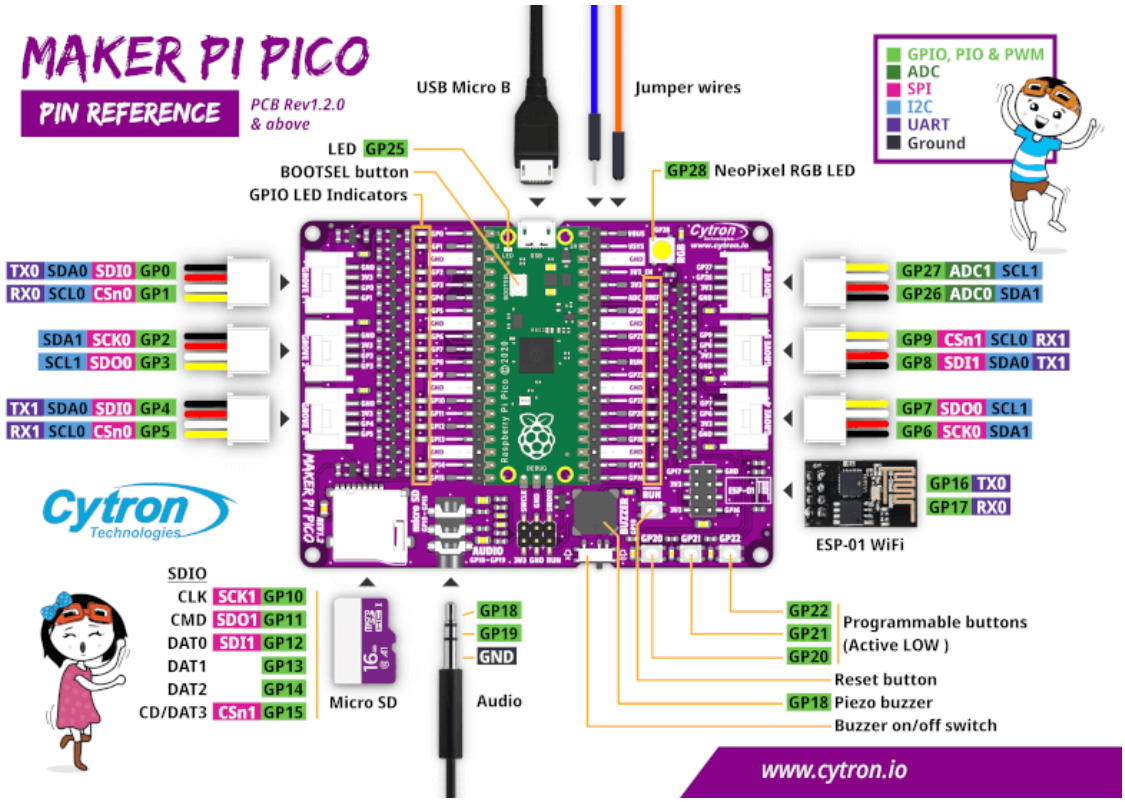


Detalle de algunas de las partes mas importantes de la tarjeta. Los conectores son compatibles con el estándar Grove





Es una placa compacta pero repleta de funciones diseñada para sentarse en el corazón de sus proyectos de STEM y robótica. Simplemente use los cables de puente y / o cables Grove para conectar cualquier sensor y módulos de salida para ampliar su capacidad. ¡No se requiere soldadura!



Maker Pi Pico se puede programar con **CircuitPython**, **MicroPython** y C / **C++.** Conéctese a cualquier computadora (incluida Raspberry Pi SBC) a través de USB, luego **arrastre y suelte** el archivo para cargar el programa. ¡Es tan fácil como copiar un archivo en su memoria USB!

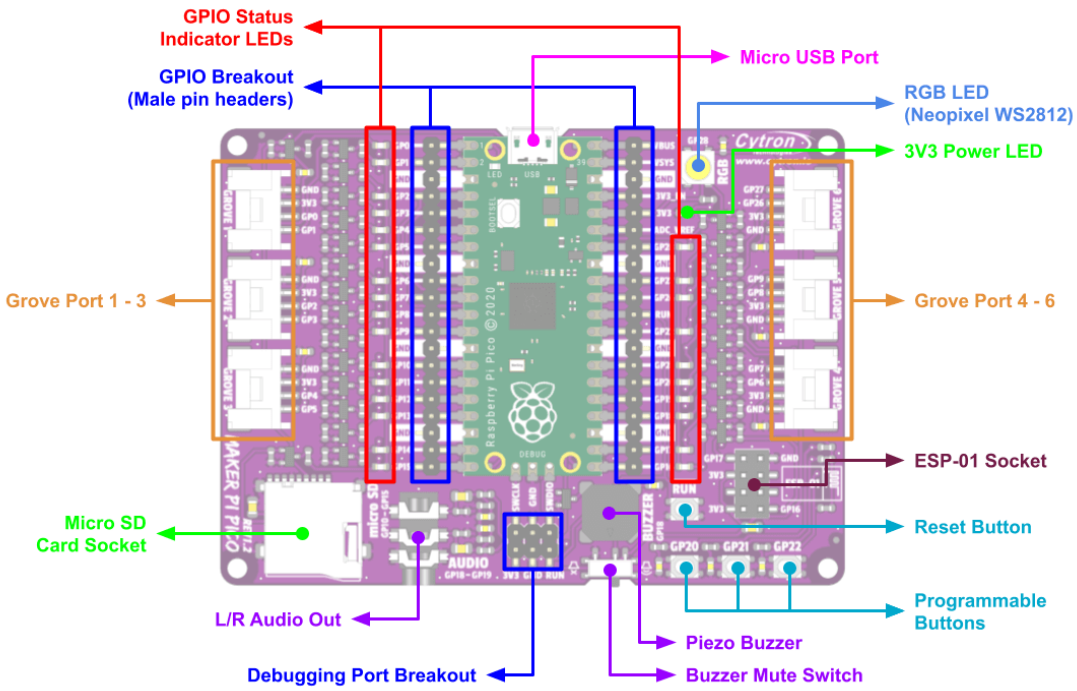
¿Qué es Raspberry Pi Pico W?

Es la Raspberry Pi Pico con :) inalámbrica incorporada Lo mejor de todo es que tiene la misma asignación de 40 pines, dimensión y PCB de borde almenado que la Raspberry Pi Pico. ¡La Raspberry Pi Pico W es un reemplazo directo de Raspberry Pi Pico si necesita actualizar su producto a Wireless o IoT! Por supuesto, Raspberry Pi Pico W se puede utilizar para proyectos de computación física donde controla cualquier cosa, desde pequeños componentes electrónicos, LED y motores; para leer información de sensores o comunicarse con otros microcontroladores.

Raspberry Pi Pico W

Features/Specs	Raspberry Pi Pico W
Release Date	**30th June 2022**
Microcontroller	**RP2040**
Cores	**Dual-Core**
Core Architecture	32-bit ARM **Cortex-M0+**
CPU Clock	Flexible clock upto **133MHz**
RAM Size	**264 KByte SRAM**
Flash Size	**2 MByte Q-SPI Flash**
Wireless	**IEEE 802.11 b/n/g (2.4GHz), Bluetooth 5.2**
Antenna	On board PCB Antenna
Programming Language	**MicroPython, CircuitPython, C, C++**
Board Power Input	**5VDC** via USB Micro B
Alternative Board Power	**2 - 5VDC** via VSYS Pin (Pin 39)
MCU Voltage	**3.3VDC**
GPIO Voltage	**3.3VDC**
USB Interface	USB 1.1 **Device and Host**
Program Loading	USB Micro B, **USB Mass Storage**
GPIO	**26 x Digital** Input/Output (Total)
ADC	**3 x 12-bit** 500ksps
Temperature Sensor	**Built-in, 12-bit**
UART	**2 x UART**
I^2C	**2 x I^2C**
SPI	**2 x SPI**
PWM	**16 x PWM**
Timer	1 x Timer with 4 x Alarm
Real Time Counter	1 x Real Time Counter
PIO	2 x Programmable High Speed IO
On Board LED	1 x Programmable LED (GP25)
On Board Button	1 x BOOTSEL Button
Breakout of PCB	40 x 0.1" (100mil) Standard Header Pads 40 x 0.1" (100) **Castellations Edge** (SMT Friendly)
Debug Port	3-pin ARM Serial Wire Debug (SWD) Port

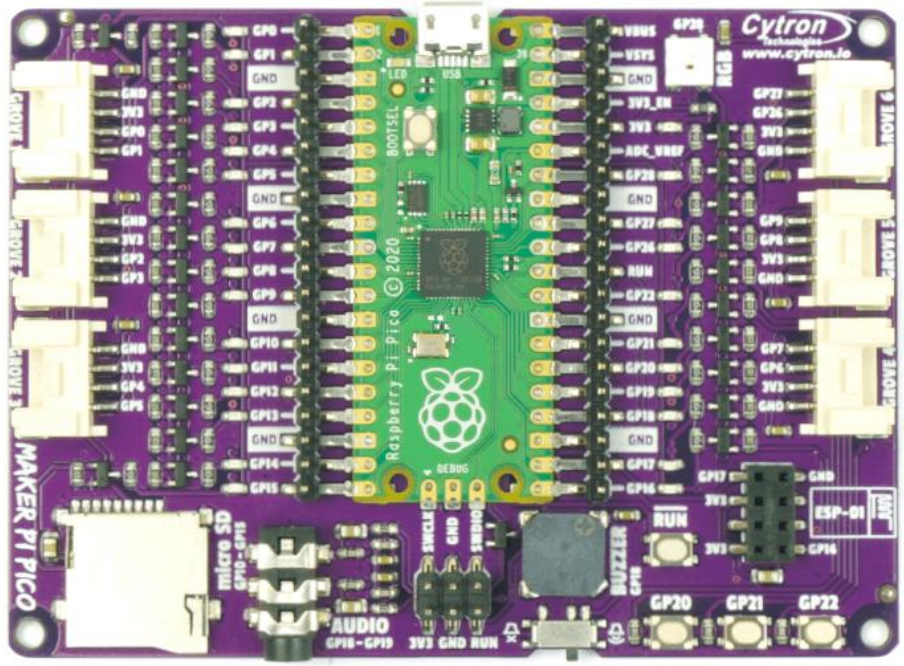
Diseño de la placa:

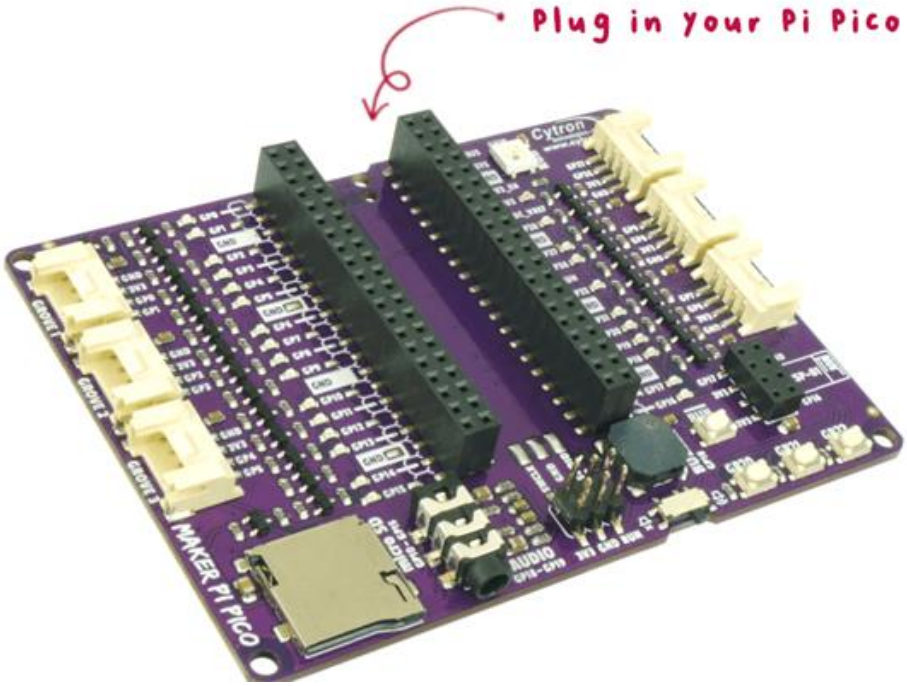


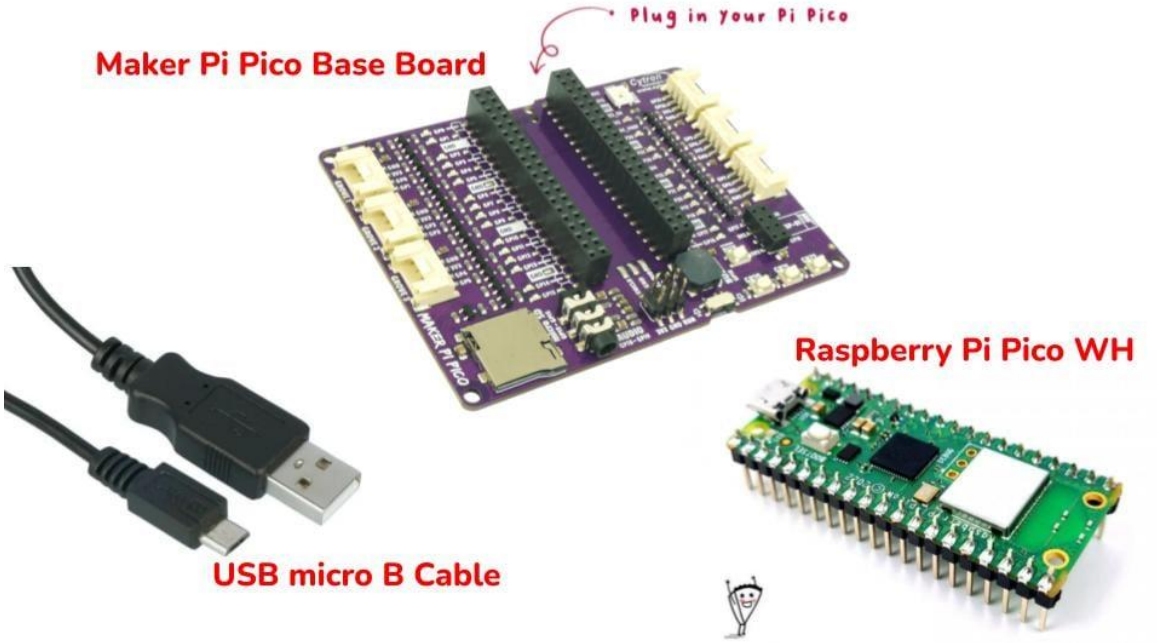
Características más destacables

- Trabaje fuera de la caja. ¡Sin soldadura!
- Acceso a todos los pines de Raspberry Pi Pico o Pico W en dos cabezales de pines de 20 vías
- Indicadores LED en todos los pines GPIO
- 3x pulsador programable (GP20-22)
- 1x LED RGB - NeoPixel (GP28)
- 1x zumbador piezoeléctrico (GP18)
- 1x conector de audio estéreo de 3,5 mm (GP18-19)
- 1x ranura para tarjeta Micro SD (GP10-15)
- 1x zócalo ESP-01 (GP16-17)
- 6x puerto Grove
- **Maker** Pi Pico Base con **Raspberry Pi Pico WSH (inalámbrico y presoldado con cabezales)**
    - 1 x Maker Pi Pico Base (Raspberry Pi Pico H o Pico WSH **NO está presoldado** a bordo)
    - 1 x Raspberry Pi Pico WSH (cabezales presoldados)

- 1 x cable USB micro B







Recursos

- Hoja de datos de Maker Pi Pico (pdf)
- Esquema de Maker Pi Pico Rev1.2.0
- Demostración de Maker Pi Pico y código de ejemplo
- CAD 3D
- Página oficial de Raspberry Pi Pico
- Primeros pasos con Raspberry Pi Pico
- Hoja de datos de Raspberry Pi Pico (pdf)
- Hoja de datos de Raspberry Pi Pico W (pdf)
- Hoja de datos de RP2040
- Raspberry Pi Pico Python SDK
- Raspberry Pi Pico C/C++ SDK

Tarjeta MAKER PI RP2040

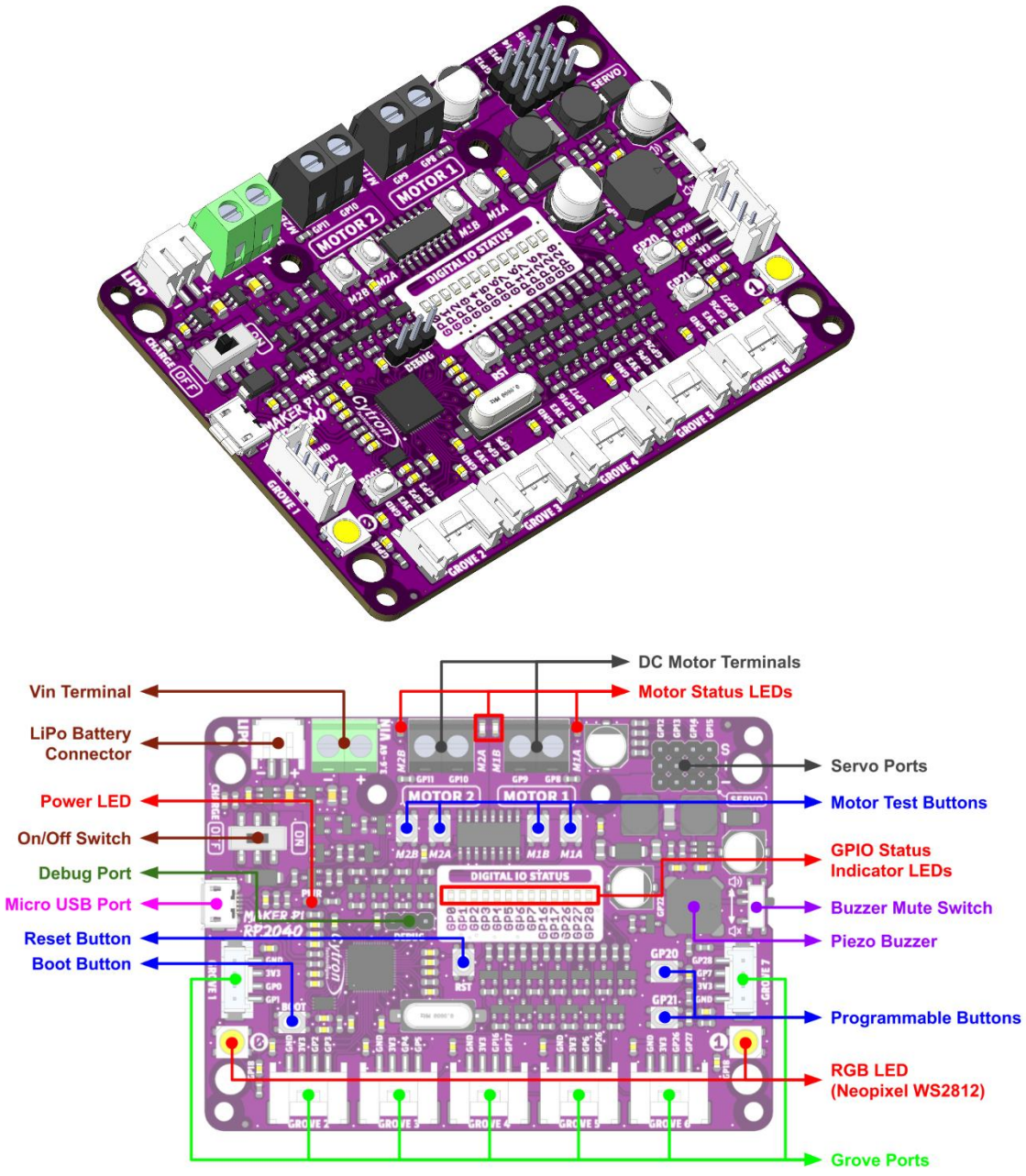


Figure 1: MAKER-PI-RP2040 Board Functions

Función	Descripción
Vin Terminal	Conecte a cualquier fuente de alimentación dentro de 3.6 - 6V.
Conector LiPo Bateria	Conector de batería LiPo Conectar a batería LiPo / Li-Ion de celda única. La batería es recargable a través de USB. * *La batería está protegida contra sobrecargas y descargas excesivas. Si la placa no se puede encender cuando la batería está conectada, cargue la batería para activar el circuito de protección de la batería.*

<table>
<tr><td>LED Encendico</td><td>Se enciende cuando se conecta</td></tr>
<tr><td>Interruptor On/Off</td><td>Enciende/apaga la alimentación.</td></tr>
<tr><td>Puerto depuración</td><td>Puerto de depuración del RP2040.</td></tr>
<tr><td>Puerto Micro USB</td><td>Utilizado para cargar programas desde la PC.
También se puede utilizar para encender la placa</td></tr>
<tr><td>Botón Reinicio</td><td>Presiónelo para reiniciar el RP2040.</td></tr>
<tr><td>Boton Para Modo Cargador</td><td>Mantenga presionado este botón mientras se reinicia el RP2040 para ingresar al modo de cargador de inicio. Se utiliza para cargar el firmware MicroPython/Circuitpython o C/C++ personalizado.</td></tr>
<tr><td>Puertos Grove</td><td>Puertos Grove Conéctese a módulos Grove externos.
<table>
<tr><th>Grove Port</th><th>GPIO</th><th>PWM</th><th>SPI</th><th>I2C</th><th>UART</th><th>Analog</th></tr>
<tr><td rowspan="2">1</td><td>0</td><td>PWM0-A</td><td>SDI0</td><td>SDA0</td><td>TX0</td><td>-</td></tr>
<tr><td>1</td><td>PWM0-B</td><td>CSn0</td><td>SCL0</td><td>RX0</td><td>-</td></tr>
<tr><td rowspan="2">2</td><td>2</td><td>PWM1-A</td><td>SCK0</td><td>SDA1</td><td>-</td><td>-</td></tr>
<tr><td>3</td><td>PWM1-B</td><td>SDO0</td><td>SCL1</td><td>-</td><td>-</td></tr>
<tr><td rowspan="2">3</td><td>4</td><td>PWM2-A</td><td>SDI0</td><td>SDA0</td><td>TX1</td><td>-</td></tr>
<tr><td>5</td><td>PWM2-B</td><td>CSn0</td><td>SCL0</td><td>RX1</td><td>-</td></tr>
<tr><td rowspan="2">4</td><td>16</td><td>PWM0-A</td><td>SDI0</td><td>SDA0</td><td>TX0</td><td>-</td></tr>
<tr><td>17</td><td>PWM0-B</td><td>CSn0</td><td>SCL0</td><td>RX0</td><td>-</td></tr>
<tr><td rowspan="2">5</td><td>6</td><td>PWM3-A</td><td>SCK0</td><td>SDA1</td><td>-</td><td>-</td></tr>
<tr><td>26</td><td>PWM5-A</td><td>-</td><td>SDA1</td><td>-</td><td>ADC0</td></tr>
<tr><td rowspan="2">6</td><td>26</td><td>PWM5-A</td><td>-</td><td>SDA1</td><td>-</td><td>ADC0</td></tr>
<tr><td>27</td><td>PWM5-B</td><td>-</td><td>SCL1</td><td>-</td><td>ADC1</td></tr>
<tr><td rowspan="2">7</td><td>7</td><td>PWM3-B</td><td>SDO0</td><td>SCL1</td><td>-</td><td>-</td></tr>
<tr><td>28</td><td>PWM6-A</td><td>-</td><td>-</td><td>-</td><td>ADC2</td></tr>
</table></td></tr>
<tr><td>RGB LEDs (WS2812)</td><td>LED RGB WS2812B programable por el usuario. Conectado a GP18.</td></tr>
<tr><td>Botones programables</td><td>Accesibles desde el programa de usuario. Conectado a GP20 y GP21</td></tr>
<tr><td>Piezo Buzzer</td><td>Se puede utilizar para reproducir tonos o melodías. Conectado a GP22.</td></tr>
<tr><td>Interruptor Buzer OFF</td><td>Se utiliza para silenciar el zumbador piezoeléctrico.</td></tr>
<tr><td>GPIO Status LEDs</td><td>LED de estado de GPIO Indicadores LED para GPIO RP2040 en puertos Grove.
Encienda cuando el estado de GPIO sea alto.</td></tr>
<tr><td>Botones Test Motor</td><td>Presiónelos para probar la funcionalidad del controlador de motor.
El motor funcionará a toda velocidad.
● MxA: Adelante*
● MxB : Hacia atrás*</td></tr>
<tr><td>Puertos Servo</td><td>Conectores de puertos servo para 4 servomotores RC.
La señal está conectada a GP12, GP13, GP14 y GP15.
El voltaje V+ es igual al voltaje de la fuente de alimentación.</td></tr>
<tr><td>LEDs estado de motor</td><td>LED de estado del motor Se encienden cuando el motor está funcionando.
● MxA: Adelante*</td></tr>
</table>

	● MxB : Hacia atrás*
Terminales Motor DC	Terminales del motor de CC Conecte al terminal del motor. El voltaje del motor a toda velocidad es igual al voltaje de la fuente de alimentación. La dirección del motor depende de la polaridad. ● M1A: GP8 ● M2A: GP10 ● M1B: GP9 ● M2B: GP11

Table 1: MAKER-PI-RP2040 Board Functions

TABLA DE VERDAD DEL CONTROLADOR DE MOTOR

Entrada A (GP8 / GP10)	**Entrada B (GP9 / GP11)**	**Salida A (M1A / M2A)**	**Salida B (M1B / M2B)**	**Motor**
Bajo	**Bajo**	Bajo	Bajo	Freno
Alto	**Bajo**	Alto	Bajo	Adelante*
Bajo	**Alto**	Bajo	Alto	Atras*
Alto	**Alto**	Hi-Z (Open)	Hi-Z (Open)	Costa

Tabla 3: Tabla de verdad del controlador de motor

- *La dirección real del motor depende de la conexión del motor. Cambiar la conexión (MA y MB) invertirá la dirección.*

4. Programando con **MicroPython Thonny**

Para el desarrollo de las tareas de programación de las tarjetas **Cytron** que hemos seleccionad para trabajar con Raspberry Pi Pico será el entorno de programación **Thonny** que es una herramienta de libre distribución y que cumple todas las exigencias para poder trabajar de manera cómoda y sencilla.

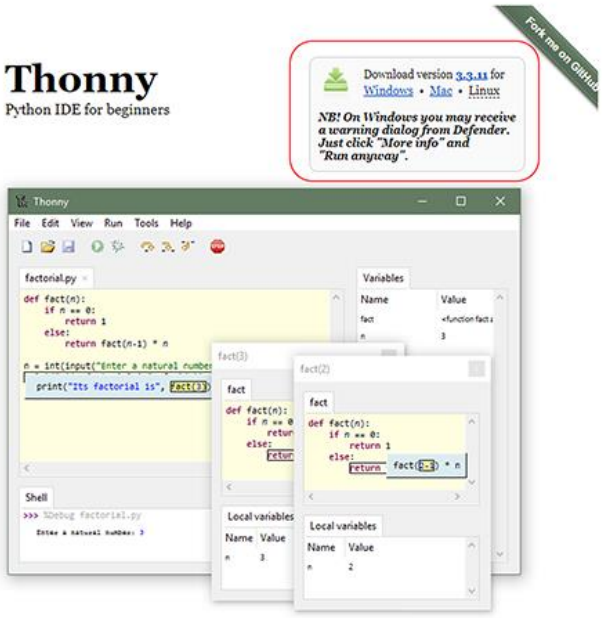
En el siguiente epígrafe vamos a repasar los pasos que se han de dar para poder realizar la programación de nuestras tarjetas

4.1. Proceso de creación de una aplicación

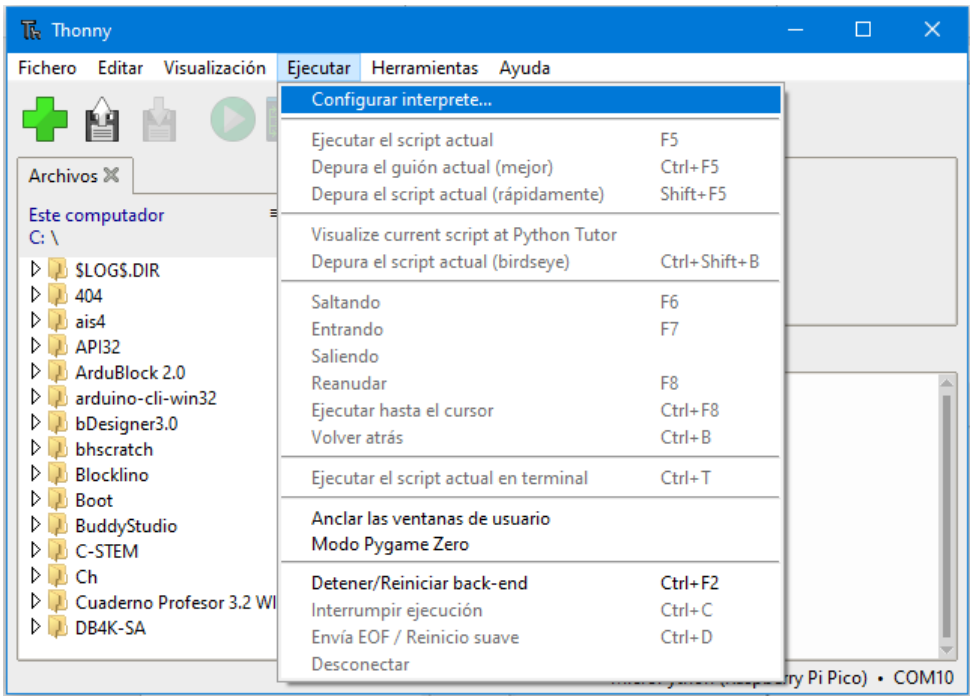
Se reinicia la tarjeta para poder grabar en ella el firmware con el que se comunicara con Tonny

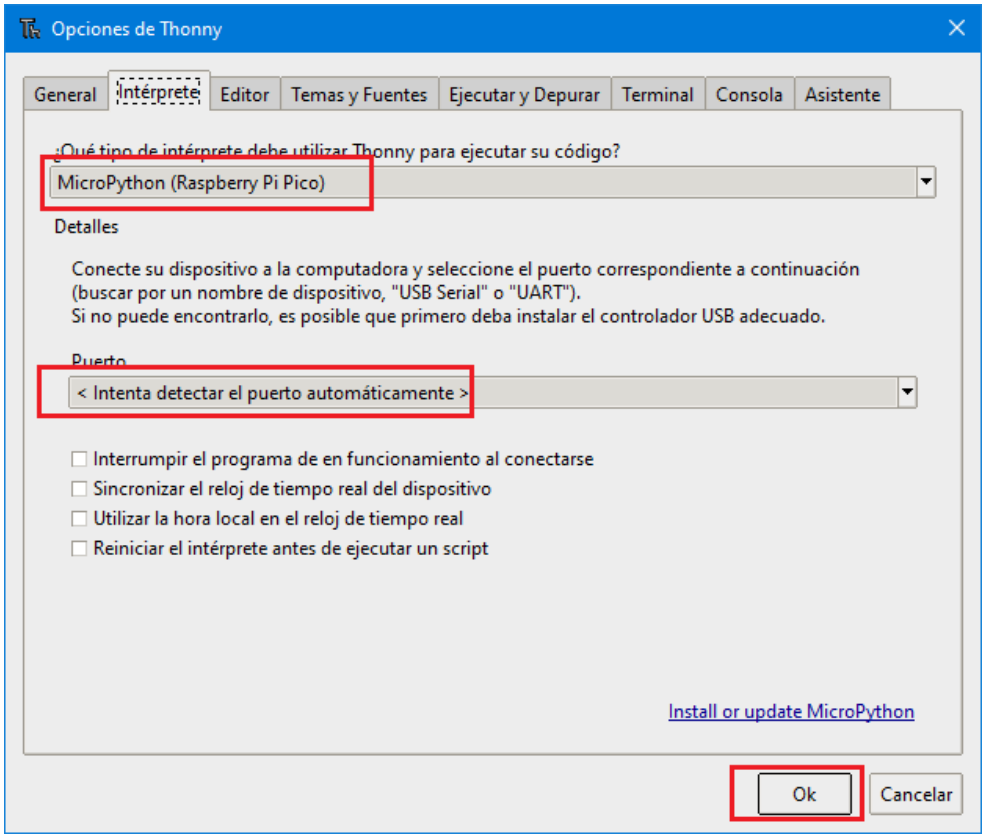
Paso 1: Conecte la placa Pico a uno de los puertos USB de su PC

Paso 2: Si no ha instalado **Thonny** Software o no sabe cómo usar **Thonny** IDE, encontrara información en la página web de la herramienta. Tambien puedes mirar en el Repositorio Github del autor de la herramienta

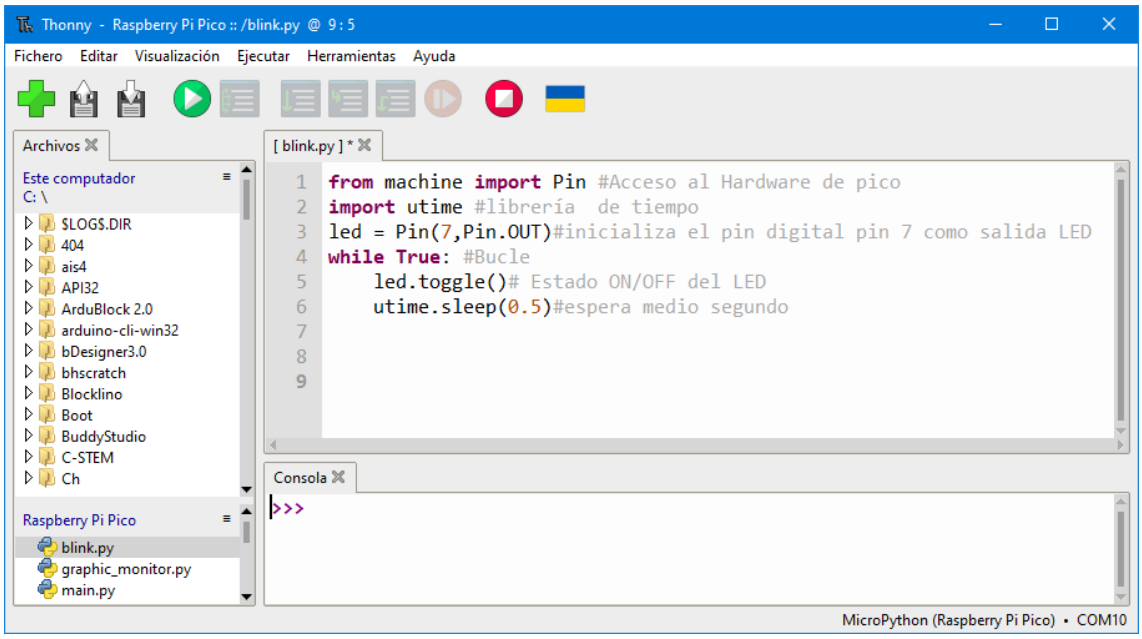


Paso 3: Ahora abra el IDE de **Thonny** Python, haga clic en **Ejecutar-> Configuracion de interprete** para seleccionar **MicroPython para Rapsberry Pi Pico** como intérprete: También seleccione el puerto COM al que está conectada su placa Pico: Después de eso, haga clic en Aceptar para guardar la configuración.


Thonny
Fichero
Editar
Visualización
Ejecutar
Herramientas
Ayuda
Configurar interprete...
Ejecutar el script actual F5
Depura el guión actual (mejor) Ctrl+F5
Depura el script actual (rápidamente) Shift+F5
Visualize current script at Python Tutor
Depura el script actual (birdseye) Ctrl+Shift+B
Saltando F6
Entrando F7
Saliendo
Reanudar F8
Ejecutar hasta el cursor Ctrl+F8
Volver atrás Ctrl+B
Ejecutar el script actual en terminal Ctrl+T
Anclar las ventanas de usuario
Modo Pygame Zero
Detener/Reiniciar back-end Ctrl+F2
Interrumpir ejecución Ctrl+C
Envía EOF / Reinicio suave Ctrl+D
Desconectar
Archivos
Este computador
C: \
LOG.DIR
404
ais4
API32
ArduBlock 2.0
arduino-cli-win32
bDesigner3.0
bhscratch
Blocklino
Boot
BuddyStudio
C-STEM
Ch
Cuaderno Profesor 3.2 WI
DB4K-SA
Pi Pico) • COM10

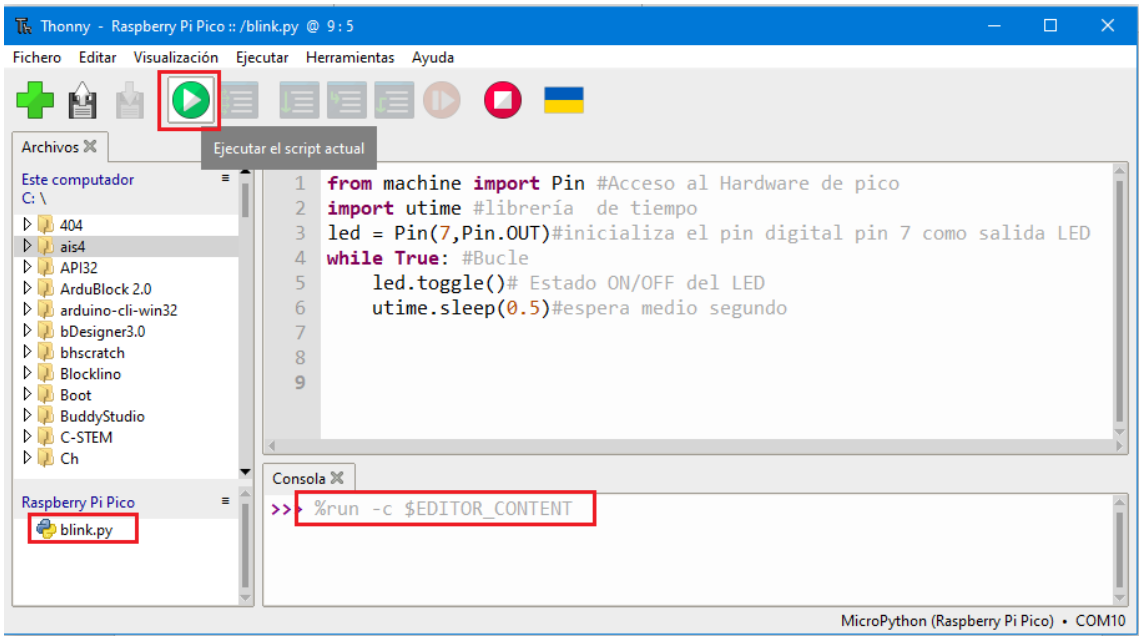

Opciones de Thonny
General
Intérprete
Editor
Temas y Fuentes
Ejecutar y Depurar
Terminal
Consola
Asistente
¿Qué tipo de intérprete debe utilizar Thonny para ejecutar su código?
MicroPython (Raspberry Pi Pico)
Detalles
Conecte su dispositivo a la computadora y seleccione el puerto correspondiente a continuación
(buscar por un nombre de dispositivo, "USB Serial" o "UART").
Si no puede encontrarlo, es posible que primero deba instalar el controlador USB adecuado.
Puerto
< Intenta detectar el puerto automáticamente >
Interrumpir el programa de en funcionamiento al conectarse
Sincronizar el reloj de tiempo real del dispositivo
Utilizar la hora local en el reloj de tiempo real
Reiniciar el intérprete antes de ejecutar un script
Install or update MicroPython
Ok
Cancelar

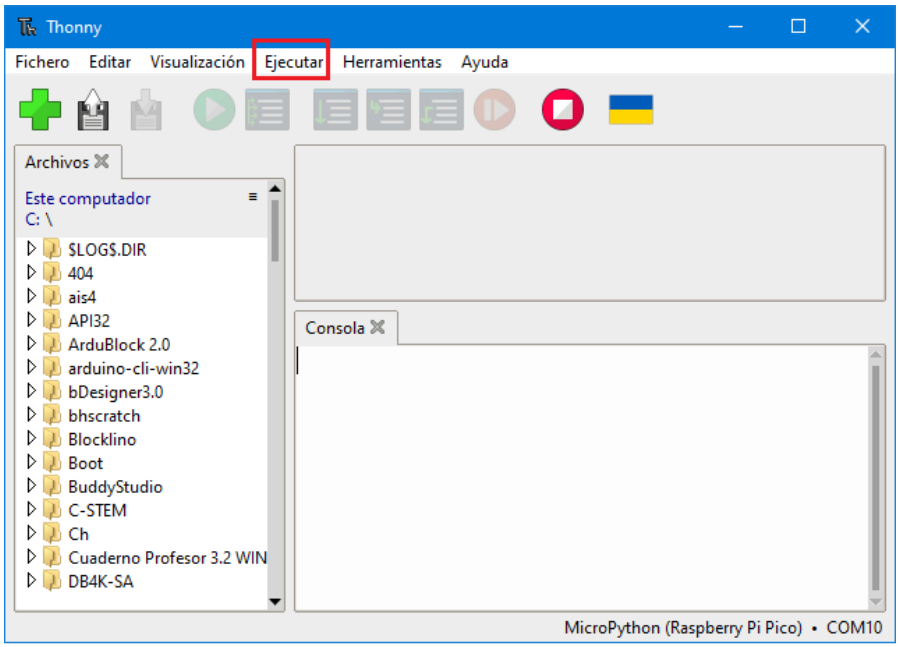
Paso 4: Ahora escribe el Código Python en la siguiente ventana que se muestra:




```
from machine import Pin #Acceso al Hardware de pico
import utime #librería  de tiempo
led = Pin(7,Pin.OUT)#inicializa el pin digital pin 7 como salida LED
while True: #Bucle
    led.toggle()# Estado ON/OFF del LED
    utime.sleep(0.5)#espera medio segundo
```

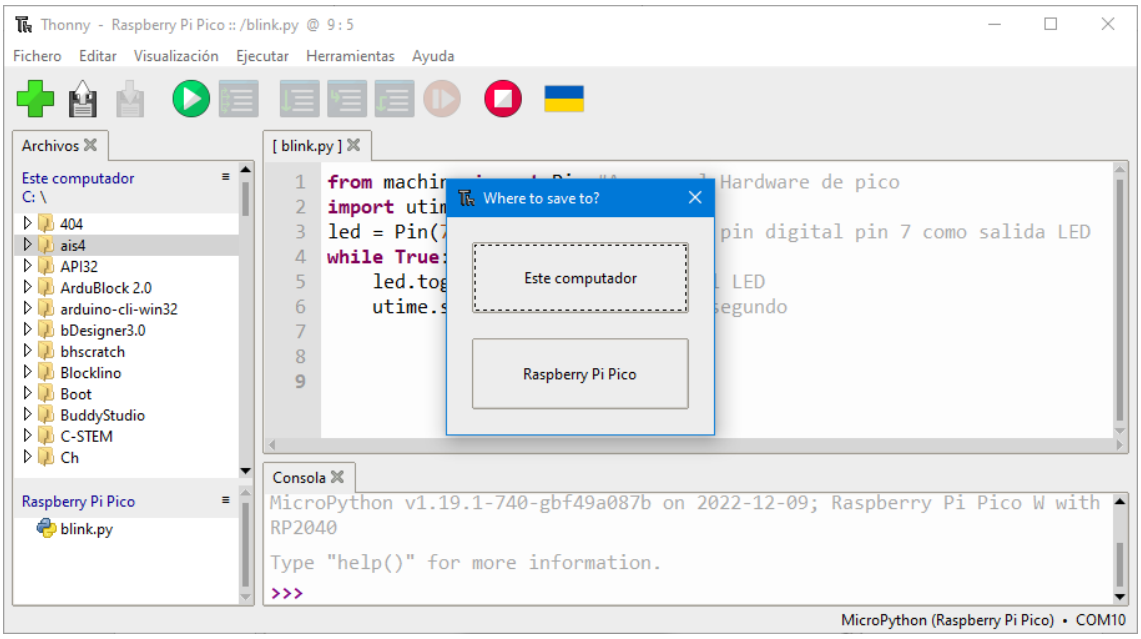
Paso 5: Haz click en el pequeño botón ► y programa se ejecutará y se cargará el fichero en la tarjeta Raspberry Pi Pico



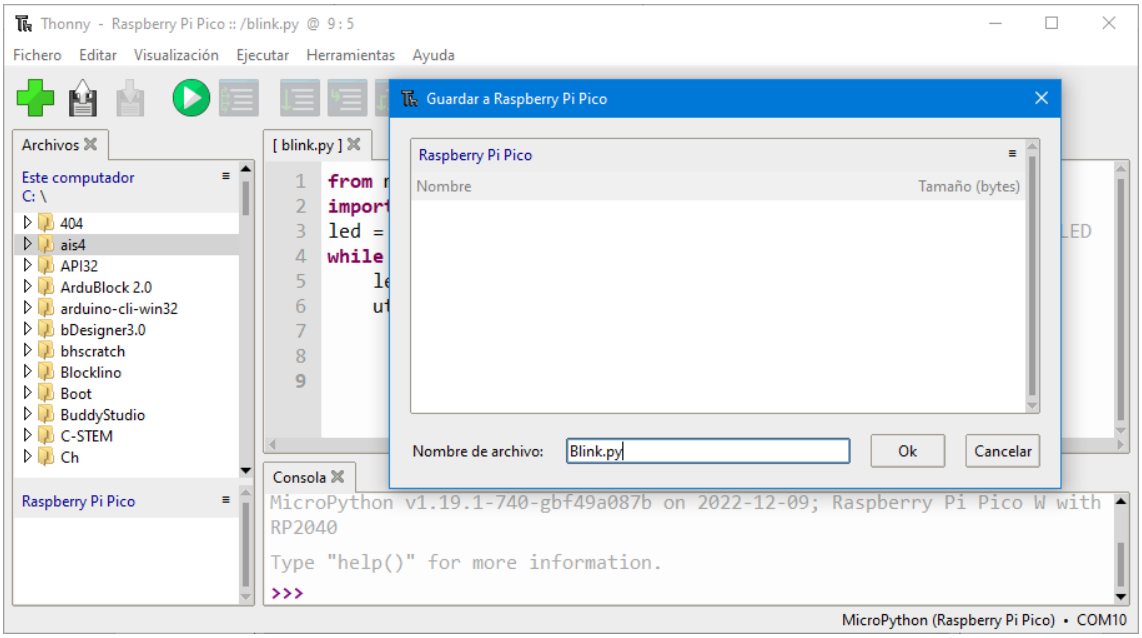


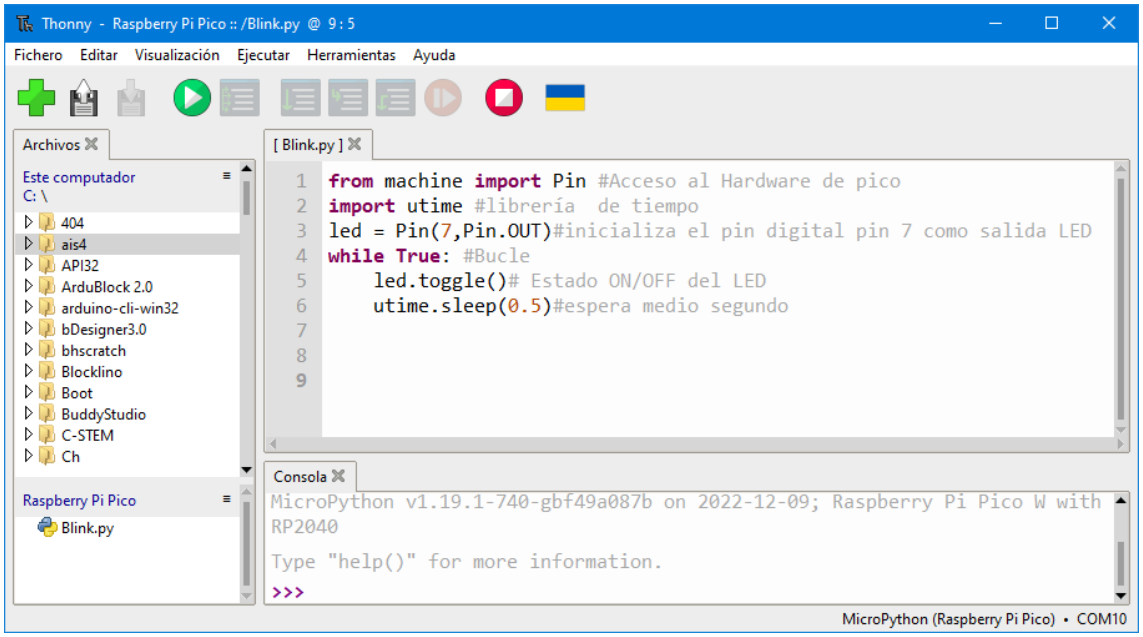
Si nuestra aplicación llevara varios ficheros podemos grabarlos manualmente. Para grabar puedes pulsar en la opción “Guardar como” del menú de Fichero y te aparecerá la siguiente ventana; selecciona Raspberry Pi Pico como destino.

La grabación podemos derivarla al PC o a la unidad de la tarjeta.



Step 6 : Escribe el nombre del fichero *pico-lesson2.py*, y pulsa OK





Paso 7: Haga clic en el pequeño botón ► de nuevo para ejecutar el código Python.

El resultado será

Ahora puede ver que el LED colocado en el PIN GP7 de la tarjeta se enciende y apaga.

5. Estructura básica de un programa. Uso de librerías

Nuestros programas van a tener una estructura muy parecida. De manera general las **PARTES** de un programa serán:

- Carga de las librerías
- Definición de las variables
- Creación de los "Objetos" que vayan a ser usados en nuestro programa
- Definición de funciones propias en el programa (si es el caso)
- Bucle de ejecución del nuestro programa

Existen numerosos manuales y tutoriales en los que podrás aprender Python para poder ir incorporando nuevas funciones y recursos de este poderoso lenguaje de programación.

Las **LIBRERÍAS** son programas que incluyen funciones a las que se puede invocar desde nuestros programas. Existen librerías propias del lenguaje (**MicroPython** y Phyton) que son las que se invocan en la cabecera del programa.

```
from machine import Pin
import time
```

from "**machine**" invoca a la librería "machine" de **MicroPython** y la instrucción **import** combina dos operaciones; busca el módulo con nombre y, a continuación, enlaza los resultados de esa búsqueda a un nombre en el ámbito local.

Usando las palabras reservadas **from** e **import** que provee el sistema, podemos utilizar variables, funciones, clases, excepciones y cualquier otro objeto que estén dentro de un módulo.

En este libro usaremos librerías de desarrolladores orientadas al uso de nuestro Microcontrolador Raspberry Pi Pico

6. Botones y Leds

En este apartado vamos a realizar prácticas en las que intervengan señales de E/S digital.

Contaremos con las señales que en nuestras tarjetas **Cytron** son accesibles a través de los conectores Grove, pero eso no significa que estemos obligados a usar esas y no otras señales de los pines disponibles. Podemos usarlos todos.

En la tarjeta MAKER PI PICO disponemos de los siguientes pines Digitales accesibles a través de los conectores Grove:

Conexiones Grove	Pines dedicados en la tarjeta
GROVE1: GP0 y GP1	Buzzer GP18
GROVE2: GP2 y GP3	Botón GP20
GROVE3: GP4 y GP5	Botón GP21
GROVE4: GP6 y GP7	Botón GP22
GROVE5: GP8 y GP9	Audio GP18 y GP19
GROVE6: GP26 y GP27 (Analogicas)	Tarjeta SSD Memoria GP10 al GP15
	Dispositivo ESP-01 GP16 y GP17
	LED RGB GP28

Normalmente las salidas digitales se conectan a diodos LEDs pero también pueden conectarse a relés u otros dispositivos de salida.

Las entradas digitales suelen ser Botones, Pulsadores o detectores digitales.

A continuación, vamos a comenzar con las prácticas de este capítulo.

6.1. Salida Digital: Blink (intermitente)

Objetivo

Manejo de una salida digital

Funcionalidad

Se trata de crear en la salida pin GP1 una salida digital a la que conectaremos un LED y este se encenderá y apagara con una cadencia temporal de **T.encendido=0,5** s. **T.apagado=0,5 s**.

Instrucciones más importantes que vamos a usar.

import	Importa una función de una librería
Pin(nº, Pin.OUT)	Configura el nº de pin indicado como salida
led.value(Boolean)	Da un valor booleano al objeto led
time.sleep(numero)	Retarda un tiempo indicado en s

Programa

El programa se escribe a continuación de acuerdo a los siguientes pasos:

1. De la librería "**machine**" importamos la función "**Pin**" que nos permitirá definir nuestro pin de trabajo.
2. Declaramos el objeto "**led**" como salida
3. Creamos el "**bucle principal de ejecución**" dentro del cual se colocan las instrucciones de encendido y apagado del LED con la orden **led.value(***booleano***)** y ponemos entre ambas acciones los retardo correspondientes "**time.sleep**(*numero*)"

```
from machine import Pin
import time

led = Pin(1, Pin.OUT)   # creamos el objeto LED en el pin PG1 como salida

while True:
led.value(1)            # Pone LED en on
time.sleep(0.5)          # Espera 0.5s
led.value(0)            # Pone el LED en off
time.sleep(0.5)         # Espera 0.5s
```

Podríamos realizar la aplicación de otra forma, usando la instrucción **led.toggle()** con la librería **"utime"**

```
from machine import Pin
import utime

# LED BLINKING

led = Pin(1, Pin.OUT)   # Configra GP10 como salida
led.low()

while True:
    led.toggle()     # conmuta LED
    utime.sleep(0.1) # Espera 100ms
```

Para esta primera practica vamos a realizar paso a paso todas las operaciones con el fin de que aprendas los pasos a seguir en las restantes prácticas:

Vamos a utilizar los siguientes componentes

- La placa de MAKER PI PICO
- Un dispositivo GROVE LED
- Un cable USB

Pasos a seguir

Vamos a suponer que ya hemos instalado el software **Thonny** y también henos configurado nuestro Raspberry Pi PICO para trabajar con **MicroPython** procediendo como se ha explicado anteriormente en el punto 4.1

1. **Conectamos** la tarjeta **Raspberry Pi Pico** en el puerto **USB** de la Computadora y abrimos la aplicación **Thonny**.

2. Empezamos a escribir nuestro programa en el área de edición de **Thonny** **Importar las bibliotecas necesarias**. Nuestro código está escrito en el gran espacio en blanco sobre el REPL y comenzamos importando dos bibliotecas **MicroPython**. La primera es la clase Pin de la biblioteca Machine, la segunda es **utime**, utilizada para controlar el ritmo de nuestro código.

```
#Blink Básico
from machine import Pin   #Importamos librerías
import time
```

3. **Crear un objeto, "led"** que se utiliza para crear un enlace entre el pin GPIO físico y nuestro código. En este caso, establecerá GPIO 1 (que se asigna al pin físico 2 en la placa) como un pin de salida, donde la corriente fluirá desde la

Raspberry Pi Pico GPIO 1 al LED. Luego usamos el objeto para asegurar que el pin GP1, al iniciarse el programa este apagado

```
led = Pin(1, Pin.OUT)  # creamos el objeto LED en PG1
led.low()        #Al arrancar el LED estará apagado
```

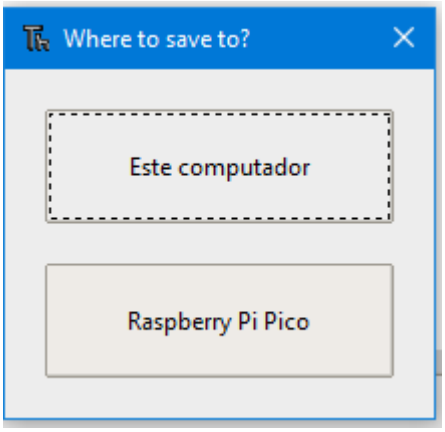
4. Dentro de un bucle true, un bucle sin fin, **activamos y apagamos el LED**

```
while True:
  led.toggle()
  print("Toggle")
  utime.sleep(1)
```

5. **Haga clic en Guardar y elija guardar el código** en el dispositivo **MicroPython** (Raspberry Pi Pico). **Asigne un nombre al archivo blink.py** y **haga clic en Aceptar** para guardarlo. El código debería tener este aspecto.

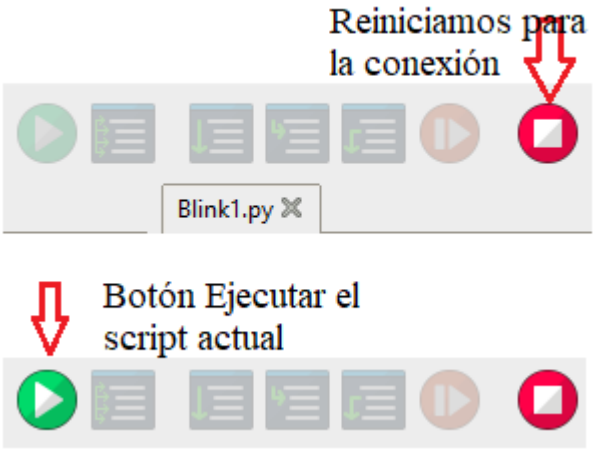
```
#Blink Basico
from machine import Pin     #Importamos librerías
import time

led = Pin(1, Pin.OUT)   # creamos el objeto LED en PG1
led.low()        #Al arrancar el LED estará apagado

while True:
   led.value(1)    # Pone LED en on
   time.sleep(0.5)       # Espera 0.5s
   led.value(0)    # Pone el LED en off
   time.sleep(0.5) # Espera 0.5s
```

6. Ahora guardamos nuestro programa en el PC. Para ello seleccionamos del menú la opción **Archivo->Guardar Como** y cuando nos aparezca la pantalla de la imagen seleccionamos "**Este Computador**" y guardamos en la carpeta de trabajo que hayamos consignado para guardar nuestros trabajos.



Le ponemos de nombre, por ejemplo, Blink1.py

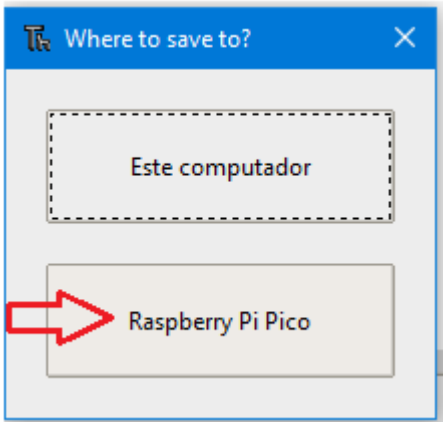
7. Lo que procede ahora es probar nuestro primer proyecto. Para ello lo que haremos es establecer la conexión con nuestra tarjeta y ejecutar pulsando el botón "Retener/Reiniciar" y si esta bien conectada la tarjeta en el puerto USB se activara el botón de "Ejecutar script actual"

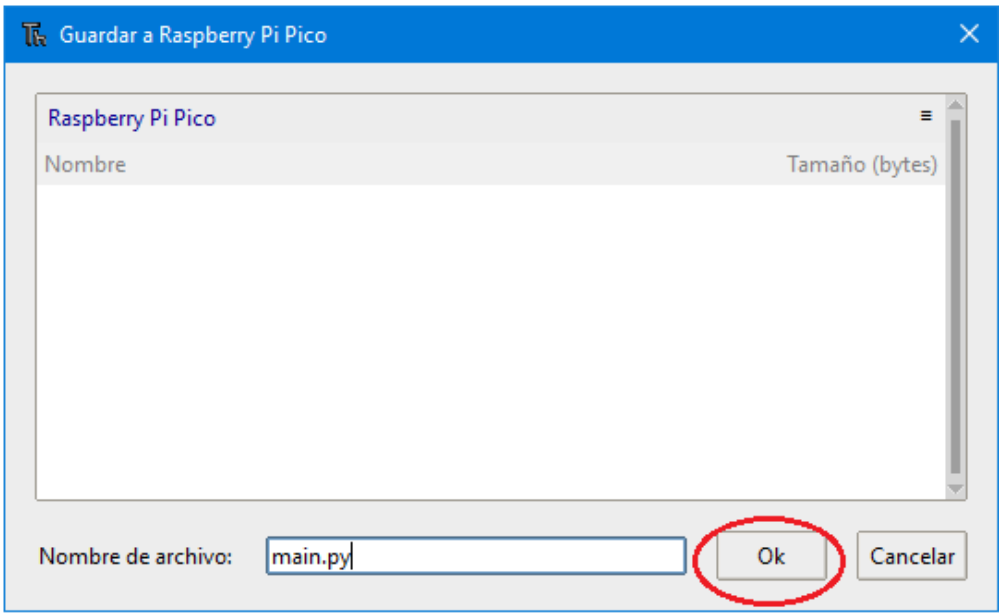


Si todo esta correcto pulsamos en el botón "Ejecutar Script actual" y nuestro programa se ejecutará en modo online a través de **Thonny** y el puerto USB (pero no se habrá cargado en la memoria de Raspberry Pi Pico)

8. Cuando creamos un programa si queremos que se ejecute en la tarjeta de manera autónoma (sin el software **Thonny**) debemos guardarlo en Raspberry Pi Pico. Para esta operación no olvidemos algo muy **IMPORTANTE**: el fichero que guardemos debe tener el nombre "**main.py**" porque en el arranque el sistema ira a leer un fichero llamado así para ejecutarlo, si lo grabamos con otro nombre no funcionará de modo autónomo.

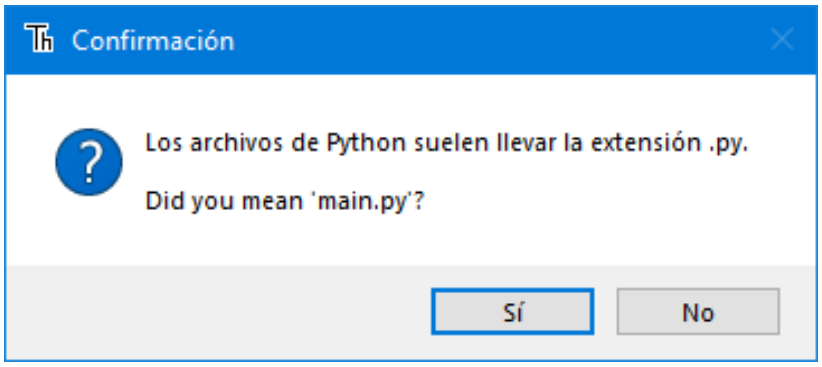
9. Detendremos la ejecución del programa si es que está funcionando, y volveremos a la opción del menú **Archivo->Guardar Como** y cuando nos aparezca la pantalla de la imagen seleccionamos "**Raspberry Pi Pico**" y guardamos en la carpeta de trabajo que hayamos consignado para guardar nuestros trabajos.



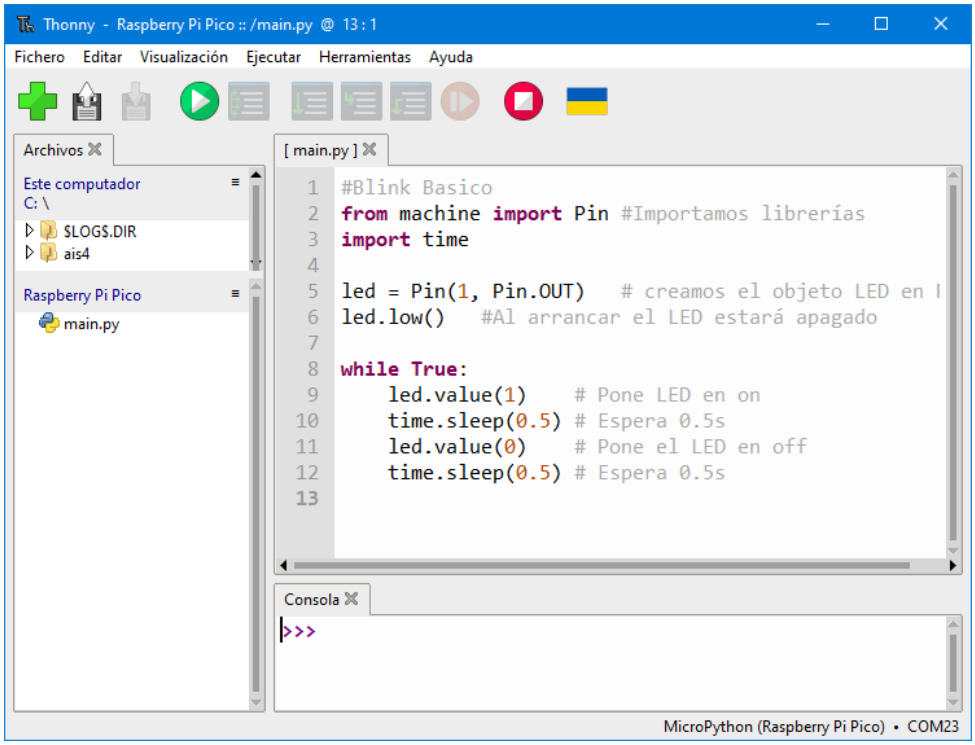


Aparecerá la siguiente ventana que muestra los ficheros que hay, si es que los hay, en la tarjeta y nos invita a poner el nombre de nuestro fichero: Ponemos **main** y damos "**OK**". A continuación, veremos que nuestro fichero aparece con el nombre "**main.py**" en la carpeta de archivos que hay en el microcontrolador Raspberry Pi Pico

Si se nos olvidó la extensión "**main,py**" el programa nos lo indica y nos pide permiso para que el mismo ponga la extensión. Seleccionamos "**Si**" y el fichero se guarda.

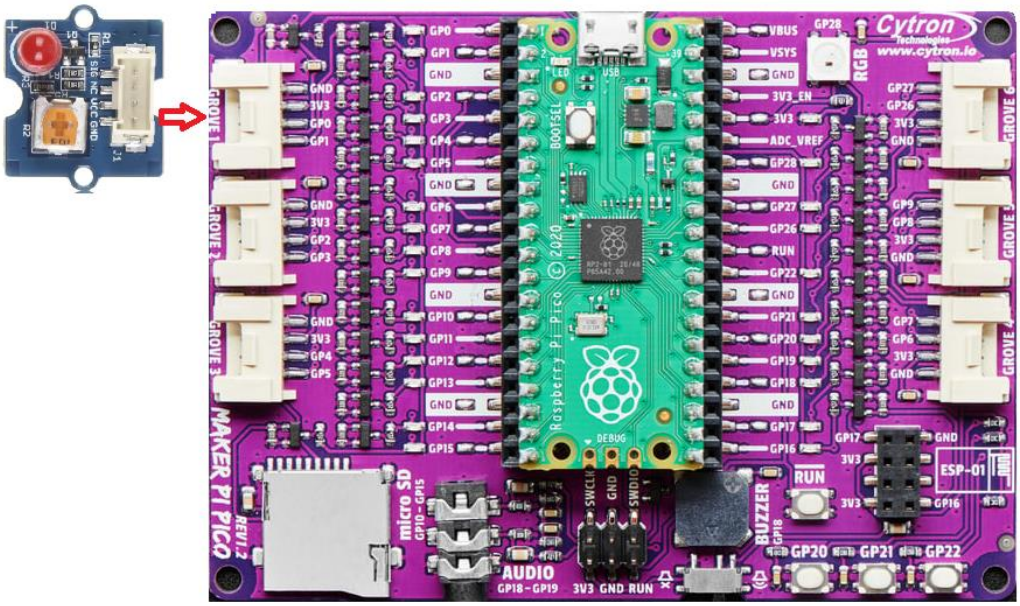


Este es el aspecto del programa



10. En este caso si desconectamos la comunicación entre **Thonny** y la tarjeta y alimentamos la tarjeta simplemente con el mismo cable USB el programa se ejecutará de manera autónoma

Montaje de la practica



Análisis y propuesta de actividades

1. Realizar un Blink con la instrucción **led.toggle** de la librería **utime**

`utime.sleep`(*secons*)

Detiene la ejecución durante el número de segundos dado. Algunas tarjetas pueden aceptar *segundos* como Número de punto flotante para esperar durante una fracción de segundos. Tenga en cuenta que Otros tarjetas pueden no aceptar un argumento de punto flotante, por compatibilidad con utilizan las funciones **sleep_ms()** y **sleep_us().**

```
from machine import Pin

import utime

# LED BLINKING

led = Pin(1, Pin.OUT)   # Configura GP10 como salida

while True:
    led.toggle()          # conmuta LED
    utime.sleep(0.1)      # Espera 100ms
```

2. Realizar un Blink con la instrucción **led.value(not led.value())**

```
#Blink Basico Otro modo
from machine import Pin     #Importamos librerías
from time import sleep

led = Pin(1, Pin.OUT)   # creamos el objeto LED en PG1
led.low()
while True:
    led.value(not led.value())
    sleep(0.5)
```

3. Realizar un Blink en el que los tiempos de estado ON y OFF esten grabados en un fichero txt y el programa debe abrirlo y capturar de el los valores.

```
# Destella un LED en un ciclo de tiempo de encendido y
# apagado dado, donde los dos tiempos se toman de un archivo txt
# guardado en la tarjeta Raspberry Pi Picoen el que se escribe
# en una linea los dos tiempos con numeros
# enteros "  200 400 ".
from machine import Pin
import time
from time import sleep

led = Pin(1, Pin.OUT) #LED Salida GP1
s = open('delay.txt','r') #Abrimos fichero delay.txt
line = s.readline() #Leemos la línea de los datos
s.close() #Cerramos el fichero

delay=line.split()
on=int(delay[0]) #designamos el tiempo activo
off=int(delay[1])#Designamos el tiempo apagado
print('Tiempo ON:',on,'mseg.') #Mostramos tiempos
print('Tiempo OFF:',off,'mseg.')

#Bucle Blink
while True:
    led.value(1)
    time.sleep_ms(on)
    led.value(0)
    time.sleep_ms(off)
```

6.2 Activación secuencial de Salidas

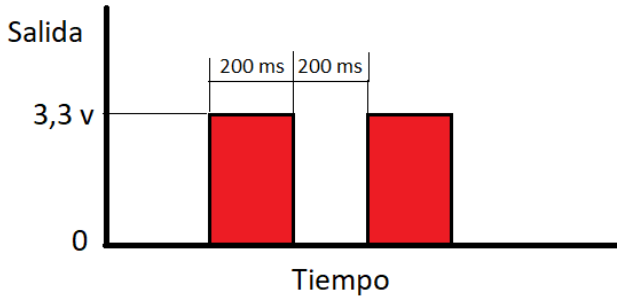
Objetivo

Con esta práctica vamos a trabajar con varias salidas digitales de Raspberry Pi Pico que se activarán de modo secuencial.

Funcionalidad

Usaremos los pines GP0 a GP7 (8 salidas digitales) en las que colocaremos LEDs

Se trata de encender y apagar cada LED de manera secuencial partiendo del pin GP0 hasta llegar al pin GP7. Con los tiempos de encendido y apagado que se muestran en la figura siguiente.



Programa

Para empezar como siempre cargamos las librerías a usar:

```
from machine import Pin
import utime
```

El programa que ejecutara esta secuencia básicamente incluye un bucle de repetición continua **while** dentro del cual incluiremos una instrucción de tipo **for** que lo que hará será realizar el contaje desde **i=o** hasta **i=7** (8 valores) que serán los que se asocian a cada una de las instrucciones de activación y desactivación de cada salida.

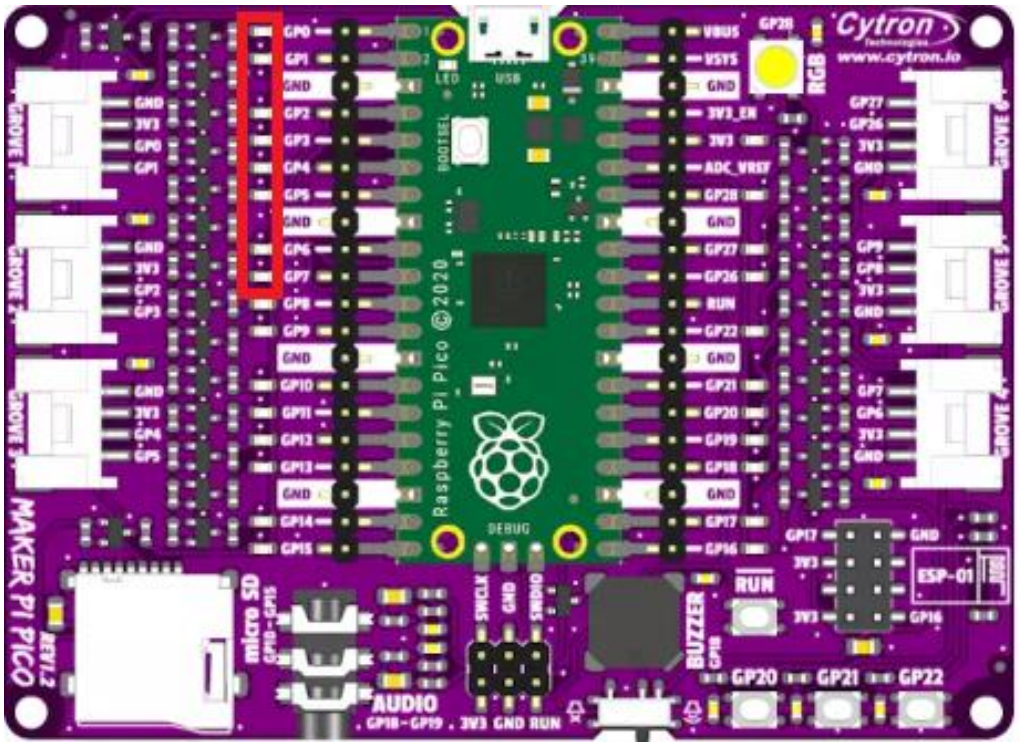
Los tiempos de espera asociados a cada estado de la salida se definen con la función **utime.sleep(n)** donde **n** representa el numero de segundos de espera que en nuestro caso será de 0,2 (200 ms.)

```
###  Secuencia de encendido del pines GP0 a GP7

from machine import Pin
import utime

# Ejecución secuencial

while True:
    for i in range(7):
        Pin(i).value(1)     # pone en on el LED
        utime.sleep(0.2)  # espera 200 ms
        Pin(i).value(0)     # pone en off el LED
        utime.sleep(0.2)  # espera 200 ms
```

Montaje de la practica

Para realizar el montaje deberíamos usar una tarjeta protoboard y cablear las salidas a LEDs.

Para probar nuestro montaje bastara con observar el estado de los lEDs de la tarjeta **Cytron**



Análisis y propuesta de actividades

Una vez creado el programa podemos experimentar con el modificando los tiempos de encendido y apagado de los LEDs así como el numero de LEDs que participan en la secuencia

1. Como actividad te propongo que realices el siguiente programa en el que los pines implicados en la secuencia se integran en una lista con los números de pin de cada estado de la secuencia.

   Se definirá una función a la que podemos bautizar con el nombre **muestraLeds()** que lo que hará será leer de manera secuencial el contenido de la lista de números de pines activando y/o desactivando su estado de encendido o apagado.

   Haremos que la secuencia se realice en sentido ascendente en la lectura de la lista de pines y en sentido descendente.

   Este sería el programa.

```
#Juego de luces en los pines GP0 a GP7
from machine import Pin
import time

pins = [0, 1, 2, 3, 4, 5, 6, 7]
def muestraLeds():   #Definimos la función muestraLeds()
    #Recorremos la lista de pins en sentido ascendente
    for pin in pins:
        print(pin)
        led = Pin(pin, Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
        time.sleep_ms(100)
    #Recorremos la lista de pines en sentido inverso
    for pin in reversed(pins):
        print(pin)
        led = Pin(pin, Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
        time.sleep_ms(100)

while True:
    muestraLeds()
```

   Las pruebas, como en el caso anterior se pueden hacer observando los leds de la placa **Cytron**.

   Puedes probar con estas listas:

```
pins = [0, 1, 2, 3, 7, 6, 5, 4]
pins = [0, 1, 6, 7, 4, 5, 2, 3]
```

6.3 Monoestable con monitorización de estado

Objetivo

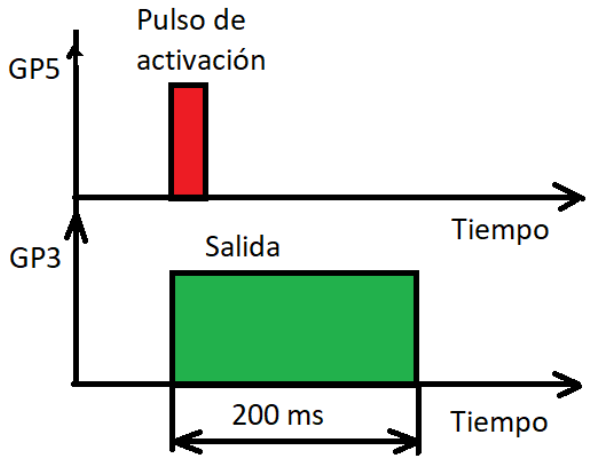
Vamos a crear un programa en el que se active una salida digital un tiempo determinado cunado se pulsa un botón que actúa de "disparo" o activación del "Monoestable"

Funcionalidad

Este es un circuito muy usado en los sistemas electrónicos de control. Se le llama también "Temporizador". Permite, ante una orden de activación, realizar la activación durante un tiempo de la salida. En la figura se muestra un diagrama de tiempo con su funcionamiento.

Vamos a usar dos pines:

- GP5 En el conectamos un botón
- GP3 en el que colocaremos un LED



Funcionamiento:

Si GP5=1 GP3= 1 durante un tiempo (200 ms)

Programa

Nuestro programa se construirá de acuerdo a los siguientes pasos

1. Incluimos las librerías

```
from machine  import Pin
import time
```

2. Definimos los pines GP5 como entrada digital y GP3 como salida digital

```
button = Pin(5,Pin.IN)
led = Pin(3, Pin.OUT) # GP1 LED SALIDA
```

3. Fijamos a “0” (apagada) la salida **GP5** en el momento de arrancar el programa.

```
led.value(0)
```

4. Fijamos el bucle de ejecución con la instrucción **While True** (bucle de ejecución permanente). Dentro de este bucle colocaremos un condicional “**si..”** que testeara se el Botón colocado en **GP5** ha sido pulsado.

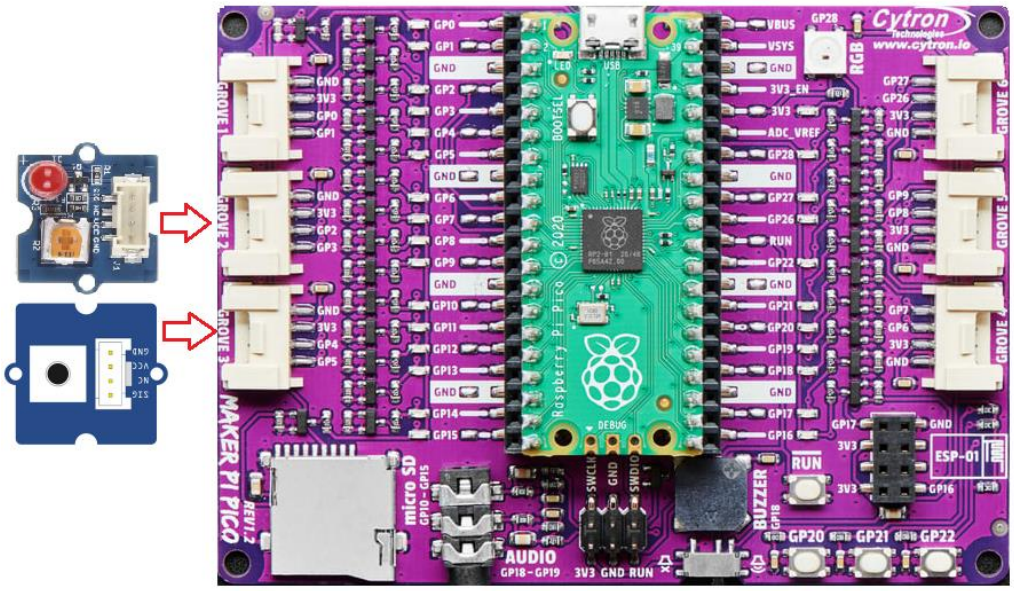
5. Si se pulsa el Botón se debe activar la salida GP3 y a la vez imprimimos en la consola de **Thonny** un mensaje “LED Activado” que nos indica que se ha ejecutado la orden de activación.

6. Lo siguiente será retardar la ejecución del programa un tiempo, el tiempo de temporización que fijamos en 2000 ms, para después imprimir el en la consola el texto “ Tiempo transcurrido”

7. De no cumplirse la condición de Botón pulsado la salida permanecerá en estado apagado “0”

```
### Monoestable con indicación de estados
from machine  import Pin
import time

# GP5 Botón ENTRADA con PULL_UP
button = Pin(5,Pin.IN)
led = Pin(3, Pin.OUT) # GP1 LED SALIDA
led.value(0)

while True:
    if button.value() == 1: # Si el Botón se presiona
        led.value(1)   # Se activa el LED
        print('LED Activado')
        time.sleep_ms(2000)   #Tiempo en ms
        led.value(0)
        print('LED Desactivado')
        print('Tiempo transcurrido', 2000,'ms')
    else:
        led.value(0)
```

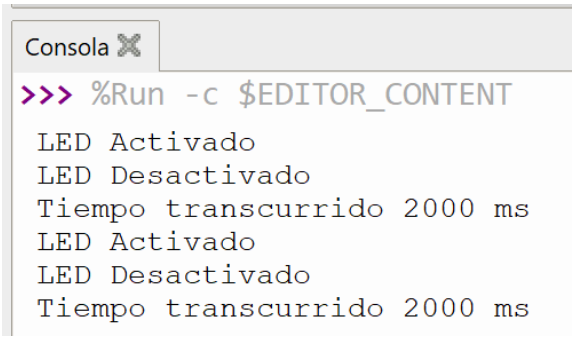
Montaje de la practica



Análisis y propuesta de actividades

En el análisis del funcionamiento del programa tenemos que fijarnos en la consola para ver los estados de trabajo de nuestro programa.

En la figura vemos el aspecto de la consola de **Thonny**



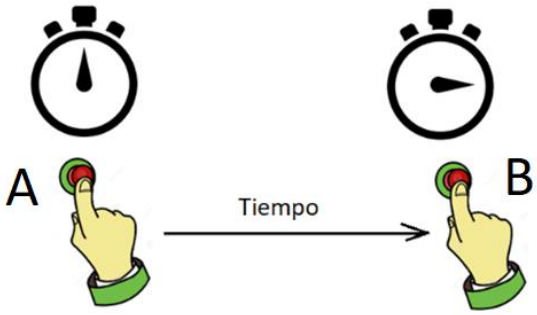
6.4. Medir tiempo entre dos eventos

Objetivo

Son numerosas las ocasiones en las que se hace necesario medir tiempos con un microcontrolador. En esta práctica vas a aprender una sencilla forma de medir el tiempo trascurrido entre dos eventos

Funcionalidad

Estableceremos dos eventos, uno de inicio de la cuanta del tiempo mediante un botón A conectado en el pin GP20 y otro que marcara el final de la cuanta del tiempo que se llevara a cabo mediante un pulsador B conectado en el pin GP21



Estos botones son los que vienen en la propia placa de **Cytron** que estamos usando.

Usaremos la consola de **Thonny** para mostrar el tiempo transcurrido.

Programa

1. Nuestro programa lo primero que debe hacer es importar las librerías

```
from machine  import Pin
import time
from time import sleep
```

2. Definimos los pines **Botón A** y **Botón B** asociados a los pines **GP20** y **GP21**
3. Mediante dos condicionales estableceremos los eventos de inicio y fin de la medida de tiempo testeamos el estado del botón. Téngase en cuenta que en reposo estos botones envían un "1" por lo tanto testeamos el valor del botón como "0".
4. Usamos dos variables:

   start que recoge el valor del tiempo en el inicio **start = time.time()**
   end que recoge el valor del tiempo en el final **new=time.time()**

5. La diferencia de ambos será el tiempo transcurrido **end-start**

6. En cada condicional de detección de pulsación de botón lo que hacemos es poner un retardo de tiempo para evitar los rebotes a la hora de pulsar el botón **time.sleep_ms(300)**

7. Para monitorizar los eventos de pulsación de botón imprimiremos

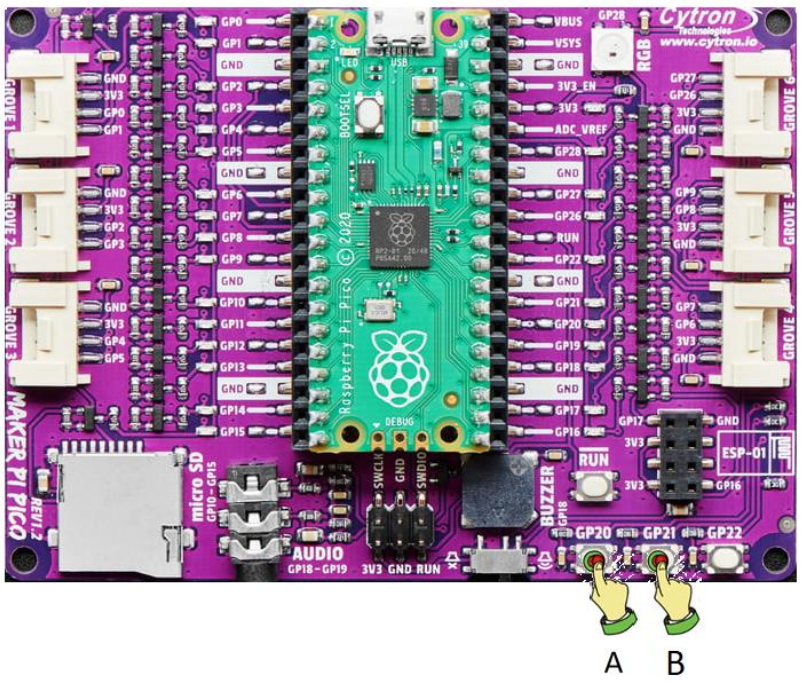
   print('Comienza tiempo=0 seg') al inicio y **print("Tiempo transcurrido", end-start, "seg.")** al final

   Este sería el programa completo.

```
# Medir el tiempo transcurrido entre dos eventos
from machine  import Pin
import time
from time import sleep
buttonA = Pin(20,Pin.IN) #Inicia la cuenta del tiempo
buttonB = Pin(21,Pin.IN) #Detiene el contador de tiempo

while True:

    if buttonA.value() == 0: # Si el Botón se presiona
        start = time.time() #Inicia tiempo
        print('Comienza tiempo=0 seg')
        time.sleep_ms(300)

    if buttonB.value() == 0: # Si el Botón se presiona
        end = time.time() #Detiene tiempo
        print("Tiempo transcurrido", end-start, "seg.")
        time.sleep_ms(300)
```

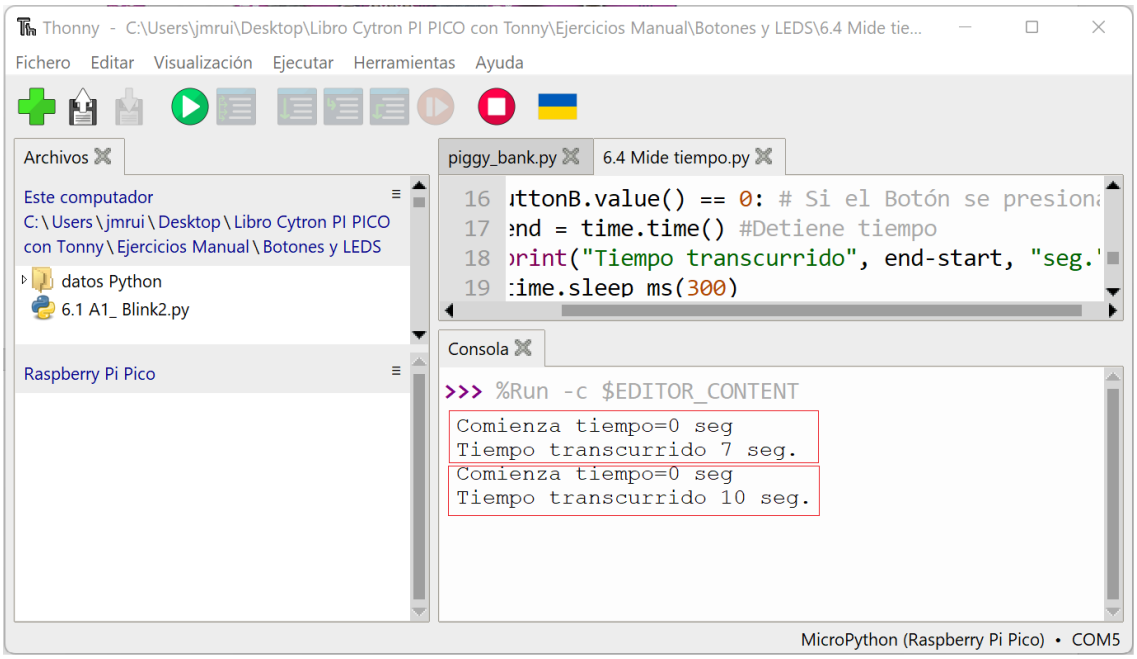
Montaje de la practica

Este sería el montaje. Como vemos solo nos basta con la tarjeta **Cytron**



Análisis y propuesta de actividades

Cuando ejecutamos el programa observamos el área de Consola



6.5. Contador Binario

Objetivo

Crear un contador binario y probarlo usando los LEDs indicadores de la tarjeta **Cytron**

Funcionalidad

Este contador binario de 4 bits implementado con **MicroPython** permite realizar un conteo desde 0 hasta 15.

Al ejecutar el programa, comenzará un conteo desde 0 hasta 15, con un tiempo de espera de 1 segundo. Los leds mostraran el valor binario correspondiente a cada número.

Usaremos la función de conversión:

```
b = "{0:04b}".format(a)
```

```
>>> b = "{0:04b}".format(26)
>>> print(b)
11010
```

```
>>> b = "{0:08b}".format(26)
>>> print(b)
00011010
```

Cuando el contador alcance su valor máximo (15), comenzará nuevamente desde su valor mínimo (0), el ciclo se repetirá indefinidamente.

Si queremos modificar el número de bits de nuestro contador bastara que en la definición de la función contador pongamos las instrucciones de acuerdo con lo que se muestra a continuación:

#4 Bits	#6 Bits	#6 Bits
def contador(a):	**def contador**(a):	**def contador**(a):
b = "{0:04b}".format(a)	b = "{0:06b}".format(a)	b = "{0:08b}".format(a)
#Imprime el valor en binario y decimal **print**('Binario',b,'Decimal',i)	#Imprime el valor en binario y decimal **print**('Binario',b,'Decimal',i)	#Imprime el valor en binario y decimal **print**('Binario',b,'Decimal',i)
pin_a.value(int(b[0]))	pin_a.value(int(b[0]))	pin_a.value(int(b[0]))
pin_b.value(int(b[1]))	pin_b.value(int(b[1]))	pin_b.value(int(b[1]))
pin_c.value(int(b[2]))	pin_c.value(int(b[2]))	pin_c.value(int(b[2]))

pin_d.value(int(b[3]))	pin_d.value(int(b[3])) pin_c.value(int(b[4])) pin_d.value(int(b[5]))	pin_d.value(int(b[3])) pin_c.value(int(b[4])) pin_d.value(int(b[5])) pin_c.value(int(b[6])) pin_d.value(int(b[7]))

Programa

Nuestro programa se construirá de acuerdo a los siguientes postulados de diseño:

1. Se cargan las librerías que usaremos.

```
from machine import Pin
from time import sleep
```

2. Definimos los pines por los que sacaremos los códigos binarios.

```
pin_a = Pin(0, Pin.OUT)
pin_b = Pin(1, Pin.OUT)
pin_c = Pin(2, Pin.OUT)
pin_d = Pin(3, Pin.OUT)
```

3. Definimos la función contador designando en ella el valor de cada uno d ellos bits que se asocian a los pines. Teniendo en cuenta que el valor "**b**" se considera una lista de la que extraemos cada uno de los bits que se incluyen en cada posición. En la función también colocamos la orden de impresión del valor binario con el fin de poder monitorizar en la "**consola**" de **Thonny** los códigos

```
def contador(a):
    b = "{0:04b}".format(a)
    #Imprime el valor en binario y decimal
    print('Binario',b,'Decimal',i)
    pin_a.value(int(b[0]))
    pin_b.value(int(b[1]))
    pin_c.value(int(b[2]))
    pin_d.value(int(b[3]))
```

4. En el bucle **while True** lo que haremos es definir una función de tipo **for** en el que el valor del puntero contador "**i**" varía entre 0 y 15 (se trata de recorrer 16 códigos que son los que podemos realizar con 4 bits binarios). Invocamos la función contador y retardamos un segundo

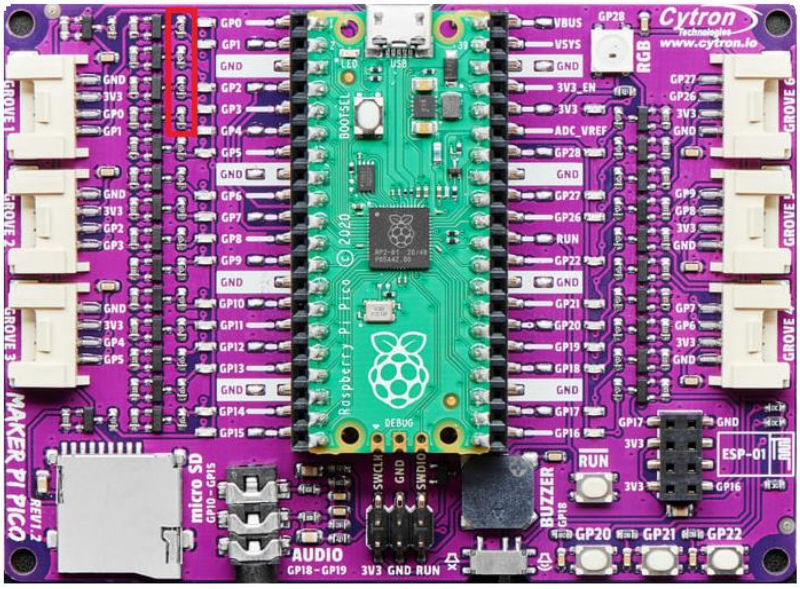
```
for i in range(16):  #Indicamos el maximo numero de cuenta
```

```
        #Envia a la funcion el valor de conteo
        contador(i)
        #Retardo 1s
        sleep(1)  #espera 1 seg.
```

Seguidamente mostramos el programa

```
#Contador Binario en las salidas GP0 a GP3

from machine import Pin
from time import sleep
#Setup Pines de Salida
pin_a = Pin(0, Pin.OUT)
pin_b = Pin(1, Pin.OUT)
pin_c = Pin(2, Pin.OUT)
pin_d = Pin(3, Pin.OUT)
#Funcion contador para activar cada salida
def contador(a):
    b = "{0:04b}".format(a)
    #Imprime el valor en binario y decimal
    print('Binario',b,'Decimal',i)
    pin_a.value(int(b[0]))
    pin_b.value(int(b[1]))
    pin_c.value(int(b[2]))
    pin_d.value(int(b[3]))
#Bucle de repeticion infinita
while True:
    #Bucle que cuenta de 0 a 15
    #16 es el valor maximo no incluido (4 bits)
    for i in range(16):  #Indicamos el maximo numero de cuenta
        #Envia a la funcion el valor de conteo
        contador(i)
        #Retardo 1s
        sleep(1)  #espera 1 seg.
```

Montaje de la práctica



El montaje es sencillo, solo necesitamos la tarjeta **Cytron** y observar el ella los LEDs indicadores GP0 a GP3

Análisis y propuesta de actividades

Vemos en la imagen la venta "Consola" en la que van apareciendo los códigos binarios y decimal tal como se van generando cada 1 segundo en la ejecución del programa

```
Consola
>>> %Run -c $EDITOR_CONTENT
 Binario 0000 Decimal 0
 Binario 0001 Decimal 1
 Binario 0010 Decimal 2
 Binario 0011 Decimal 3
 Binario 0100 Decimal 4
 Binario 0101 Decimal 5
 Binario 0110 Decimal 6
 Binario 0111 Decimal 7
 Binario 1000 Decimal 8
 Binario 1001 Decimal 9
```

Proponemos que se modifique el número de bits a 6 y 8 bits de salida y observemos el comportamiento del programa y la visualización en la tarjeta **Cytron**

6.6. Biestable

Objetivo

Con esta practica vamos a construir un biestable gobernado con un solo botón

Funcionalidad

Sabemos que un biestable es también una función “memoria” que mantiene su estado “0” o “1” dependiendo de la actuación sobre su o sus entradas. En nuestro caso usamos una sola entrada.

Entradas		Salida
D	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

En la figura se muestra la Tabla de Verdad de dicho dispositivo.

Designaremos los siguiente pines

- GP5 Botón de activación D
- GP7 LED Salida Q del Biestable

Programa

Nuestro programa se construirá de acuerdo a las siguientes fases:

1. Importación de librerías

```
from machine import Pin
import time
```

2. Definimos los pines E/S que usaremos y ponemos a “0” la salida del Biestable al iniciarse el programa. El pin de entrada se define como PULL_UP para asegurarnos su estado en reposo

```
led = Pin(7, Pin.OUT)
button = Pin(5, Pin.IN, Pin.PULL_UP)
led.value(0)
```

3. A continuación tenemos que definir la función “biestable” que la que nos permitirá la conmutación de la salida Q del dispositivo.

def *(nombre)*():	Definición de una función llamada “nombre”

```
def biestable(): #Función biestable
    if led.value():
        led.value(0) #Pone el LED en on
    else:
        led.value(1)    #Pone el LED en off
```

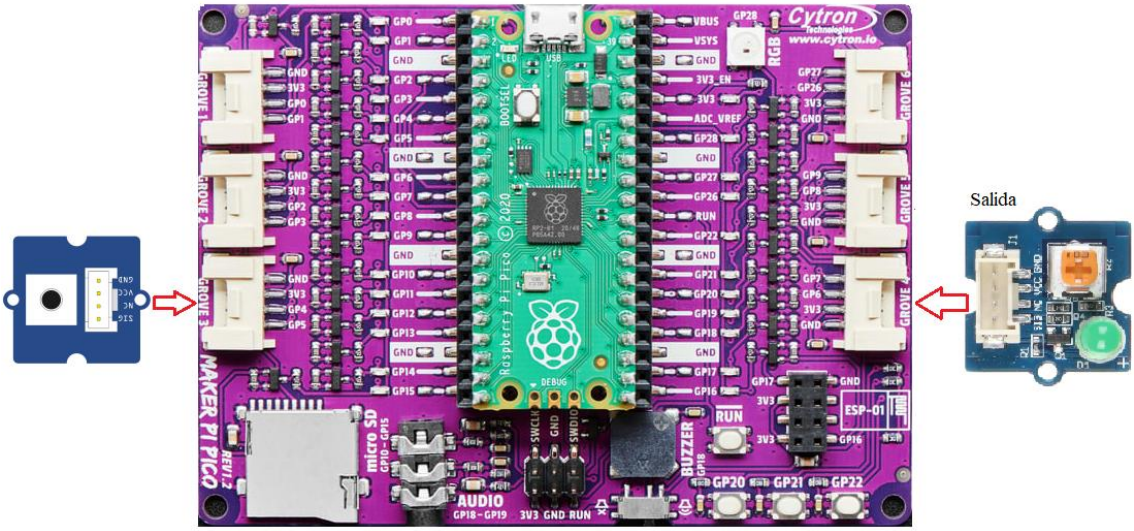
Esta función lo que hace es que testea el valor del estado del led y si es “1” lo pone en “0” y viceversa.

4. Finalmente lo que hacemos es colocar el bucle de ejecución continua dentro del cual se testea el estado del botón D de gobierno del biestable ejecutando la función “biestable” y colocamos finalmente un retardo que evita los rebotes en el botón.

```
while True:
    if button.value(): #Si el botón se pulsa se envía un "0"
        biestable()
        time.sleep_ms(300) #Retardo para evitar rebote en el botón
```

```
#Biestable mediante función

from machine import Pin
import time

led = Pin(7, Pin.OUT)
button = Pin(5, Pin.IN, Pin.PULL_UP)    #botón el pin GP5 definido como entrada
led.value(0)

def biestable(): #Función biestable
    if led.value():
        led.value(0) #Pone el LED en on
    else:
        led.value(1)    #Pone el LED en off

while True:
    if button.value(): #Si el botón se pulsa se envía un "0"
        biestable()
        time.sleep_ms(300) #Retardo para evitar rebote en el botón
```

Montaje de la practica



6.7 Codificador Rotativo

Objetivo

Con este montaje vamos a probar el funcionamiento de un codificador rotatorio muy usado para mecanismos de posicionamiento de ejes giratorios.

Funcionalidad

El "codificador rotativo" que usaremos es el de la figura

Este módulo es ideal para añadir detección de rotación. El codificador te permite añadir control a tu proyecto similar a los mandos de control de volumen de radio en coches, detectando la dirección de rotación, la velocidad y el estado del botón.

En funcionamiento cuando en su estado sin girar las salidas **CLK** y **DT** serán altos, cuando gire el codificador uno de ellos irá bajo seguido por el otro poco después. El primero a señal bajo proporcionará su dirección. El interruptor **SW** se lee bajo cuando se presiona.

El eje giratorio de 6 mm proporciona una gran variedad de pomos compatibles para elegir, lo que le permite obtener el pomo adecuado para su trabajo.

La designación de pines es la siguiente:

- **clk** Pin **GP2**
- **dt** Pin **GP3**
- **SW** Pin **GP4**

Para este programa usaremos dos librerías que nos facilita la firma **KeyStudio** para poder usar su dispositivo "Codificador rotativo". Estas librerías son **rotary_irq_rp2** y **rotary**. Con estas librerías podremos crear un objeto llamado

r = RotaryIRQ(pin_num_clk=2, pin_num_dt=3, min_val=0, reverse=False, range_mode=RotaryIRQ.RANGE_UNBOUNDED)

también debemos definir el objeto: *SW=Pin(4,Pin.IN,Pin.PULL_UP)*

Este objeto tiene un procedimiento que lo que hace es devolver el valor de la posición:

r.value()

Programa

La realización del programa en secuencia es la siguiente:

1. Importar las librerías, incluidas las propias del dispositivo Codificador

```
import time
from rotary_irq_rp2 import RotaryIRQ #Carga librerías
from machine import Pin
```

2. Definiremos el objeto SW digital en el GP4 que sera la señal para cargar o fijar el valor de cuenta

3. Definimos el objeto "r" que representa al dispositivo codificador con los parámetros siguientes: de la función **RotaryIRQ**

```
pin_num_clk=2
pin_num_dt=3
min_val=0
reverse=False
range_mode=RotaryIRQ.RANGE_UNBOUNDED)
```

4. Definimos la variable **val_old = r.value()** (valor leído del objeto r)

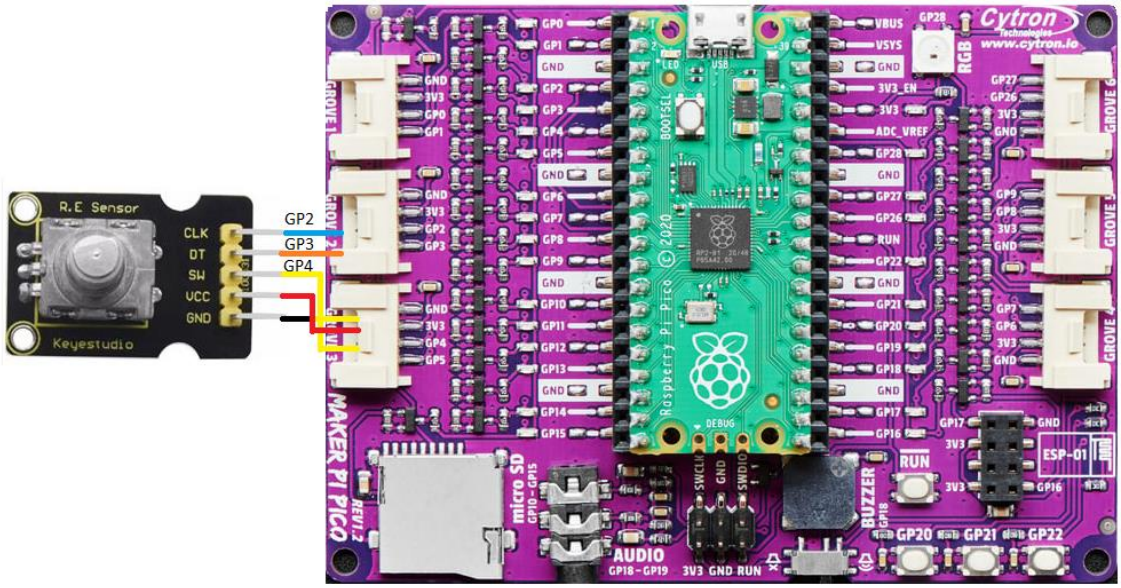
5. Dentro del bucle **while True** pondremos dos condicionales. En el primer condicional evaluamos si se pulsa el botón W del "Codificador" en cuyo caso se fija el valor y el segundo condicional detecta el cambio de "valor" y si efectivamente el valor nuevo es distinto del valor anterior entonces el antiguo valor es igual al nuevo valor

```
while True:
# El valor leido se asocia a la variable val_new
    try:
        val_new = r.value()
        if SW.value()==0 and n==0:
            print("Butón presionado")
            print("Valor seleccionado es : ",val_new)
            n=1
            while SW.value()==0:
                continue
        n=0
        if val_old != val_new:
            val_old = val_new
            print('valor =', val_new)
            time.sleep_ms(50)
    except KeyboardInterrupt:
        break
```

El programa completo será el siguiente.

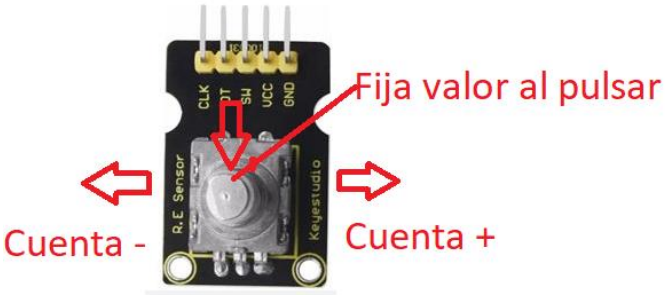
```
'''
 * CODIFICADOR ROTATIVO Keystudio para Raspberry Pi Pico
'''
import time
from rotary_irq_rp2 import RotaryIRQ #Carga librerías
from machine import Pin

SW=Pin(4,Pin.IN,Pin.PULL_UP)  #Define pines SW,CLK y DT
r = RotaryIRQ(pin_num_clk=2,
              pin_num_dt=3,
              min_val=0,
              reverse=False,
              range_mode=RotaryIRQ.RANGE_UNBOUNDED)
val_old = r.value()
while True:
# El valor leido se asocia a la variable val_new
    try:
        val_new = r.value()
        if SW.value()==0 and n==0:
            print("Butón presionado")
            print("Valor seleccionado es : ",val_new)
            n=1
            while SW.value()==0:
                continue
        n=0
        if val_old != val_new:
            val_old = val_new
            print('valor =', val_new)
            time.sleep_ms(50)
    except KeyboardInterrupt:
        break
```

Montaje de la practica



Análisis y propuesta de actividades

Una vez cargado el programa se ejecutará y observemos la consola en la que se van escribiendo los valores de la cuenta. Probar con los giros a derecha e izquierda. Probar que pulsando el botón del dispositivo se asocia el valor al último




```
Consola
>>> %Run -c $EDITOR_CONTENT
 valor = 1
 valor = 2
 valor = 3
 valor = 2
 valor = 1
 valor = 0
 valor = 1
 valor = 2
 valor = 3
 valor = 4
 Butón presionado
 Valor seleccionado es :  4
```

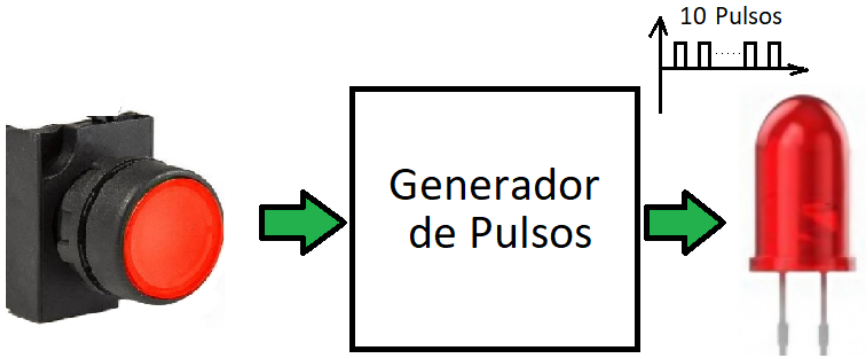
6.8. Generador de pulsos

Objetivo

Nuestro ejemplo consiste en la activación y desactivación un número de veces determinado de una salida. Es decir crear un generador de pulsos.

Funcionalidad

Colocaremos en el pin **GP1** de la tarjeta **Cytron** un LED que se activara y desactivará 10 veces con una cadencia de tiempo de 1 seg.



Programa

1. En primer lugar cargaremos las librerías

```
from machine import Pin
import time
```

2. Seguidamente definimos el objeto led en el pin GP1 y el botón en el pin GP20 (este pin esta asociado en la tarjeta **Cytron** a un botón). Ponemos en estado apagado "0" la salida led.

```
led = Pin(1, Pin.OUT)   # creamos el objeto LED en el pin PG1 como salida
botón=Pin(20, Pin.IN, Pin.PULL_UP)
led.value(0)
```

3. Seguidamente definimos nuestro bucle de ejecución **while True** dentro del cual pondremos un condicional que testea la pulsación del **boton 20** de **GP20,** el botón en reposo envía un "1" por lo que debemos poner la condición en modo negado **if not** botón.value().

4. Si se cumple la condición lo que haremos será ejecutar una instrucción **for** con 10 ciclos de ejecución **for** i **in** range(10)

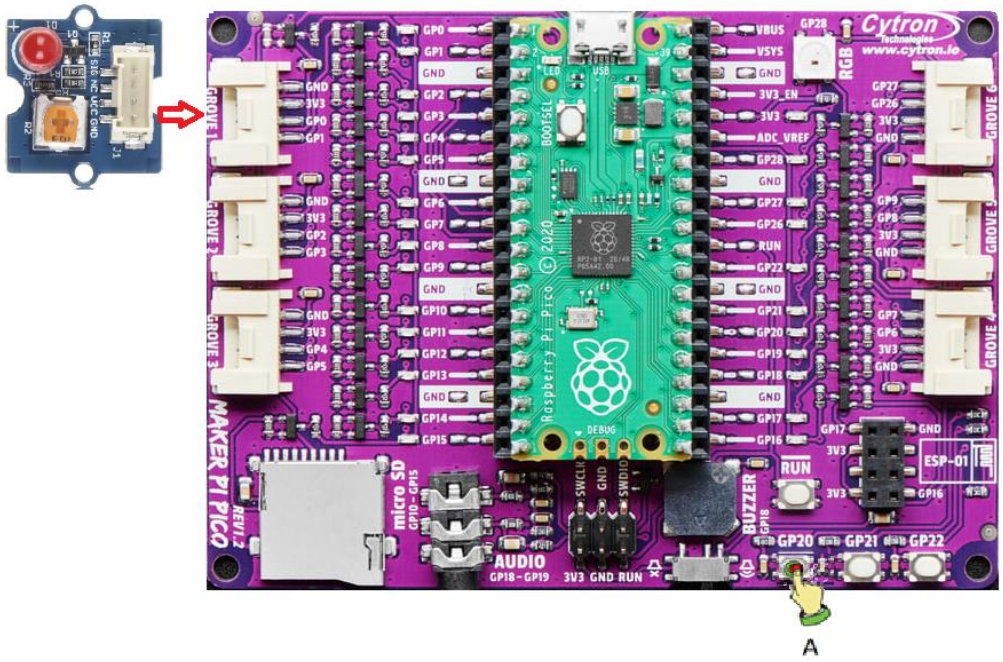
5. En cada ciclo se ejecutan las siguientes instruciones:

```
        print("Pulso numero", 1+i)
        led.value(1)
        utime.sleep(1)
        led.value(0)
        utime.sleep(1)
        print("FIN!")
```

En el cuadro siguiente aparece el programa completo

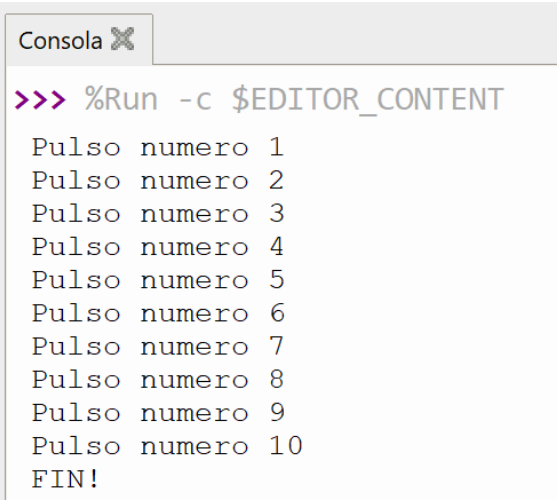
```
#Generador de pulsos
from machine import Pin
import time
led = Pin(1, Pin.OUT)   # creamos el objeto LED en el pin PG1 como salida
botón=Pin(20, Pin.IN, Pin.PULL_UP)
led.value(0)
while True
   if not botón.value():
        for i in range(10):      #Ejecutamos 10 veces
           print("Pulso numero", 1+i)
           led.value(1)
           utime.sleep(1)
           led.value(0)
           utime.sleep(1)
           print("FIN!")
```

Montaje de la practica



Análisis y propuesta de actividades

Cuando iniciamos nuestro programa en la ventana "**consola**" de **Thonny** se visualizan los pulsos generados



Realiza cambios en el programa para que varie el número de pulsos, en este caso 20 pulsos y los tiempos serán *Tencendido=100* ms y *Tapagado=300 ms*

6.9. Semáforo

Objetivo

Con este ejemplo vamos a realizar la simulación de un semáforo básico usando tres salida digitales de la tarjeta **Cytrón**

Funcionalidad

Se usarán tres dispositivos LED de tipo Grove, si es posible de color rojo, amarillo y verde que se activaran de manera secuencial.

En este caso se trata de que cada uno de los LEDs del semáforo conectados a los Pines GP0, GP1 y GP2 se enciendan y apaguen de manera secuencial de acuerdo a la siguiente tabla.

ESTADO	Rojo	Amarillo	Verde	Tiempo
1	ON	OFF	OFF	7 seg.
2	OFF	ON	OFF	5 seg.
3	OFF	OFF	ON	10 seg.
4	OFF	ON	OFF	1 seg.

Usaremos los pines

- GP0 LED Rojo
- GP1 LED Amarillo
- GP2 LED Verde

Programa

La elaboración del programa se llevara a cabo siguiendo los pasos:

1. Importación de las librerías

```
from machine import Pin
import utime
```

2. Definición de los pines en los que conectaremos los LEDs

```
rojo = Pin(0 , Pin.OUT)
amarillo = Pin(1 , Pin.OUT)
verde = Pin(2 , Pin.OUT)
```

3. Puesta a “0” de todas las salidas en el comienzo del programa.

```
rojo.value(0)
amarillo.value(0)
verde.value(0)
```

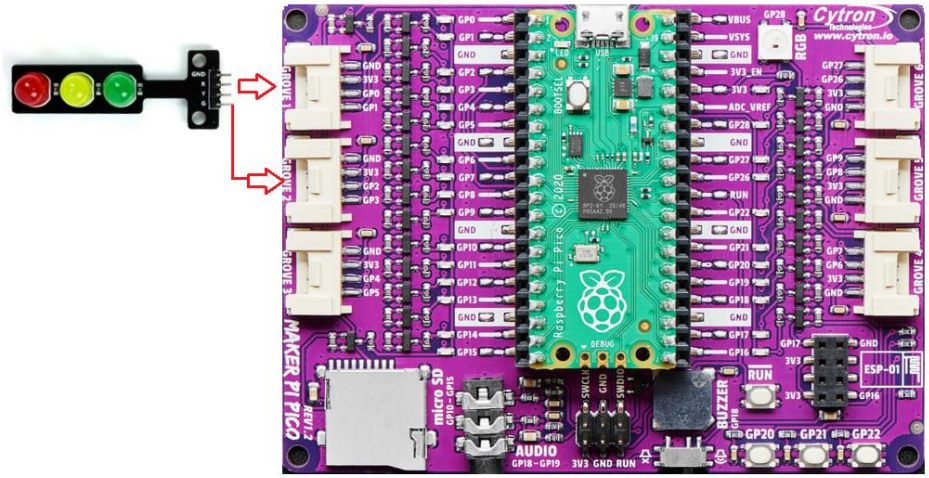
4. Establecimiento del bucle **while True** dentro del cual se colocaran las instrucciones de activación *rojo.value(1)* y desactivación *rojo.value(0)* de los LEDs así como los tiempos de espera *utime.sleep(5)*. El ciclo es el que se ha puesto en la tabla anterior.

Este es el programa completo

```
#Semáforo básico
from machine import Pin
import utime
rojo = Pin(0 , Pin.OUT)
amarillo = Pin(1 , Pin.OUT)
verde = Pin(2 , Pin.OUT)
#Puesta a cero de todas las salida
rojo.value(0)
amarillo.value(0)
verde.value(0)

while True:
    rojo.value(1) #Enciende rojo
    utime.sleep(7)   # Espera 7 seg.
    amarillo.value(1)
    rojo.value(0)
    utime.sleep(5)
    amarillo.value(0)
    verde.value(1)
    utime.sleep(10)
    verde.value(0)
    amarillo.value(1)
    utime.sleep(5)
    amarillo.value(0)
```

Esquema de montaje:

Este sería el montaje de la aplicación.



Análisis y propuesta de actividades

Realizar cambios en los tiempos de encendido de cada LED

6.10. Barra de LEDs MY9221 de Seeed Grove

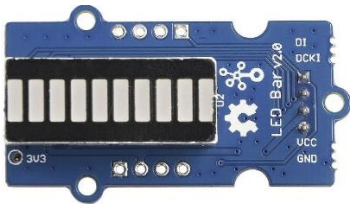
Objetivo

Vamos a probar un componente de la colección **Seeed Grove** que es una **Barra de LEDs MY9221**

Funcionalidad

Este componente es muy útil para monitorizar el valor de una variable. Se trata de la disposición de 10 LEDs en formato de barra que se encenderán en función del valor que le entreguemos al dispositivo.

En la siguiente figura vemos el componente.



Los terminales son: **DI** (datos) **DCKI** (reloj) y alimentación **VCC** y **GND**

Para crear un valor que se use como muestra para enviar al dispositivo usaremos los dos botones que incluye la tarjeta **Cytron GP20** y **GP21**

Usaremos una librería que nos facilita el fabricante para usa con **MicroPython my9221**

Programa

1. En primer lugar cargamos las librerías

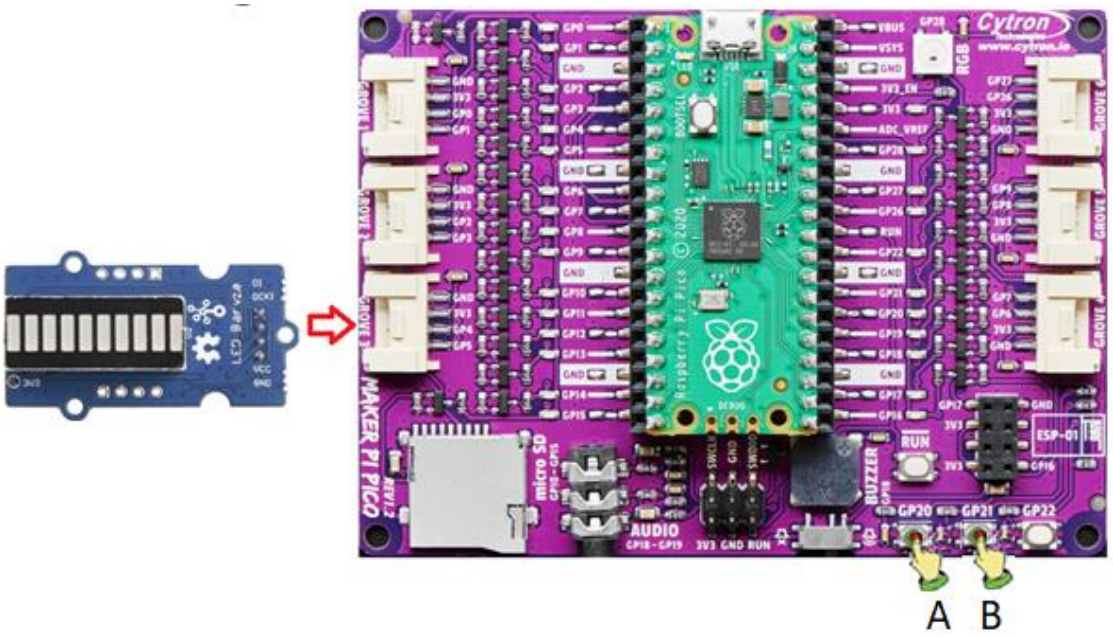
2. Definimos el objeto **ledbar** que conectaremos en los pines **DI=GP5** y **DCKI=GP4** y definimos los botones **Botón A** y **Botón B** en los pines **GP20** y **GP21**

3. Definimos la variable "**num**" y la igualamos a "0" y enviamos este valor al objeto *ledbar* con el procedimiento **ledbar.level(num)**

4. En el bucle **while True** integramos los dos condicionales cuyas condiciones son la pulsación de los botones **A** y **B** que incrementan o decrementan el valor a mostrar en la barra de LEDs. Colocamos un retardo para evitar rebotes en el pulsador e imprimimos el valor de la variable "**num**"

```
#Barra de LEDs deMY9221
from machine import Pin
from my9221 import MY9221
import time
#Barra de LEDs = MY9221(di=pin1, dcki=pin2, reverse=False)
ledbar = MY9221(Pin(5), Pin(4))
botonA = Pin(20,Pin.IN)
botonB = Pin(21,Pin.IN)

num = 0 #Ponemos a "0" la variable "num"
ledbar.level(num) # Incluimos "num" en el objeto ledbar.level

while True:
    if botonA.value() == 0: #Incrementamos el valor de num
        num = num + 1
        ledbar.level(num) #Mostramos num en la barra
        time.sleep_ms(200)
        print(num) # Imprimimos el valor de num
    if botonB.value() == 0: #Decrementamos el valor de num
        num = num - 1
        ledbar.level(num)
        time.sleep_ms(200)
        print(num) # Imprimimos el valor de num
```

Montaje de la practica



Análisis y propuesta de actividades

Una vez que se haya montado el programa se someterá a las pruebas correspondientes.

1. Se sugiere montar el programa siguiente en el que se muestra en el dispositivo una lista de valores dentro de una secuencia en bucle.

```
# Ciclo de activación de LEDs
from machine import Pin
from my9221 import MY9221
import time

ledbar = MY9221(Pin(5), Pin(4))

buf = [0,0,1,3,7,15,31,63,127,255]

while True:
    buf.insert(0,buf.pop())
    ledbar.bytes(buf)
    time.sleep_ms(30)
```

2. En esta segunda propuesta probaras como es posible mostrar en el dispositivo de LEDs el valor de un canal analógico entrada en el **GP27** en el que se cablera un potenciómetro. Mapearemos la señal generada por el potenciómetro (0 a 65535) a un rango de 0 a 10 para poder visualizar el rango completo de la variable.

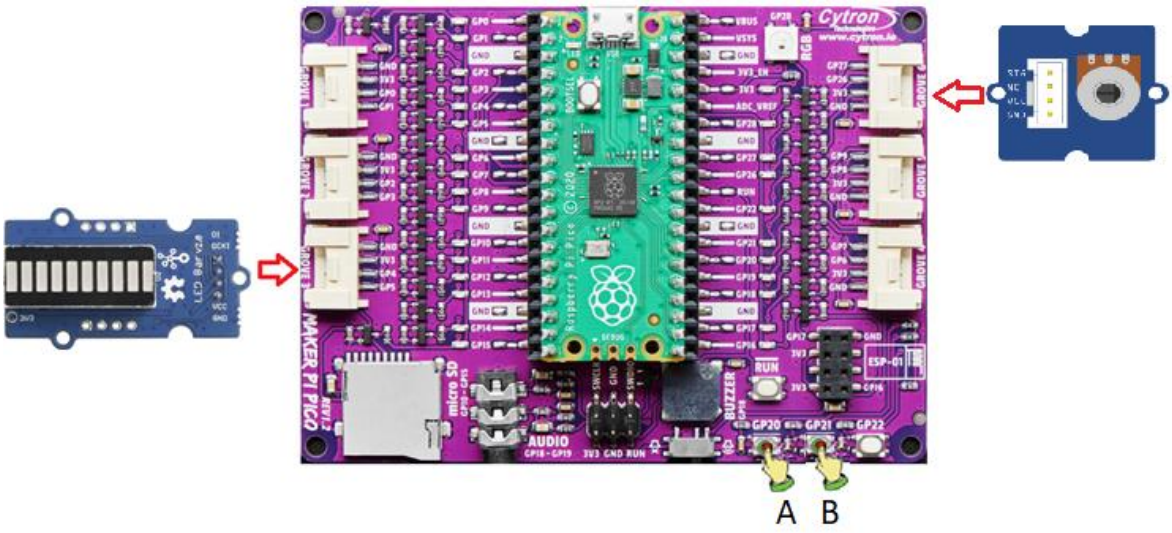
```
#Visualizacion de valr Analogico GP27
from machine import ADC, Pin
from my9221 import MY9221
import time
#Barra de LEDs = MY9221(di=pin1, dcki=pin2, reverse=False)
ledbar = MY9221(Pin(5), Pin(4))
adc = ADC(27)

voltaje = 0 #Ponemos a "0" la variable "voltaje"
ledbar.level(voltaje) # Incluimos "voltaje" en el objeto ledbar.level

def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)

try:
   while True:
       adcValue = adc.read_u16()
       voltaje = map(adcValue, 0, 65535, 0, 10)
       print("ADC Valor:", adcValue, "Voltaje:", int(voltaje), "V")
       ledbar.level(voltaje)
       time.sleep(0.1)
except:
```

En la siguiente imagen vemos el aspecto de la consola mostrando los valores leídos así como la representación grafica del valor.

```
from machine import ADC, Pin
from my9221 import MY9221
import time
#Barra de LEDs = MY9221(di=pin1, dcki=pin2, reverse=False)
ledbar = MY9221(Pin(5), Pin(4))
adc = ADC(27)

voltaje = 0 #Ponemos a "0" la variable "voltaje"
ledbar.level(voltaje) # Incluimos "voltaje" en el objeto ledbar.level

def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)

try:
    while True:
        adcValue = adc.read_u16()
        voltaje = map(adcValue, 0, 65535, 0, 10)
        print("ADC Valor:", adcValue, "Voltaje:", int(voltaje), "V")
        ledbar.level(voltaje)
        time.sleep(0.1)
except:
```

```
ADC Valor: 44218 Voltaje: 6 V
ADC Valor: 46011 Voltaje: 7 V
ADC Valor: 47787 Voltaje: 7 V
ADC Valor: 43946 Voltaje: 6 V
ADC Valor: 38793 Voltaje: 5 V
ADC Valor: 33304 Voltaje: 5 V
ADC Valor: 27334 Voltaje: 4 V
ADC Valor: 23189 Voltaje: 3 V
ADC Valor: 20709 Voltaje: 3 V
ADC Valor: 19172 Voltaje: 2 V
ADC Valor: 18548 Voltaje: 2 V
ADC Valor: 18100 Voltaje: 2 V
ADC Valor: 17204 Voltaje: 2 V
ADC Valor: 16772 Voltaje: 2 V
ADC Valor: 16163 Voltaje: 2 V
ADC Valor: 16163 Voltaje: 2 V
ADC Valor: 16083 Voltaje: 2 V
```

Este sería el montaje del ejercicio



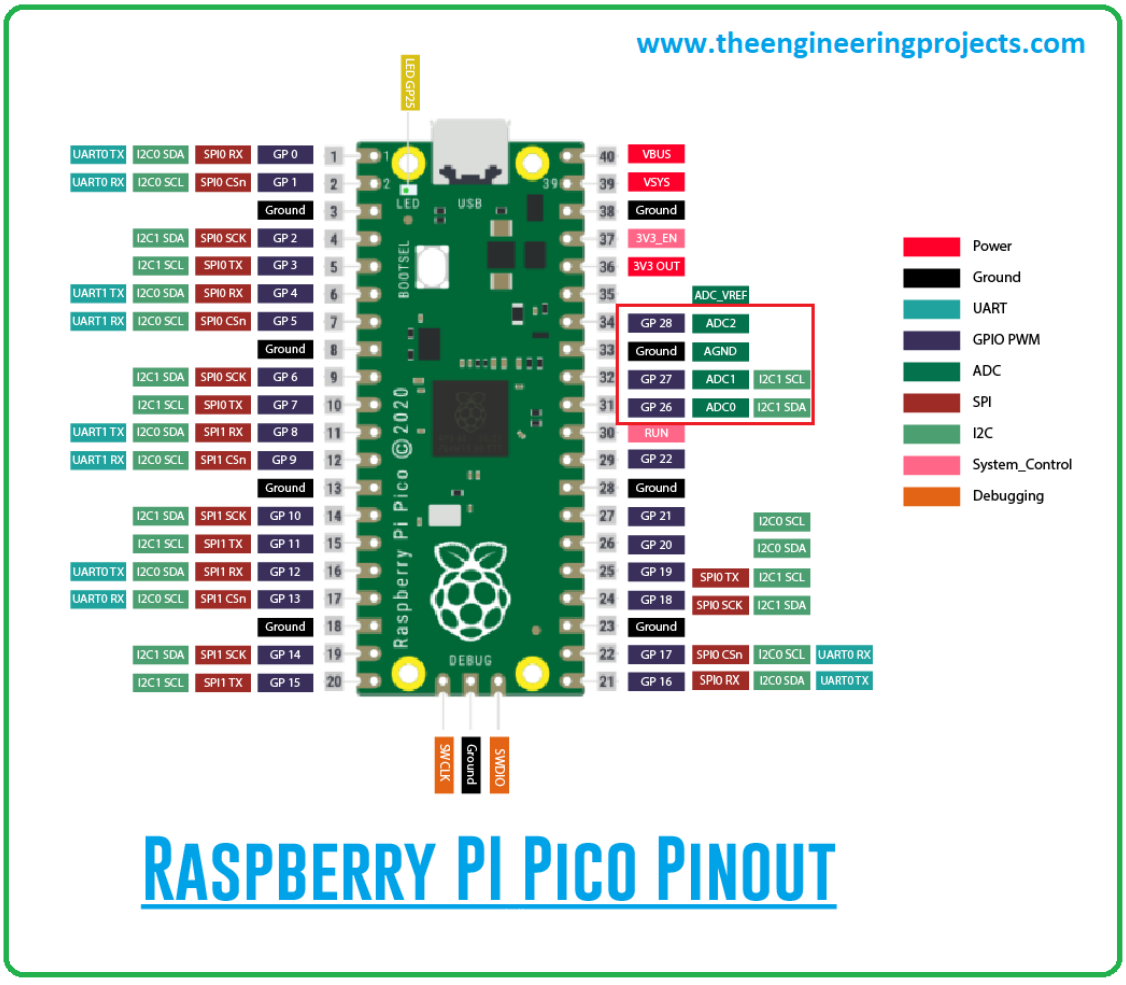
7 Señales analógicas

Raspberry Pi Pico tiene una sola resolución de 12 bits, ADC de 4 canales con una referencia interna de 3.3V y entradas disponibles en 0-3 (GPIO 26-29), el sensor de temperatura del chip interno está en la entrada 4.

Esto significa que puede medir voltajes de entrada entre 0V y 3.3V (Máximo) a una resolución de 0.81mV (3.3 / 4095) aunque en **MicroPython** esto se escala a un valor de 16 bits de 0 a 65535.

En el primer ejemplo el voltaje a través del potenciómetro se lee como una entrada ADC sin procesar que es un valor entre 0 y 65535. Esto se convierte en un valor de voltaje utilizando un factor de conversión que realizara el ajuste del potenciómetro cambiando el valor bruto de un mínimo de **0** a un máximo de **65535** y el voltaje de **0V** a **3.3V aproximadamente.**

Para obtener más información sobre el SDK de Python de Raspberry Pi Pico, visite: https://datasheets.raspberrypi.org/pico/raspberry-pi-pico-python-sdk.pdf



7.1. Leer un valor analógico desde un pin

Objetivo

Con este ejemplo leeremos in pin de entrada analógica GP26 de la tarjeta **Cytron** y lo imprimiremos en la consola de **Thonny**

Funcionalidad

Para generar una señal analógica usaremos un potenciómetro que conectaremos para poner en el pin CP26 el valor necesario. Recordemos que los valores analógicos en Raspberry Pi Pico oscilan entre el valor 0 y el 65535

Programa

1. En primer lugar cargamos las librerías

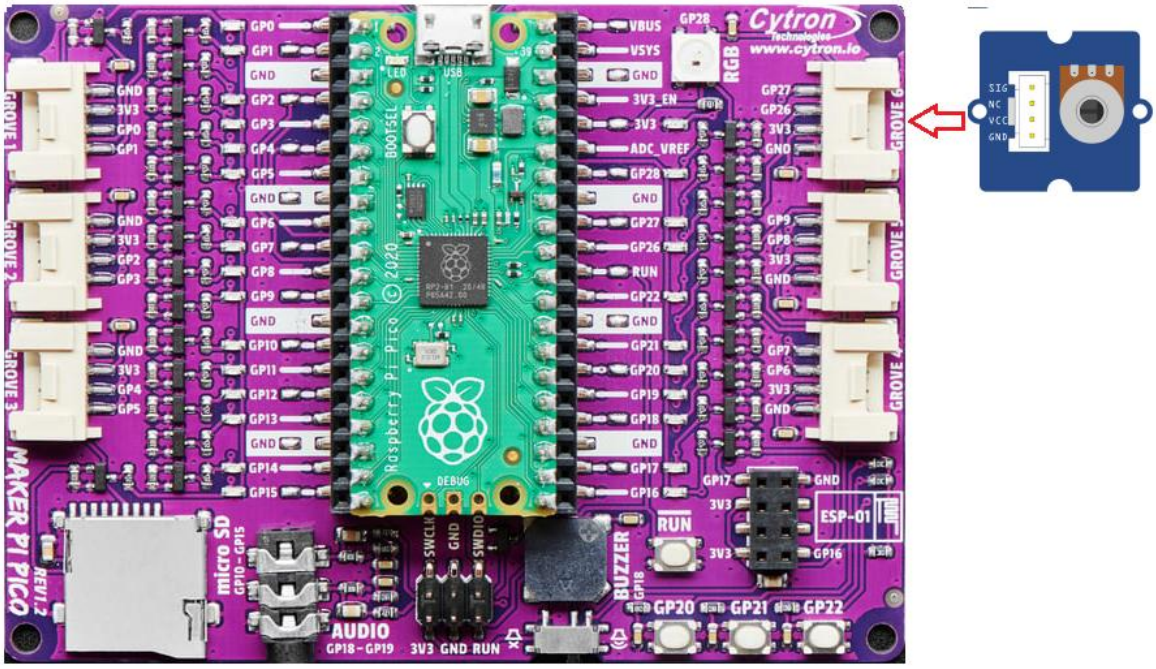
```
from machine import ADC, Pin
import time
```

2. Definimos el objeto “potenciómetro” y le asignamos el canal pin ADC(27)

3. En el bucle **while True** imprimimos el valor de la señal leída (0 a 65535) y seguidamente retardamos 50 ms para evitar colapso del puerto

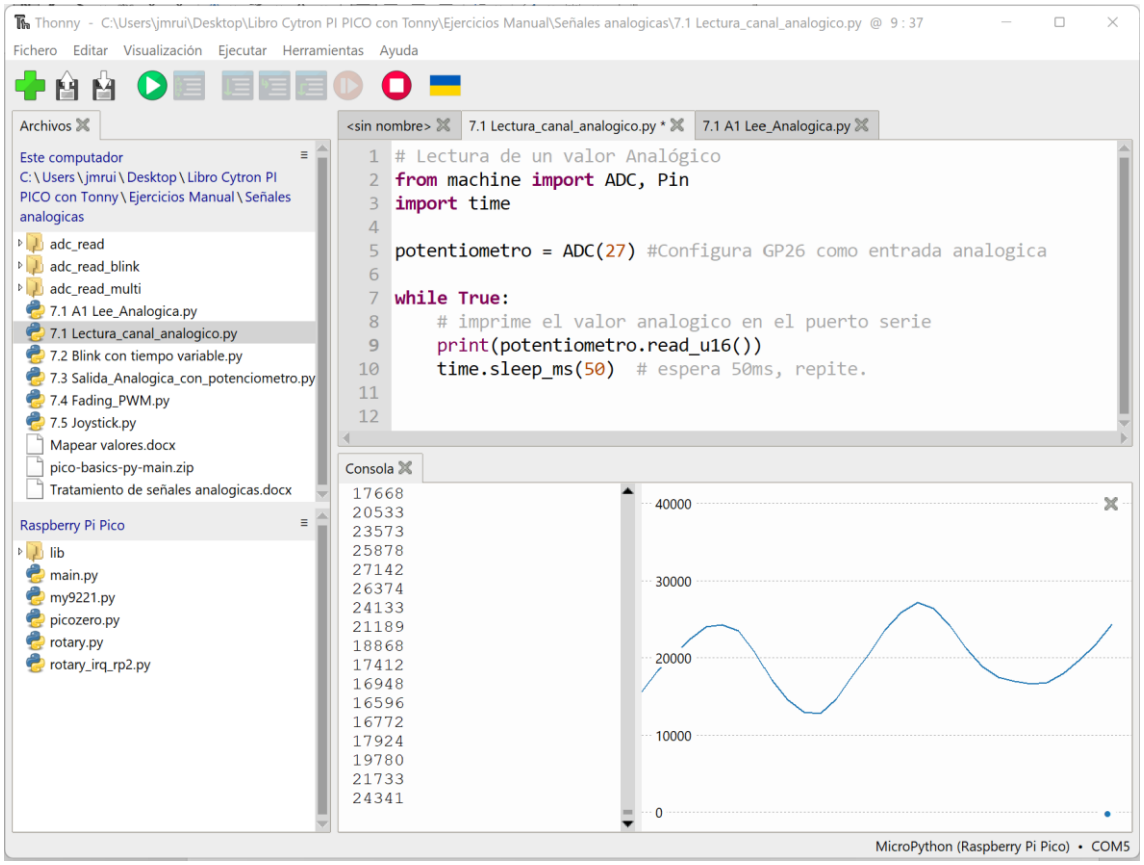
```
# Lectura de un valor Analógico
from machine import ADC, Pin
import time

potentiometro = ADC(27) #Configura GP26 como entrada analogica

while True:
    # imprime el valor analogico en el puerto serie
    print(potentiometro.read_u16())
    time.sleep_ms(50)  # espera 50ms, repite.
```

Montaje de la practica



Análisis y propuesta de actividades

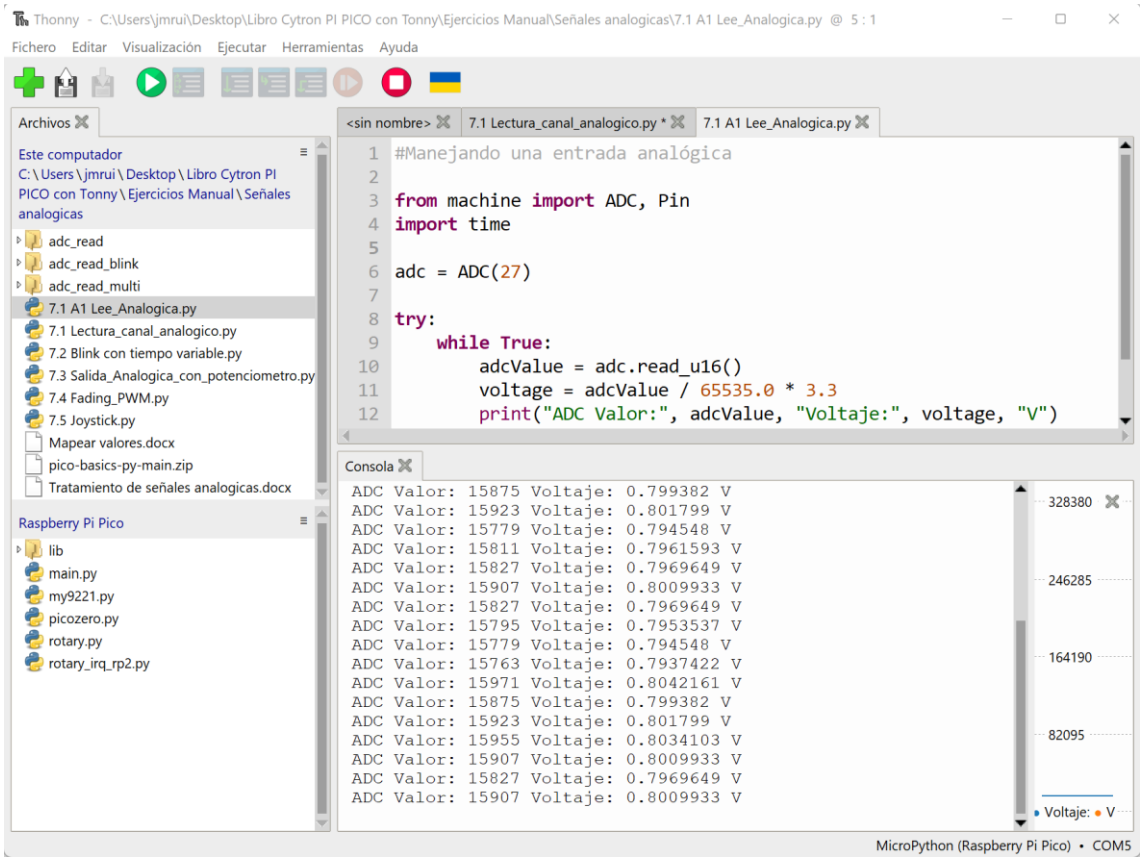
En la figura siguiente vemos el aspecto de la pantalla de **Thonny** estando conectado y ejecutándose el programa.



1. En el siguiente actividad que te proponemos veremos cómo es posible ajustar el valor leído del canal a un valor analógico que variara entre 0 y 3.3 v

```
#Manejando una entrada analógica
from machine import ADC, Pin
import time

adc = ADC(27) #Designamos el pin de lectura en GP27

try:
    while True:
        adcValue = adc.read_u16()
        voltage = adcValue / 65535.0 * 3.3  #Escalamos la señal
        print("ADC Valor:", adcValue, "Voltaje:", voltage, "V")
        time.sleep(0.1)
except:
    pass
```

Este es el aspecto del software con la aplicación funcionando



7.2. Blink con tiempo variable

Objetivo

Con este ejemplo vamos a construir una aplicación en la que un LED se pondrá en forma intermitente y el tiempo de encendido/apagado lo controlaremos con la entrada analógica del pin **GP27**

Funcionalidad

El programa es exactamente igual al que ya vimos en el capítulo anterior 6.1, pero esta vez lo que haremos será obtener el tiempo de retardo del valor leído en la entrada analógica pin GP27 a la que le haremos un mapeado ya que como sabemos el valor que entrega la orden de lectura del canal esta comprendida entre 0 y 65535, nosotros pasaremos este valor a un valor entre 0 y 1 de tal manera que nuestro intervalo de tiempo estará comprendido ente 0 y 1 seg.

En el cuadro de texto siguiente ponemos las diversas formas de crear un mapeado de valor con el fin de fijar conocimientos.

Sobre mapeado de señales

```
#NOTAS IMPORTANTES

#Devuelve un número decimal float
def convert(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min


# Devuelve un entero

def convert(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) // (in_max - in_min) + out_min


# Devuelve un entero entre  out_min and out_max

def convert(x, i_m, i_M, o_m, o_M):
    return max(min(o_M, (x - i_m) * (o_M - o_m) // (i_M - i_m) + o_m), o_m)


# Test

for i in range(200):
    print(i, convert(i, 40, 80, 0, 1023))
```

Programa

El programa se realizará de acuerdo a las siguientes premisas:

1. Importamos las librerías

```
from machine import ADC, Pin
from utime import sleep
```

2. Designamos el objeto pot como una entrada ADC en el pin GP27. Designamos un segundo objeto llamado led en el pin GP1 que será el LED sobre el que realizaremos la función intermitente.

```
pot = ADC(27)
led = Pin(1, Pin.OUT)
```

3. Definimos la función map

```
def map(s, a1, a2, b1, b2): #Mapeado de la señal
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)
```

que nos devolverá el valor mapeado en el intervalo de 0 a 1 cuando asignemos valores a sus parámetros.

```
delay = map(raw, 0, 65535, 0, 1
```

4. Dentro del bucle **while True** leeremos el valor de la variable "**raw**" asignada a la lectura de "**pot**", y asignamos a la variable local **delay** el valor mapeado de la señal. Imprimiremos, para ver en la consola, los valores "raw" y "delay" debidamente formateados para su correcta impresión:

```
print('Valor: {} '.format(raw), 'Retardo {:.1f}s'.format(delay))
```

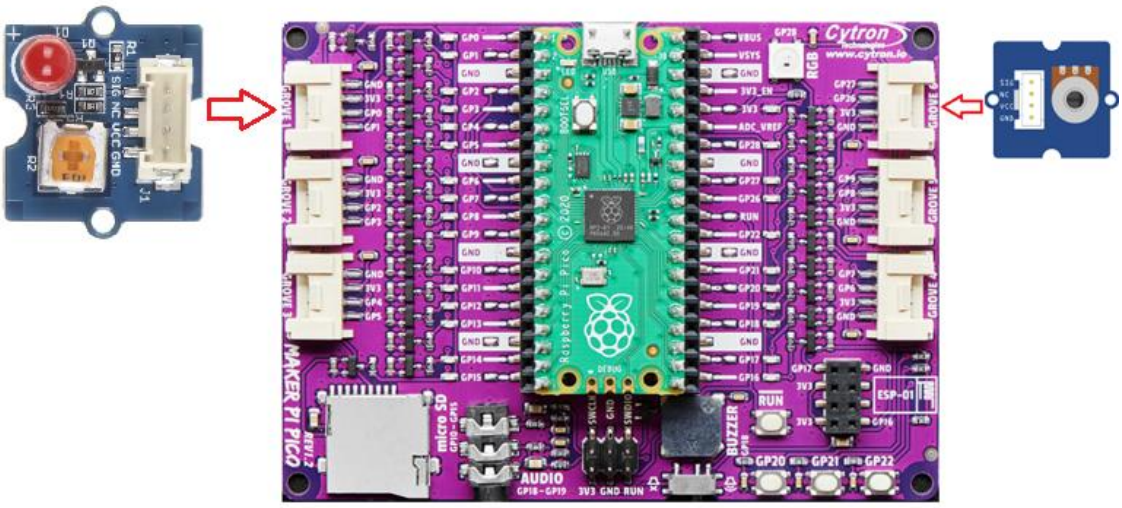
5. Finalmente establecemos el encendido y apagado del LED con su retardo correspondiente

```
led.value(1)
    sleep(delay)
    led.value(0)
    sleep(delay)
```

Este sería el programa completo

```
#Mapeado de una variable analógica
#Controlamos un Blink con tiempo 0 a 1 seg
from machine import ADC, Pin
from utime import sleep

pot = ADC(27)
led = Pin(1, Pin.OUT) #in de salida LED

def map(s, a1, a2, b1, b2): #Mapeado de la señal
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)

while True:
    raw = pot.read_u16()
    delay = map(raw, 0, 65535, 0, 1)
    print('Valor: {} '.format(raw), 'Retardo {:.1f}s'.format(delay))
    led.value(1)
    sleep(delay)
    led.value(0)
    sleep(delay)
```

Montaje de la práctica



Análisis y propuesta de actividades

Una vez cargado el programa se verificara el funcionamiento, comprobando que el valor del tiempo cambia al girar el potenciómetro

7.3. Salida analógica. PWM

Objetivo

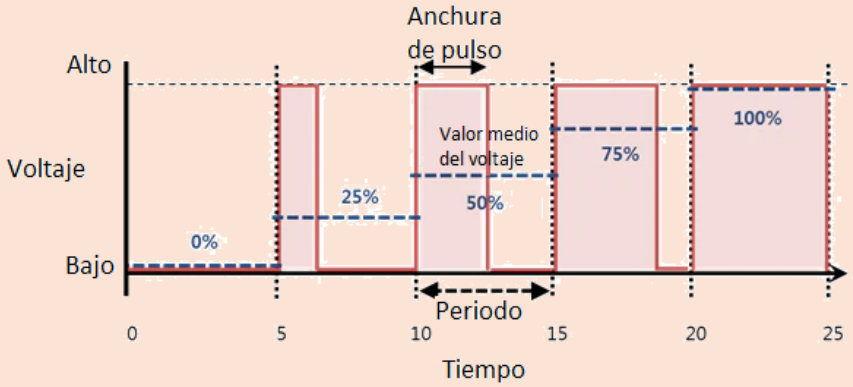
Funcionalidad

Veamos antes de nada un poco de teoría sobre las señales PWM

La señal PWM (Modulación por Ancho de Pulso) es una señal periódica configurable donde se puede modificar el ciclo de trabajo (**Duty Cycle** – En Ingles) utilizando el **MicroPython**.

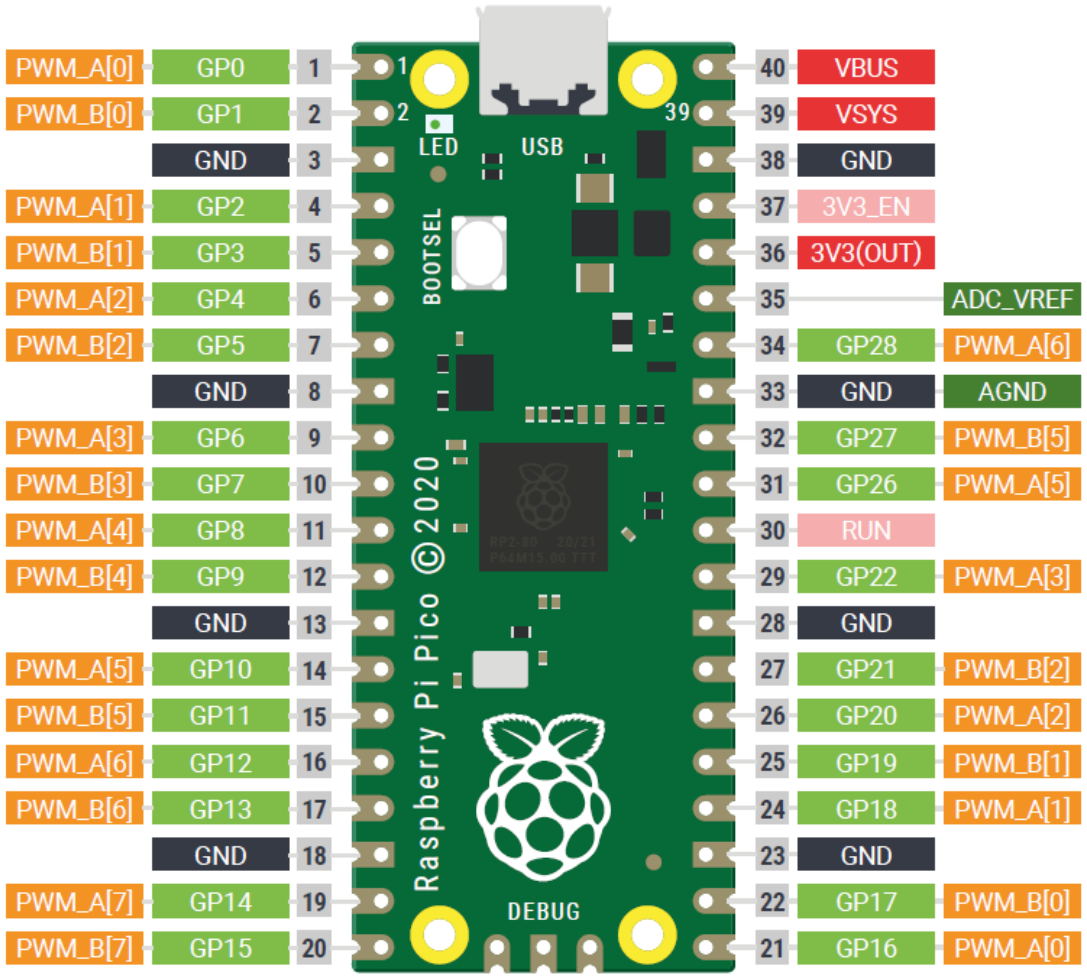
En la página de Sergio Andrés Castaño se muestra información sobre este tema, de ese lugar sacamos la información que colocamos aquí.

Por ejemplo, el PWM de un PIN en una Raspberry pi Pico o en un ESP es simplemente una señal binaria (0v o 3.3v) que podremos configurar para que trabaje un determinado tiempo en **Encendido o 3.3v** y el resto de tiempo en **Apagado o 0v**, y repita este procedimiento infinitamente, pudiendo variar este ancho de pulso en cualquier instante, tal y como se puede observar en la siguiente figura:



Cada pin **GPIO** en la Raspberry Pi Pico puede ser configurado como salida PWM, pero el bloque de modulación de ancho de pulso (PWM) del microcontrolador RP2040 está compuesto por **ocho segmentos**, cada uno con **dos salidas**.

Si observamos el pinout de la Raspberry Pi Pico veremos que cada pin tiene una letra y un número entre corchetes. El número representa el segmento PWM conectado a ese pin; la letra representa qué salida del segmento es utilizado.



PWM Raspberry Pi PICO

Se debe verificar que salidas de PWM se están utilizando bien, evitando conectar en nuestro proyecto pines con una combinación de letras y números que ya hayamos empleado.

Por ejemplo, si utilizamos el **PWM_A[0]** en el pin *GP0* y el **PWM_B[0]** en el pin *GP1*, todo va a funcionar correctamente y continuará funcionando si agregamos un tercer **PWM_A[1]** en el pin *GP2*.

Sin embargo, si intentamos configurar para este caso el PWM del *GP16*, note que ese PWM ya está siendo usado por el *GP0* ya que ambos están conectados a **PWM_A[0]**. Por lo tanto, tendríamos un conflicto si intentamos hacer esa configuración.

PWM con MicroPython

Para configurar un pin PWM en **MicroPython** necesitamos importar la clase **PWM** y la clase **Pin** del módulo **machine**.

```
from machine import Pin, PWM
```

Luego, creamos la instancia o objeto con el método PWM para la Raspberry Pi Pico:

```
led = PWM(Pin(15))
```

```
led.freq(frequancy)
```

O para la NodeMCU ESP8266 podemos hacerlo en una linea:

```
led = PWM(Pin(4), frequency)
```

Para crear un objeto PWM, debemos pasar como parámetros, el pin vamos a configurar, la frecuencia de la señal y el ciclo de trabajo.

```
led.duty_u16(duty_cycle)
```

Con **MicroPython** podemos asignar un ciclo de trabajo para la **Raspberry Pi Pico** un entero de 16 bits sin signo, el mismo formato de número que recibe del pin de entrada analógica. Esto se logra con el uso de la función **duty_u16.**

También podemos usar duty() que sería empleado en el NodeMCU ESP8266 v3 Lolin.

```
led.duty(duty_cycle)
```

Note que para configurar el PWM necesitamos dos parámetros: Frecuencia (frequency) y el ciclo de trabajo (duty_cycle)

- **Frecuencia Raspberry Pi Pico** : La frecuencia puede tener un valor entre 0 y 19200000. Se puede usar una frecuencia de 5000 Hz para controlar el brillo del LED.
- **Ciclo de trabajo Raspberry Pi Pico**: El ciclo de trabajo puede tener un valor entre 0 y 65535. En el cual 65535 corresponde al 100% del ciclo de trabajo (brillo total) y 0 corresponde al 0% del ciclo de trabajo (LED apagado).
- **Frecuencia NodeMCU8266** : La frecuencia puede tener un valor entre 0 y 78125. Se puede usar una frecuencia de 5000 Hz para controlar el brillo del LED. Por defecto la placa está configurada a 1khz.
- **Ciclo de trabajo NodeMCU8266**: El ciclo de trabajo puede tener un valor entre 0 y 1023. En el cual 1023 corresponde al 100% del ciclo de trabajo (brillo total) y 0 corresponde al 0% del ciclo de trabajo (LED apagado).

El ciclo de trabajo podemos configurarlo en el loop principal del programa, por lo tanto no hay necesidad de establecer el ciclo de trabajo al crear una instancia PWM ya que será 0 por defecto.

Para configurar el ciclo de trabajo (duty Cycle), usamos el método duty() del objeto PWM creado y como parámetro de entrada le definimos el ciclo de trabajo deseado.

Dentro de mientras bucle, creamos un por bucle que aumenta el ciclo de trabajo en 1 en cada bucle con un intervalo de 5 ms entre cada cambio.

```
for duty_cycle in range(0, 65535):
  led.duty_u16(duty_cycle)
  sleep(0.005)
```

Nuestro programa será muy sencillo. Se trata de leer el valor de una señal analógica de entrada desde el pin **GP27** y llevar el valor leído a una salida **PWM_B[3]** en el pin **GP7.** Las señales no se mapean. Se designa la frecuencia **pwm.freq(1000)**

Programa

En nuestro programa seguimos las siguientes etapas:

1. Importamos las librerías

```
from machine import ADC, Pin, PWM
import time
```

2. Definimos los objetos adc y pwm

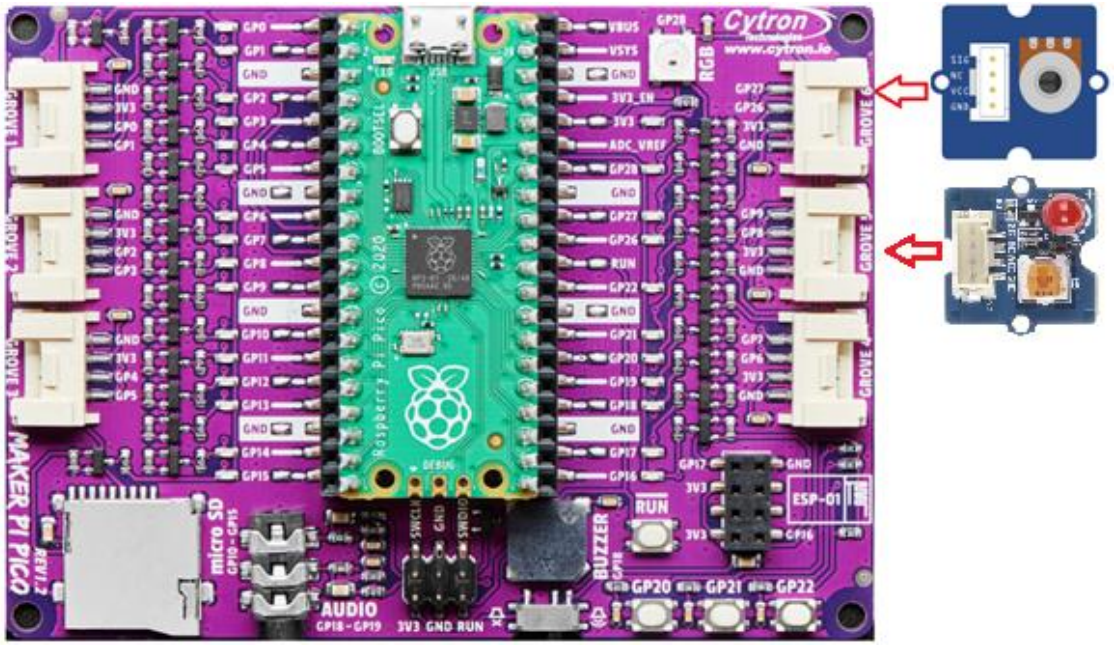
```
adc = ADC(27)
pwm = PWM(Pin(7))
pwm.freq(1000)
```

3. Se utiliza las estructura **try.. except** dentro de la cual colocaremos el bucle **while True** en la parte **try** y la parte **except** el reinicio del objeto **pwm.deinit**

try…. except	Para control de errores
for i in range(…)	Bucle controlado

```
#Control de una salida Analógica PWM con una entrada analógica
from machine import ADC, Pin, PWM
import time
adc = ADC(27)e ut
pwm = PWM(Pin(7))
pwm.freq(1000)

try: #Si todo va bien se ejecuta
   while True:
      #Se lee la señal alaógica en el pin GP27
      adcValue = adc.read_u16()
      #Se lleva el valor leído a la salida PWM pin GP7
      pwm.duty_u16(adcValue)
      time.sleep(0.1) #Esperamos un tiempo
except:
   #Si se produce un error se reinicia el objeto pwm
   pwm.deinit()
```

Montaje de la practica



Análisis y propuesta de actividades

```
from machine import Pin, PWM
import time

#Define el pin GP1 como salida PWM (analógica)
pwm = PWM(Pin(1))
pwm.freq(10000)

#En esta aplicación ensayamos las sentencias try y except para
#control de errores, aunque no es necesario hacerlo
try:      #Código a ejecutar
   while True:
      for i in range(0, 65535, 1):        #Cuenta ascendente
         pwm.duty_u16(i)                        #Escribe en la salida el valor i
         time.sleep_us(100)              #Retardo en el contador
      for i in range(65535, 0, -1): #Cuenta descendente
         pwm.duty_u16(i)
         time.sleep_us(100)
except:#En caso de error se ejecuta  lo que sigue
   pwm.deinit()         #Reinicia la salida pwn
```

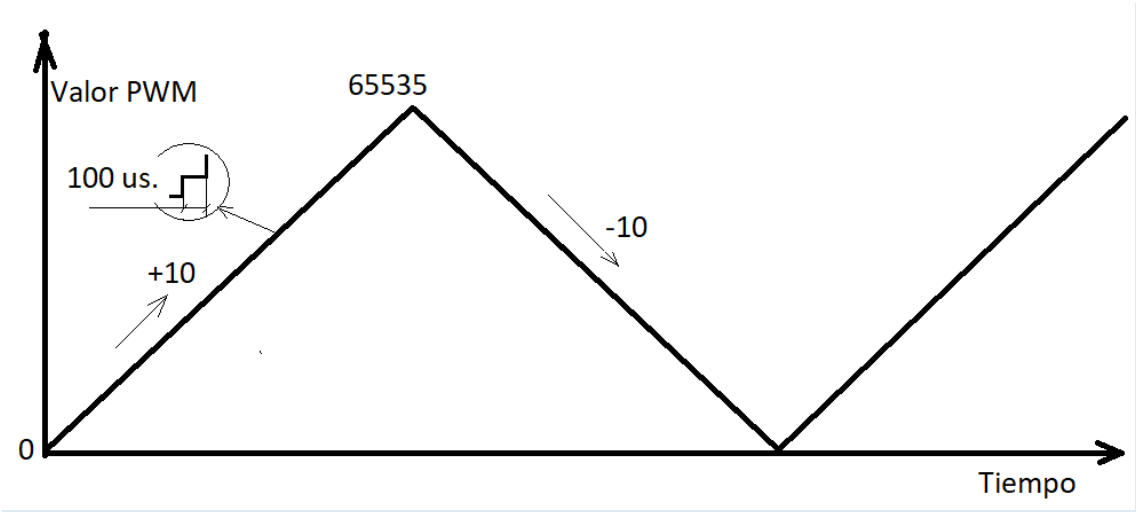
7.4. Efecto fadding de salida analógica

Objetivo

Se trata de emular el efecto "fading" en el encendido de un LED haciendo uso una de una salida PWM

Funcionalidad

Se trata de sacar los valores PWM en el GP1. Creamos un bucle de cuenta ascendente con la función **for i in range(0, 65535, 10**) y otro descendente **for i in range(65535, 0, -10)** en la que el valor de la variable **"i"** es el que se escribe en la salida **pwm.duty_u16(i).**



Programa

En nuestro programa seguimos las siguientes etapas:

1. Importamos las librerías

```
from machine import ADC, Pin, PWM
import time
```

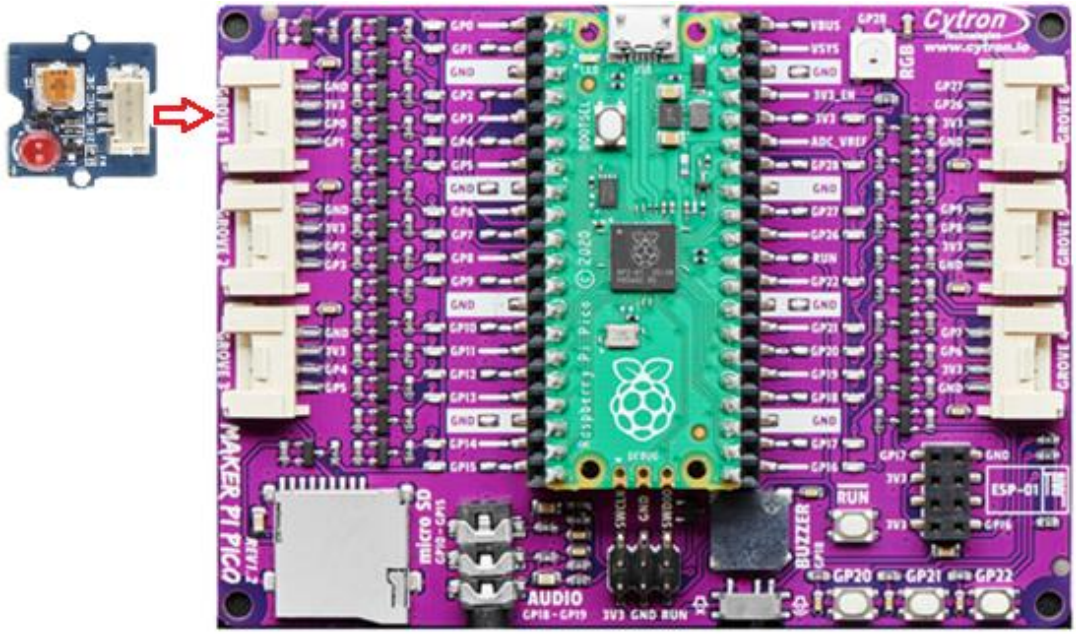
2. Definimos el objeto pwm que será una salida en el pin GP1 y fijamos la frecuencia de la señal base del PWM en 1000 hz

```
adc = ADC(27)
pwm.freq(1000)
```

3. Seguidamente en la estructura **try… except** incluimos el bucle **while True** colocamos los dos bucles contadores y escribimos en la salida los valores de la variable "**i**" en la salida **PWM y** la orden para la parte **except** que sería inicializar el objeto PWM **pwm.deinit()**

```
#Efecto Fading
from machine import Pin, PWM
import time

#Define el pin GP1 como salida PWM (analógica)
pwm = PWM(Pin(1))
pwm.freq(10000)

#En esta aplicación ensayamos las sentencias try y except para
#control de errores, aunque no es necesario hacerlo
try:     #Código a ejecutar
   while True:
      for i in range(0, 65535, 10):     #Cuenta ascendente
         pwm.duty_u16(i)                #Escribe en la salida el valor i
         time.sleep_us(100)             #Retardo en el contador
      for i in range(65535, 0, -10):    #Cuenta descendente
         pwm.duty_u16(i)
         time.sleep_us(100)
except:  #En caso de error se ejecuta  lo que sigue
   pwm.deinit()
```

Montaje de la practica



Análisis y propuesta de actividades

Observando el funcionamiento intenta modificar las variables:

1. Valor del incremento/decremento de los contadores
2. Valor del retardo en la cuenta

7.5 Joystick

Objetivo

En esta práctica vamos a estudiar el funcionamiento del dispositivo Joystick que sabemos es de gran uso en los sistemas de control

Funcionalidad

El dispositivo presenta 4 pines: GND, VCC, X e Y

Por los pines X e Y se obtiene u valor de tensión en función de la posición del mando.

Al pulsar hacia abajo el mando el valor de la tensión en el pin Y se hace Vy=3,3 v.

Lo que vamos a hacer es insertar el dispositivo en el puerto Grove que ofrece los pines analógicos GP27 y GP26 y de ellos leeremos los valores analógicos.

Vamos a monitorizar los valores X e Y en la consola de **Thonny** y también vamos a detectar la pulsación del mando testeando el valor Vy=3.3v

Programa

1. En primer lugar cargamos como siempre hacemos las librerías.

```
from machine import ADC,Pin
import time
```

2. Seguidamente escribimos la función de mapeado que nos permitirá convertir el rango de valor original de la lectura del pin (0 a 65535) al rango (0 a 10) para cada una de las dos señales X e Y que leemos.

```
def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)
```

3. Lo siguiente es definir los pines de entrada analógica y la variable de salida digital "botón" que es la que nos devuelve el estado del botón.

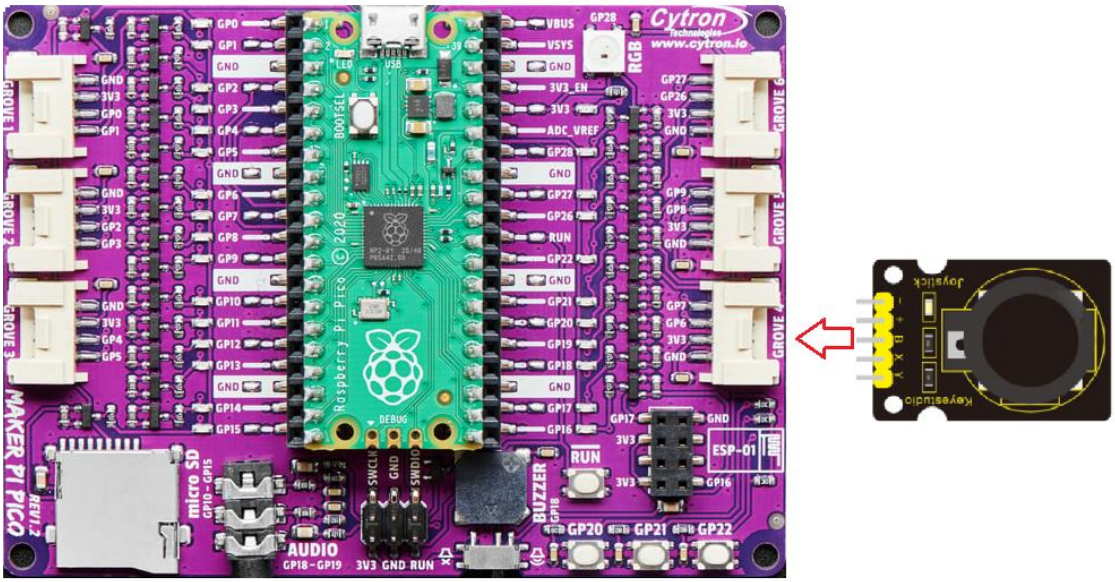
```
xValue = ADC(26)
yValue = ADC(27)
boton = Pin(1, Pin.OUT)
```

4. En el bucle **while True** lo que hacemos es leer los valores analógicos a los que llamamos x e y , los mapeamos e imprimimos dichos valores con el adecuado formato de impresión y recorte de decimales. Mediante un condicional detectamos el valor del parámetro valor de y>9 y activamos o no la salida en el pin GP1

```
#Experimentación con un Joystick
from machine import ADC,Pin
import time
# Funcion mapeado
def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)
#Definimos los canales de lectura analógica
xValue = ADC(26)
yValue = ADC(27)
boton = Pin(1, Pin.OUT)

while True:
   x = xValue.read_u16()
   y = yValue.read_u16()
   #Mapeado de los valores x e y
   x= map(x, 0, 65530, 0, 10)
   y= map(y, 0, 65530, 0, 10)
   #Imprimimos solo dos decimales "{:.2f}".format(x)
   print("X, Y:", "{:.2f}".format(x), "{:.2f}".format(y))
   #Salida del botón pulsado del Joystick
   if y > 9.00:
      boton.value(1)
   else:
      boton.value(0)
   time.sleep(0.1)
```

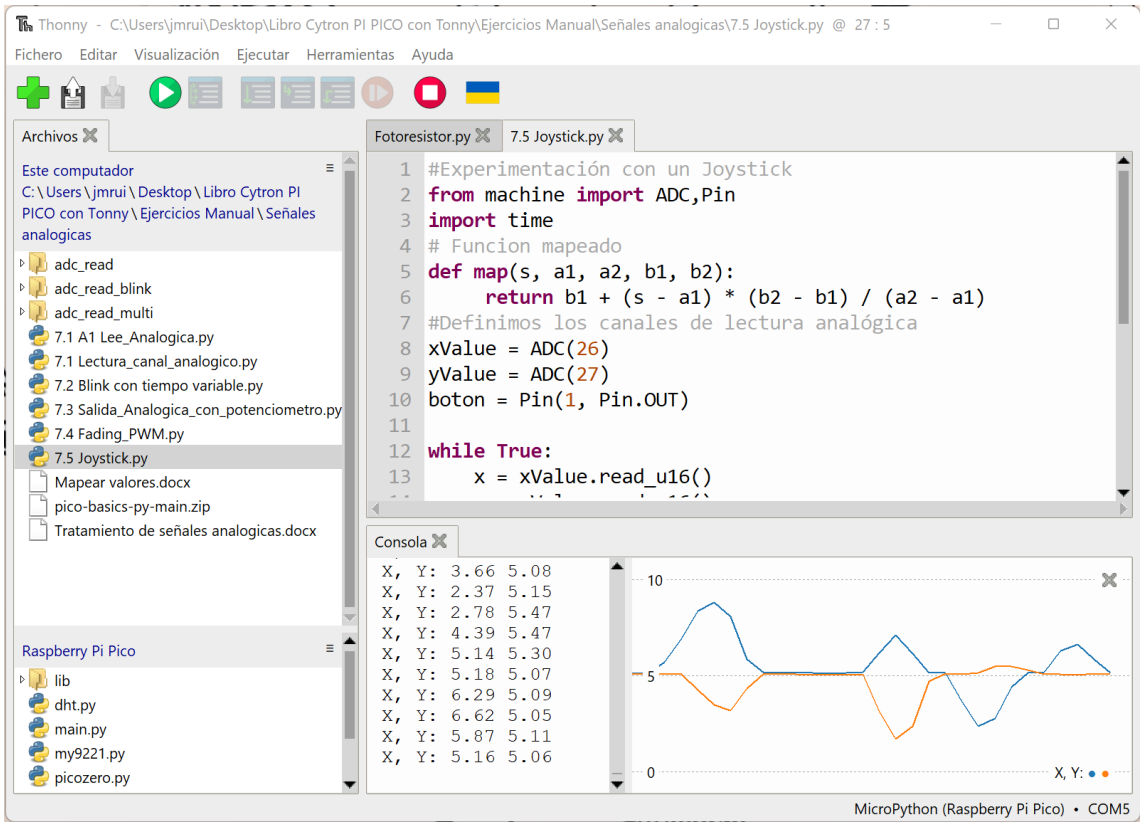
Montaje de la practica

El montaje de la práctica es el que se muestra en la siguiente imagen.



Análisis y propuesta de actividades

Una vez descargado el programa procederemos a probarlo y podemos visualizar en la consola los valores así como el trazado mediante la opción plot



8. Sonido

Conceptos introductorios

Funciones mas importantes en la programación del zumbador

1. Comience importando Pin y PWM desde la biblioteca de máquinas y duerma desde la biblioteca utime.

   ```
   from machine import Pin, PWM
   from utime import sleep
   ```

2. **Inicialice PWM** (también conocido como modulación de ancho de pulso) en el Pin 15 y asígnelo al zumbador variable.

   ```
   buzzer = PWM(Pin(15))
   ```

3. Asigne una propiedad freq al zumbador. Debe elegir un número que oscile entre 10 y 12,000. Cuanto mayor sea el número, más alto será el sonido. Probemos 500.

   ```
   buzzer.freq(500)
   ```

5. **Establezca la propiedad duty_u16** del objeto buzzer **en 1000**. Esto hace que el timbre sea lo más fuerte posible. Un valor más bajo es más silencioso y 0 no es sonido en absoluto. Teniendo en cuenta lo silenciosos que son estos zumbadores, el volumen máximo no es muy alto en absoluto.

   ```
   buzzer.duty_u16(1000)
   ```

6. **Establezca un retraso de 1 segundo** y luego **establezca el deber en 0** para que el sonido se detenga. Si no hace esto, el zumbido continuará, incluso después de que el programa haya terminado de ejecutarse.

   ```
   sleep(1)
   buzzer.duty_u16(0)
   ```

   Su código final para esta prueba simple debería verse así.

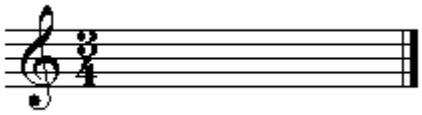
```
from machine import Pin, PWM
from utime import sleep

buzzer = PWM(Pin(15))
buzzer.freq(500)
buzzer.duty_u16(1000)
sleep(1)
buzzer.duty_u16(0)
```

8.1. Emitir un tono

Objetivo

Usando las funciones de la librería **PWM** podemos emite un sonido en el pin correspondiente a la salida del Buzzer de la tarjeta **Cytron** pin **GP18**

Funcionalidad



La aplicación se basa en emitir un tono de frecuencia 500 mediante la asignación de la propiedad freq de la librería: **buzzer.freq(500)**

Programa

1. Para empezar importamos las librerías

```
from machine import Pin, PWM
from utime import sleep
```

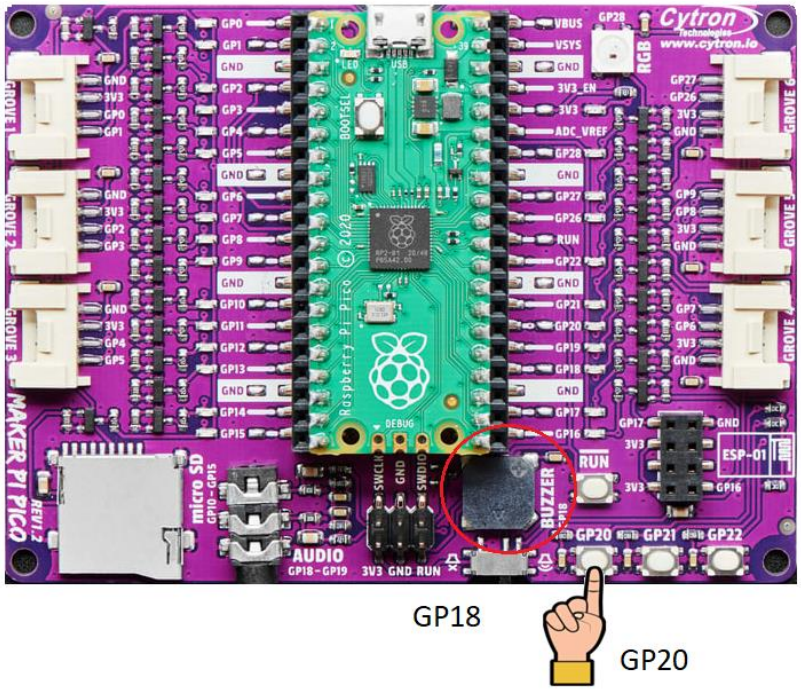
2. Definimos el objeto "**buzzer**" que representa el dispositivo de salida conectado en el pin GP18.

3. Seguidamente designamos la frecuencia del tono a producir **buzzer.freq(500)**

4. Fijamos el nivel del volumen del sonido con **buzzer.duty_u16(8000).** Seguidamente retardamos un tiempo **sleep(1)**, tiempo que se mantiene el sonido, para finalmente bajar el volumen del sonido a 0 **buzzer.duty_u16(0).**

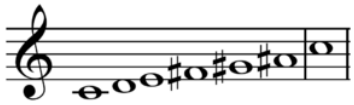
```
#Emitir un tono de 500 Hz
from machine import Pin, PWM
from utime import sleep

buzzer = PWM(Pin(18)) #Define pin de salida
buzzer.freq(500) #Frecuencia del tono
buzzer.duty_u16(8000) #Volumen del sonido
sleep(1)
buzzer.duty_u16(0) #Silencio
```

Montaje de la practica



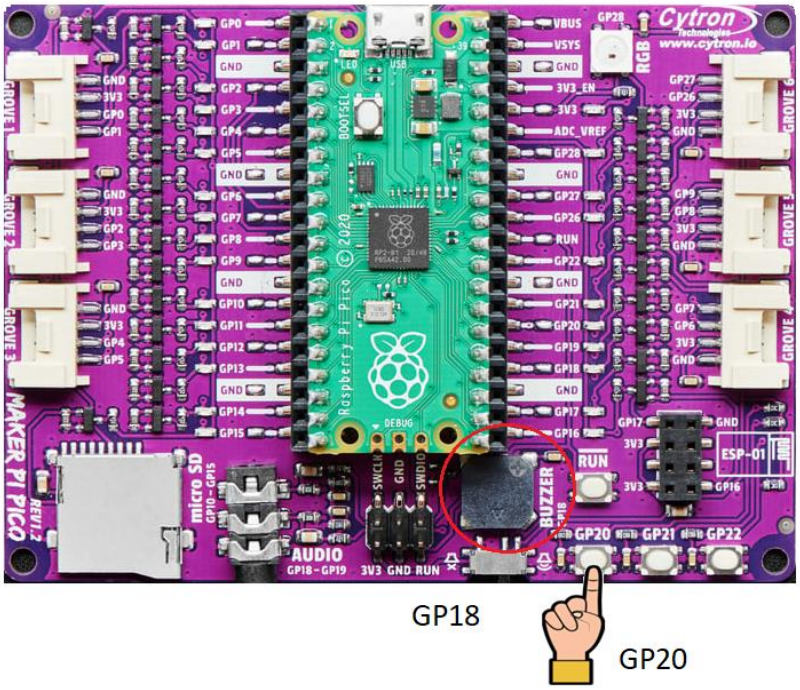
Análisis y propuesta de actividades

1. Te proponemos que crees un efecto de “variación tonal” realizando un barrido de la frecuencia desde el valor 500 al valor 5000 en pasos de 500.



```
#Emitir una escala de tonos desde 500 a 5000 Hz
from machine import Pin, PWM
from utime import sleep

buzzer = PWM(Pin(18)) #Define pin de salida
boton = Pin(20,Pin.IN)

while True:
    if boton.value() == 0:
        #Definimos el rango de valores y el incremento
        for i in range(500,5000,500):
            buzzer.freq(i) #Frecuencia del tono
            buzzer.duty_u16(8000) #Volumen del sonido
            sleep(1)
            buzzer.duty_u16(0) #Silencio
    else:
        buzzer.duty_u16(0)
```

Montaje de la practica

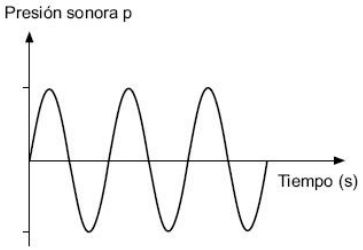


8.2. Alarma Sirena

Objetivo

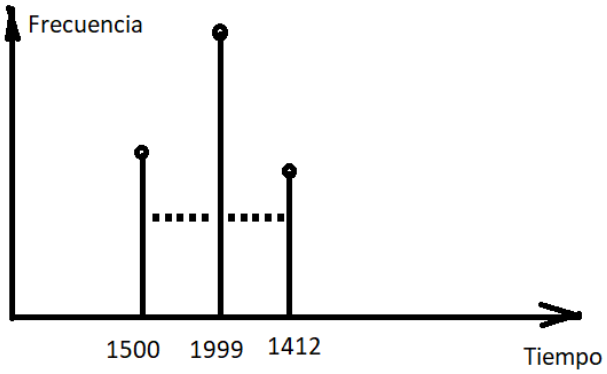
En ocasiones tenemos que crear aplicaciones en las que se deba generar un sonido de alarma para advertir de un evento determinado. En este caso al pulsar un botón emitiremos un sonido de alarma cuya señal tonal responde a una función senoidal

Funcionalidad



Nuestro espectro tonal responderá a la función senoidal de tal manera que a una frecuencia base le sumaremos un valor que varía de manera senoidal.

sinVal = math.sin(x * 10 * PI / 180)
toneVal = 1500+int(sinVal*500)



Programa

1. Para la realización del programa empezamos con la importación de las librerías.

```
from machine import Pin,PWM
import math
import time
```

2. Seguidamente definimos el parámetro **PI=3.14**, la entrada de **botón** pin **GP20** del botón integrado en placa **Cytron** y el **Buzzer** y asignamos el valor 1000 a la frecuencia de la señal PWM del buzzer: buzzer.freq(1000)

3. La definición del tono y su construcción se hace mediante una función a la que llamaremos **alert()** que viene a ser un bucle en el que dentro de un rango

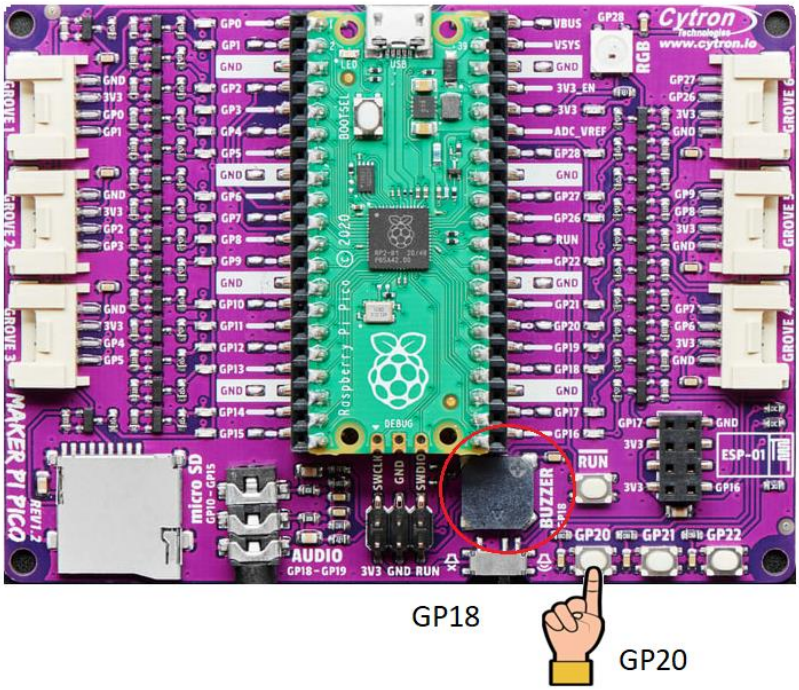
establecemos el valor de la señal senoidal que sumamos al tono base de valor 1500 hz

```
def alert():  #Contruye una señal analogica senoidal
    for x in range(0, 36):
        sinVal  = math.sin(x * 10 * PI / 180)
        toneVal = 1500+int(sinVal*500)
        buzzer.freq(toneVal)
        time.sleep_ms(10)
```

4. Seguidamente creamos el bucle **while True** dentro del cual establecemos un condicional el nivel de sonido e invocamos la función **alert()** para su ejecución

```
#Alarma
from machine import Pin,PWM
import math
import time
PI = 3.14
boton = Pin(20, Pin.IN, Pin.PULL_UP)
buzzer = PWM(Pin(18))
buzzer.freq(1000)

def alert():  #Contruye una señal analogica senoidal
    for x in range(0, 36):
        sinVal  = math.sin(x * 10 * PI / 180)
        toneVal = 1500+int(sinVal*500)
        buzzer.freq(toneVal)
        time.sleep_ms(10)

try:
    while True:
        if not boton.value():
            #Control volumen min=0 max=65535
            buzzer.duty_u16(8000)
            alert()
        else:
            #Volumen 0 en silencio
            buzzer.duty_u16(0)
except:
    buzzer.deinit()
```

Montaje de la practica



Análisis y propuesta de actividades

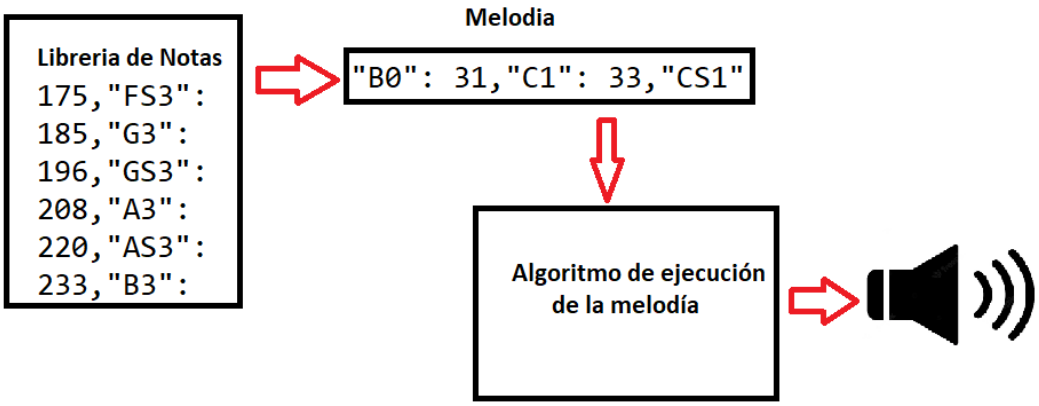
8.3 Generacion de una melodia

Objetivo

Generación de melodías mediante la ejecución de listas de tonos

Funcionalidad

En esta ocasión vamos a generar melodías mediante el uso de una tabla de tonos o notas de la que podemos crear una lista con nuestra propia melodía.



Programa

Vamos a construir el programa

Debido a que puedes usar diferentes frecuencias, puedes crear una gama completa de notas musicales. Hay algunas listas de frecuencias de notas musicales en línea y muchas de ellas se remontan a la biblioteca de tonos Arduino de Brett Hagman en Github. Usaremos estos valores para crear una lista que podamos usar para reproducir cualquier canción al principio de nuestro código.

1. **Importa las librerías necesarias** e **inicializa PWM en el pin 15**.

```
from machine import Pin, PWM
from utime import sleep

buzzer = PWM(Pin(15))
```

3. **Crea un diccionario llamado tones** con los siguientes valores.

```
tones = {
"B0": 31,"C1": 33,"CS1": 35,"D1": 37,"DS1": 39,"E1": 41,"F1": 44,"FS1":
46,"G1": 49,"GS1": 52,"A1": 55,"AS1": 58,"B1": 62,"C2": 65,"CS2": 69,"D2":
73,"DS2": 78,"E2": 82,"F2": 87,"FS2": 93,"G2": 98,"GS2": 104,"A2":
110,"AS2": 117,"B2": 123,"C3": 131,"CS3": 139,"D3": 147,"DS3": 156,"E3":
```

```
165,"F3": 175,"FS3": 185,"G3": 196,"GS3": 208,"A3": 220,"AS3": 233,"B3":
247,"C4": 262,"CS4": 277,"D4": 294,"DS4": 311,"E4": 330,"F4": 349,"FS4":
370,"G4": 392,"GS4": 415,"A4": 440,"AS4": 466,"B4": 494,"C5": 523,"CS5":
554,"D5": 587,"DS5": 622,"E5": 659,"F5": 698,"FS5": 740,"G5": 784,"GS5":
831,"A5": 880,"AS5": 932,"B5": 988,"C6": 1047,"CS6": 1109,"D6":
1175,"DS6": 1245,"E6": 1319,"F6": 1397,"FS6": 1480,"G6": 1568,"GS6":
1661,"A6": 1760,"AS6": 1865,"B6": 1976,"C7": 2093,"CS7": 2217,"D7":
2349,"DS7": 2489,"E7": 2637,"F7": 2794,"FS7": 2960,"G7": 3136,"GS7":
3322,"A7": 3520,"AS7": 3729,"B7": 3951,"C8": 4186,"CS8": 4435,"D8":
4699,"DS8": 4978
}
```

4. **Crea una lista (**también conocida como matriz**) de notas** para tu canción. Usa la letra P para representar pausas en la música. Cada nota debe estar entre comillas.

```
song =
["E5","G5","A5","P","E5","G5","B5","A5","P","E5","G5","A5","P","G5","E5"]
```

5. **Crea una función llamada playtone** que tomará cualquier frecuencia y la reproducirá a todo volumen.

```
def playtone(frequency):
    buzzer.duty_u16(1000)
    buzzer.freq(frequency)
```

6. **Crea una función llamada bequiet** que silenciará el zumbador cambiando duty_u16 a 0.

```
def bequiet():
    buzzer.duty_u16(0)
```

7. **Crea una función llamada playsong** que usarás para iterar a través de la matriz de notas y reproducir cada una o hacer una pausa cuando vea P.

```
def playsong(mysong):
    for i in range(len(mysong)):
        if (mysong[i] == "P"):
            bequiet()
        else:
            playtone(tones[mysong[i]])
        sleep(0.3)
    bequiet()
try:
    while True:
        if not button.value():
            #Control volumen min=0 max=65535
            playsong(song)
        else:
```

```
            #Volumen 0 en silencio
            bequiet()
except:
   buzzer.deinit()
```

Esto es lo que está pasando aquí. Primero, creamos un bucle for que itera a través de todos los valores en la matriz mysong. Si el valor es igual a P, activa be quiet y, si no, activa playtone. Tenga en cuenta que el tono de reproducción requiere un valor de frecuencia numérico, por lo que tenemos que obtener el número para cada nota de la lista de tonos. Si solo tocamos el tono (mysong[i]), fallará porque intentará tocar la cuerda "E5", en lugar de la frecuencia 659, que es el entero que necesita.

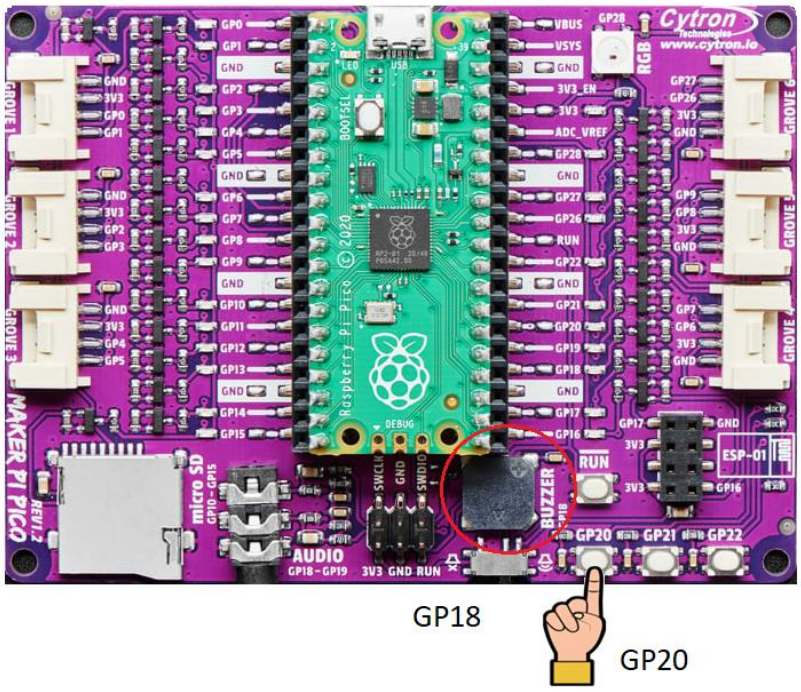
Para cada tono o pausa, el sistema mantiene el estado durante 0,3 segundos de retrdo. Si quieres un tempo más rápido, puedes bajar ese tiempo. Si quieres un tempo más lento, auméntalo.

Este sería el programa completo

```
from machine import Pin, PWM
from utime import sleep
buzzer = PWM(Pin(18))
button = Pin(20, Pin.IN, Pin.PULL_UP)
#Definicion de lista de tonos
tones = {
"B0": 31,"C1": 33,"CS1": 35,"D1": 37,"DS1": 39,"E1": 41,"F1": 44,"FS1": 46,"G1":
49,"GS1": 52,"A1": 55,"AS1": 58,"B1": 62,"C2": 65,"CS2": 69,"D2": 73,"DS2":
78,"E2": 82,"F2": 87,"FS2": 93,"G2": 98,"GS2": 104,"A2": 110,"AS2": 117,"B2":
123,"C3": 131,"CS3": 139,"D3": 147,"DS3": 156,"E3": 165,"F3": 175,"FS3": 185,"G3":
196,"GS3": 208,"A3": 220,"AS3": 233,"B3": 247,"C4": 262,"CS4": 277,"D4":
294,"DS4": 311,"E4": 330,"F4": 349,"FS4": 370,"G4": 392,"GS4": 415,"A4":
440,"AS4": 466,"B4": 494,"C5": 523,"CS5": 554,"D5": 587,"DS5": 622,"E5":
659,"F5": 698,"FS5": 740,"G5": 784,"GS5": 831,"A5": 880,"AS5": 932,"B5":
988,"C6": 1047,"CS6": 1109,"D6": 1175,"DS6": 1245,"E6": 1319,"F6": 1397,"FS6":
1480,"G6": 1568,"GS6": 1661,"A6": 1760,"AS6": 1865,"B6": 1976,"C7": 2093,"CS7":
2217,"D7": 2349,"DS7": 2489,"E7": 2637,"F7": 2794,"FS7": 2960,"G7": 3136,"GS7":
3322,"A7": 3520,"AS7": 3729,"B7": 3951,"C8": 4186,"CS8": 4435,"D8": 4699,"DS8":
4978
}

#Defician de Melodia como conjunto de tonos
song = ["E5","G5","A5","P","E5","G5","B5","A5","P","E5","G5","A5","P","G5","E5"]

def playtone(frequency):
   buzzer.duty_u16(8000)
   buzzer.freq(frequency)

def bequiet():
   buzzer.duty_u16(0)
```

```
def playsong(mysong):
    for i in range(len(mysong)):
        if (mysong[i] == "P"):
            bequiet()
        else:
            playtone(tones[mysong[i]])
        sleep(0.3)
    bequiet()
try:
    while True:
        if not button.value():
            #Control volumen min=0 max=65535
            playsong(song)
        else:
            #Volumen 0 en silencio
            bequiet()
except:
    buzzer.deinit()
```

Montaje de la practica



Análisis y propuesta de actividades

1. Realizar la practica anterior pero mediante la creación y uso de una función llamada **playtone** que tomará cualquier frecuencia y la reproducirá a un determinado volumen.

```
def playtone(frequency):
  buzzer.duty_u16(1000)
  buzzer.freq(frequency)
```

2. **Cree una función llamada bequiet** que silenciará el zumbador cambiando duty_u16 a 0.

```
def bequiet():
  buzzer.duty_u16(0)
```

3. **Cree una función llamada playsong** que usará para iterar a través de la matriz de notas **mysong** y reproducir cada una o hacer una pausa cuando vea P.

```
def playsong(mysong):
  for i in range(len(mysong)):
    if (mysong[i] == "P"):
      bequiet()
    else:
      playtone(tones[mysong[i]])
    sleep(0.3)
  bequiet()
```

Esto es lo que está pasando aquí. Primero, creamos un bucle **for** que itera a través de todos los valores en la matriz **mysong**. Si el valor es igual a **P**, activa be **quiet** y, si no, activa **playtone**. Tenga en cuenta que el tono de reproducción requiere un valor de frecuencia numérico, por lo que tenemos que obtener el número para cada nota de la lista de tonos. Si solo tocamos el tono (**mysong[i]),** fallará porque intentará tocar la cuerda "**E5**", en lugar de la frecuencia **659**, que es el entero que necesita.

Para cada tono o pausa, el sistema mantiene el estado durante 0,3 segundos de sueño. Si quieres un tempo más rápido, puedes bajar ese tiempo. Si quieres un tempo más lento, auméntalo.

4. **Active la función playsong** con el parámetro song.

```
playsong(song)
```

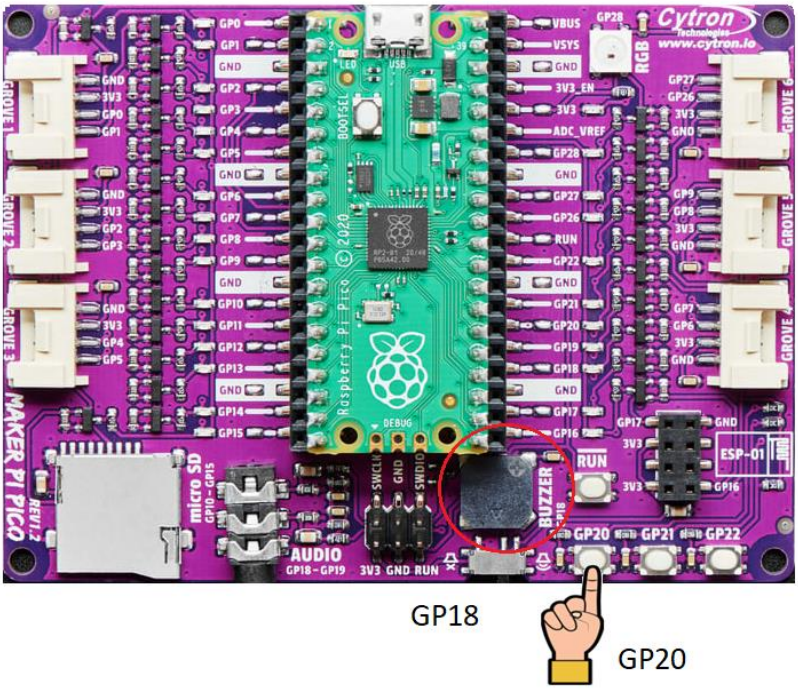
Este sería el código completo

Cuando ejecute este código, escuchará su timbre tocar una melodía familiar. Así es como debería verse el código completo:

```
from machine import Pin, PWM
from utime import sleep
buzzer = PWM(Pin(18))
```

```
tones = {
"B0": 31,"C1": 33,"CS1": 35,"D1": 37,"DS1": 39,"E1": 41,"F1": 44,"FS1":
46,"G1": 49,"GS1": 52,"A1": 55,"AS1": 58,"B1": 62,"C2": 65,"CS2":
69,"D2": 73,"DS2": 78,"E2": 82,"F2": 87,"FS2": 93,"G2": 98,"GS2":
104,"A2": 110,"AS2": 117,"B2": 123,"C3": 131,"CS3": 139,"D3": 147,"DS3":
156,"E3": 165,"F3": 175,"FS3": 185,"G3": 196,"GS3": 208,"A3": 220,"AS3":
233,"B3": 247,"C4": 262,"CS4": 277,"D4": 294,"DS4": 311,"E4": 330,"F4":
349,"FS4": 370,"G4": 392,"GS4": 415,"A4": 440,"AS4": 466,"B4": 494,"C5":
523,"CS5": 554,"D5": 587,"DS5": 622,"E5": 659,"F5": 698,"FS5": 740,"G5":
784,"GS5": 831,"A5": 880,"AS5": 932,"B5": 988,"C6": 1047,"CS6":
1109,"D6": 1175,"DS6": 1245,"E6": 1319,"F6": 1397,"FS6": 1480,"G6":
1568,"GS6": 1661,"A6": 1760,"AS6": 1865,"B6": 1976,"C7": 2093,"CS7":
2217,"D7": 2349,"DS7": 2489,"E7": 2637,"F7": 2794,"FS7": 2960,"G7":
3136,"GS7": 3322,"A7": 3520,"AS7": 3729,"B7": 3951,"C8": 4186,"CS8":
4435,"D8": 4699,"DS8": 4978
}

song =
["E5","G5","A5","P","E5","G5","B5","A5","P","E5","G5","A5","P","G5","E5
"]

def playtone(frequency):
    buzzer.duty_u16(1000)
    buzzer.freq(frequency)

def bequiet():
    buzzer.duty_u16(0)

def playsong(mysong):
    for i in range(len(mysong)):
        if (mysong[i] == "P"):
            bequiet()
        else:
            playtone(tones[mysong[i]])
        sleep(0.3)
    bequiet()
playsong(song)
```

Así que ahora sabes cómo tocar música con la Raspberry Pi Pico y un timbre piezoeléctrico. Puedes hacer que esto reproduzca cualquier canción, siempre que tengas las notas.

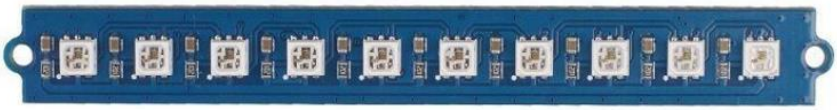

Cytron
www.cytron.io
MAKER PI PICO
Raspberry Pi Pico
BUZZER
GP18
GP20
GP21
GP22
RUN
ESP-01
AUDIO
micro SD
RGB
GP18
GP20

9. Jugando con Neopixel

Algunos conceptos importantes:

Las tiras o barras de **LEDs RGB** son unidades que disponen de un numero determinado de LEDs RGB que pueden ser direccionados de manera independiente y también les podemos poner el color que queramos mediante la codificación RGB *(num1,num2,num3).*

Uno de los mas importantes fabricantes de estos dispositivos es **Neopixel** que nos ofrece también las librerías para **MicroPython** que permiten su manejo mediante distintas funciones. En la siguiente imagen se muestra el dispositivo que vamos a usar en esta sección de nuestro libro.



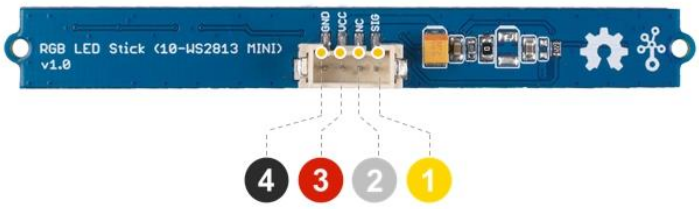
Grove - RGB LED Stick (10 - WS2813 Mini)

Esta barra de LEDs integra **10 LED RGB** a todo color en este palo, con un solo pin de señal puede controlar los 10 LED fácilmente. Todos los LED son **WS2813** Mini, que es un control inteligente y LED de alto costo. Además, el WS2813 admite la transmisión continua de punto de interrupción de señal, lo que significa que puede continuar usando otros leds con un led roto.

Mientras no se rompan dos o más LED adyacentes, los LED restantes podrán funcionar normalmente. Puedes usar este pequeño palo para crear cientos y miles de efectos de luz. Esperamos que te guste trabajar con este dispositivo.

Listo para usar con microcontroladores con entrada digital. Este producto también tiene una interfaz compatible con Plug-and-play grove de diseño modular compacto.

Pin Out del dispoitivo:



4 GND: connect this module to the system GND
3 VCC: you can use 5V or 3.3V for this module
2 NC: not connected
1 SIG: control signal input

Librería neopixel

Vamos a usar la librería neopixel que esta escrita para controlar el dispositivo de LEDs **WS2812 de Neopixel**

Esta clase almacena datos de píxeles para una tira de LED WS2812 conectada a un pin. El La aplicación debes establecer datos de píxeles y luego llamar a **NeoPixel.write()** cuando esté listo para actualizar la tira.

Constructores de la librería

***class*neopixel.NeoPixel(*pin, n, *, bpp=3, timing=1*)**

Construye un objeto NeoPixel. Los parámetros son:

- *PIN* es una máquina. Instancia de pin.
- *n* es el número de LEDs en la tira.
- *bpp* es 3 para LED RGB y 4 para LED RGBW.
- *el tiempo* es 0 para 400KHz y 1 para LED de 800kHz (la mayoría son 800kHz).

Métodos de acceso a píxeles¶

NeoPixel.fill(*pixel*)

Establece el valor de todos los píxeles en el valor de *píxel* especificado (es decir, un Tupla RGB/RGBW).

NeoPixel.__len__()

Devuelve el número de LED de la tira.

NeoPixel.__setitem__(*index,val*)

Establezca el píxel en el *índice* en el valor, que es una tupla RGB/RGBW.

NeoPixel.__getitem__(*index*)

Devuelve el píxel en *el índice* como una tupla RGB/RGBW.

Métodos de salida¶

NeoPixel.write()

Escribe los datos de píxeles actuales en la tira.

9.1. Trabajando con Neopixel

Objetivo

Vamos a usar los dos leds de la tarjeta MAKER PI RP2040 de **Cytron** para empezar a experimentar con los LEDs RGB mediante la librería “neopixel”

Funcionalidad

En la tarjeta tenemos un dispositivo RGB con dos LEDs que se gobiernan desde el pin GP18.

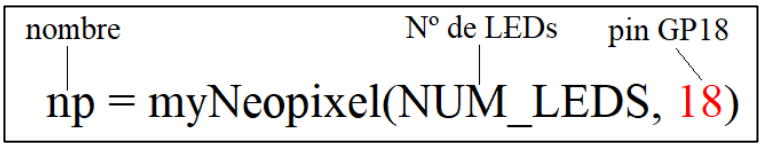
Los dispositivos RGB se programan designando el pin en el que se conecta y el numero de LEDs que tienen. En nuestro caso tenemos dos LEDs conectados al pin GP18. Estos LEDs se direccionan con los valores “0” y “1”

Programa

1. El programa comienza con la importación de las librerías en este caso se añade la librería “neopixel”.

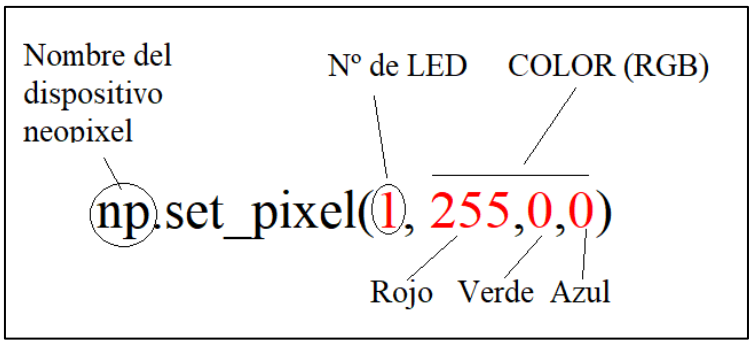
```
import time
from machine import Pin
from neopixel import myNeopixel
```

2. Definimos el número de leds del dispositivo con la variable NUM_LEDS=2



3. Definimos el objeto np como un objeto neopixel y fijamos el brillo al valor 200

```
np = myNeopixel(NUM_LEDS, 18)
np.brightness(200) #brillo: 0 ~ 255
```



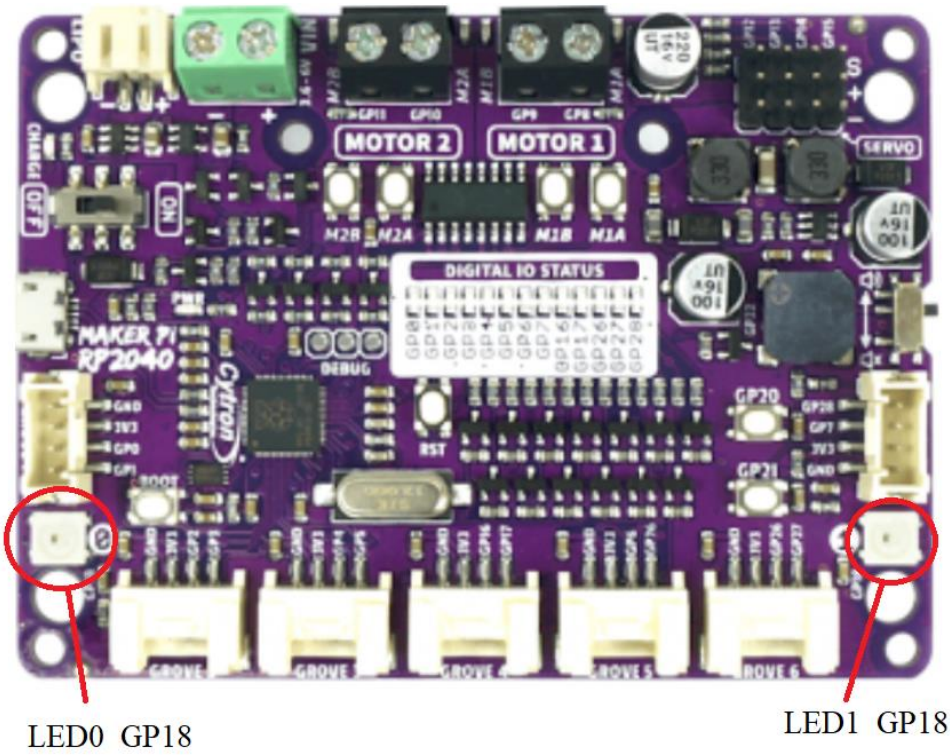
4. Ponemos el LED1 en color rojo (255,0,0) y el LED0 en azul (0,0,255)

5. Para finalizar mostraremos lo que hemos programado. Vemos cada led con su color: LED1=Rojo, LED0=azul

```
#Neopixel Basico en la tarjeta MAKER PI RP2040
import time
from machine import Pin
from neopixel import myNeopixel

NUM_LEDS = 2
#El pin GP18 es el que incluye dos RGBs en la tarjeta
np = myNeopixel(NUM_LEDS, 18)
np.brightness(200) #brillo: 0 ~ 255

np.set_pixel(1, 255,0,0)
np.set_pixel(0, 0,0,255)
np.show()
```

Montaje de la practica

La practica no necesita, en este caso, ningún elemento a conectar. Estamos usando los dos LEDs RGB

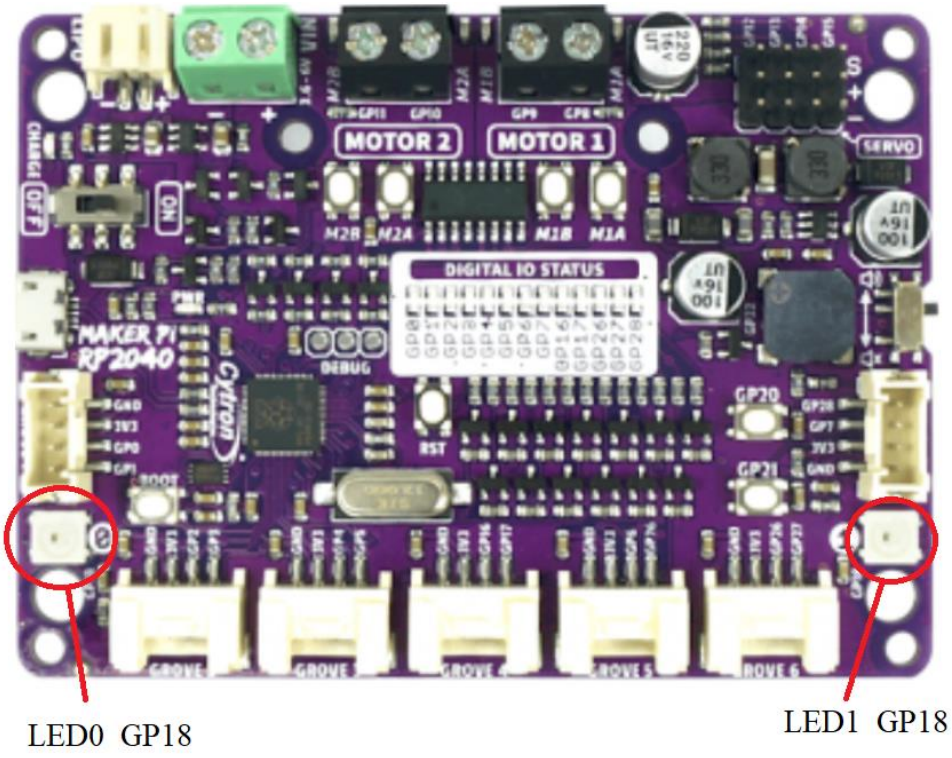


Análisis y propuesta de actividades

1. Te propongo que realices el siguiente montaje. Se trata de hacer un doble blink con los LEDs 0 y 1 de la tarjeta. Encenderemos el 1 de color “verde” y el 0 de color “azul”. Queremos que el tiempo de encendido/apagado sea de 100 ms.

```
#Neopixel Blink en la tarjeta MAKER PI RP2040
import time
from machine import Pin
from neopixel import myNeopixel

NUM_LEDS = 2
#El pin GP18 es el que incluye dos RGBs en la tarjeta
np = myNeopixel(NUM_LEDS, 18)
np.brightness(255) #brillo: 0 ~ 255

while True:
    np.set_pixel(1, 0,255,0) #LED1 en Rojo
    np.set_pixel(0, 0,0,0) #Apaga LED0
    np.show() #Muestra leds programados
    time.sleep(0.1) #Espera 500 ms
    np.set_pixel(1, 0,0,0) #Apaga LED1
    np.set_pixel(0, 0,0,255)#Muestra LED0 3n Azul
    np.show()
    time.sleep(0.1)
```

Este es el montaje.



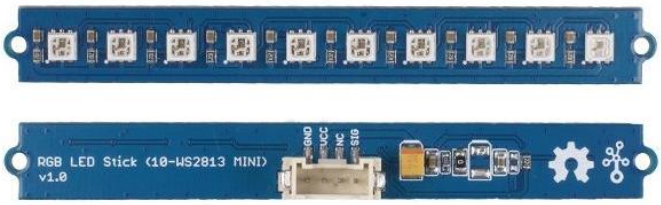
9.2. Secuencia de iluminación

Objetivo

Con este ejemplo vas a utilizar un dispositivo externo **Neopixel RGB LED de 10 LEDs**

Funcionalidad

Nuestro dispositivo, como y hemos dicho en la práctica anterior, se gobierna con un solo pin, en nuestro caso el GP3

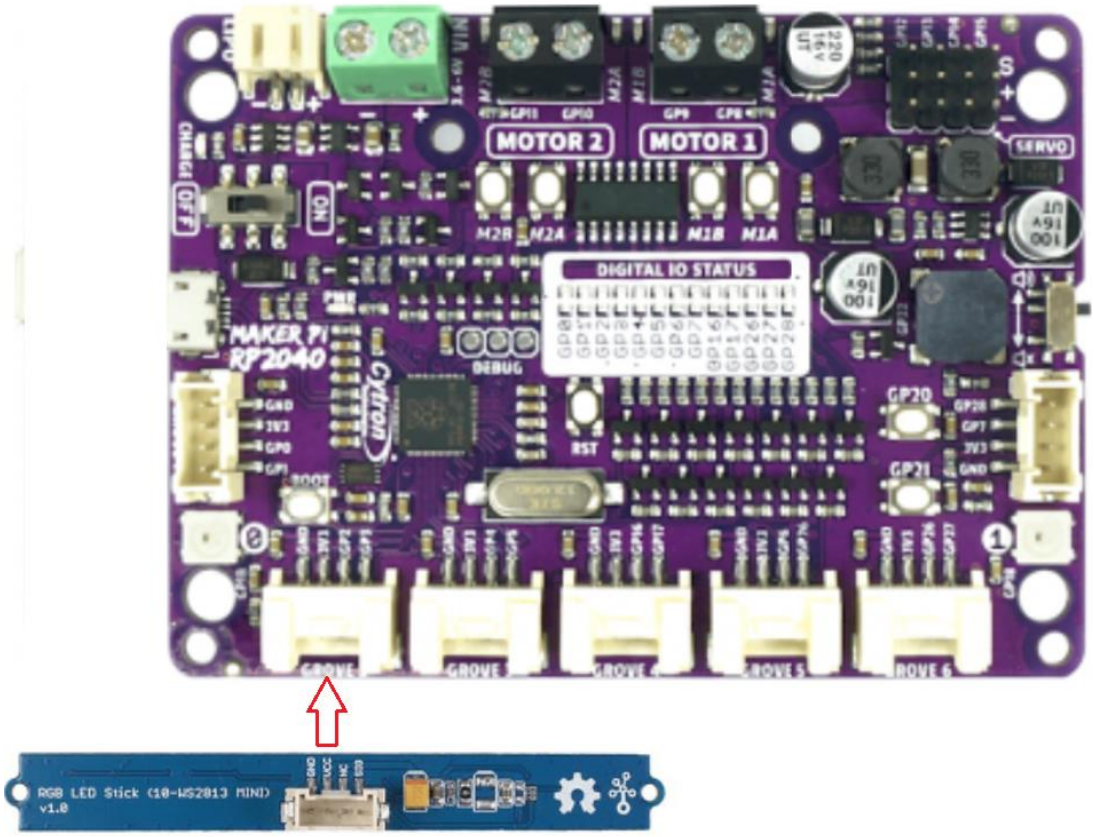


Programa

```
#Secuencia de iluminación NEOPIXEL de 10 LEDs
import time
from machine import Pin
from neopixel import myNeopixel

NUM_LEDS = 10
np = myNeopixel(NUM_LEDS, 3) #definimos el objeto RGB

#Definición de los colores
red = (255, 0, 0) # Color Rojo
green = (0, 255, 0) #Color Verde
blue = (0, 0, 255) #Color Azul
white = (255, 255, 255) #Color Blanco
close = (0, 0, 0) #Color negro (apagado)

# Lista de colores
COLORS = [red, green, blue, white, close]
#Fijación del nivel de brillo de los LEDs
np.brightness(100) #Brillo: 0 ~ 255
while True:
    for color in COLORS:
        np.fill(color[0], color[1], color[2])
        np.show()
        time.sleep(0.5)
```

Montaje de la practica

La imagen siguiente muestra la conexión del dispositivo **Neopixel** a través del puerto Grove haciendo coincidir el pin **GP3** con el terminal **Sig** del dispositivo **Neopixel**.



Análisis y propuesta de actividades

Una vez montado el ejercicio puedes comprobar el funcionamiento y realizar modificaciones en el programa, tales como:

- Cambiar los colores de la secuencia
- Cambiar el tiempo de cada color en la secuencia
- Modificar el brillo de los LEDs

9.3. Arco de Iris

Objetivo

Realizar un programa en el que la secuencia de encendido de cada led y su color emulen el efecto de Arco Iris usando el dispositivo Neopixel 10 LEDs

Funcionalidad

En este montaje lo que haremos será crear una secuencia de luz que nos provoque la sensación de la contemplación de una Arco Iris usando los 10 LEDs de nuestro dispositivo.

Programa

1. En el programa lo primero que haremos será cargar las librerías

```
from machine import Pin
from neopixel import myNeopixel
import time
```

2. Seguidamente definimos el objeto **np** definiendo el nº de LEDs y el pin GP de conexión

```
np = myNeopixel(10, 3)
```

3. Definimos los colores base en valor 0

```
rojo = 0  #rojo
verde = 0 #verde
azul = 0  #azul
```

4. Creamos una función a la que llamaremos “iris” cuyo parámetro de entrada es “pos”

```
def iris(pos):
    global rojo, verde, azul
    Posicion = pos % 255
    if Posicion < 85:
        rojo = (255-Posicion*3)
        verde = (Posicion*3)
        azul=0
    elif Posicion >= 85 and Posicion < 170:
        Posicion -= 85;
        rojo = 0
        verde = (255 - Posicion*3)
        azul = (Posicion*3)
    else :
        Posicion -= 170;
        rojo = (Posicion*3)
```

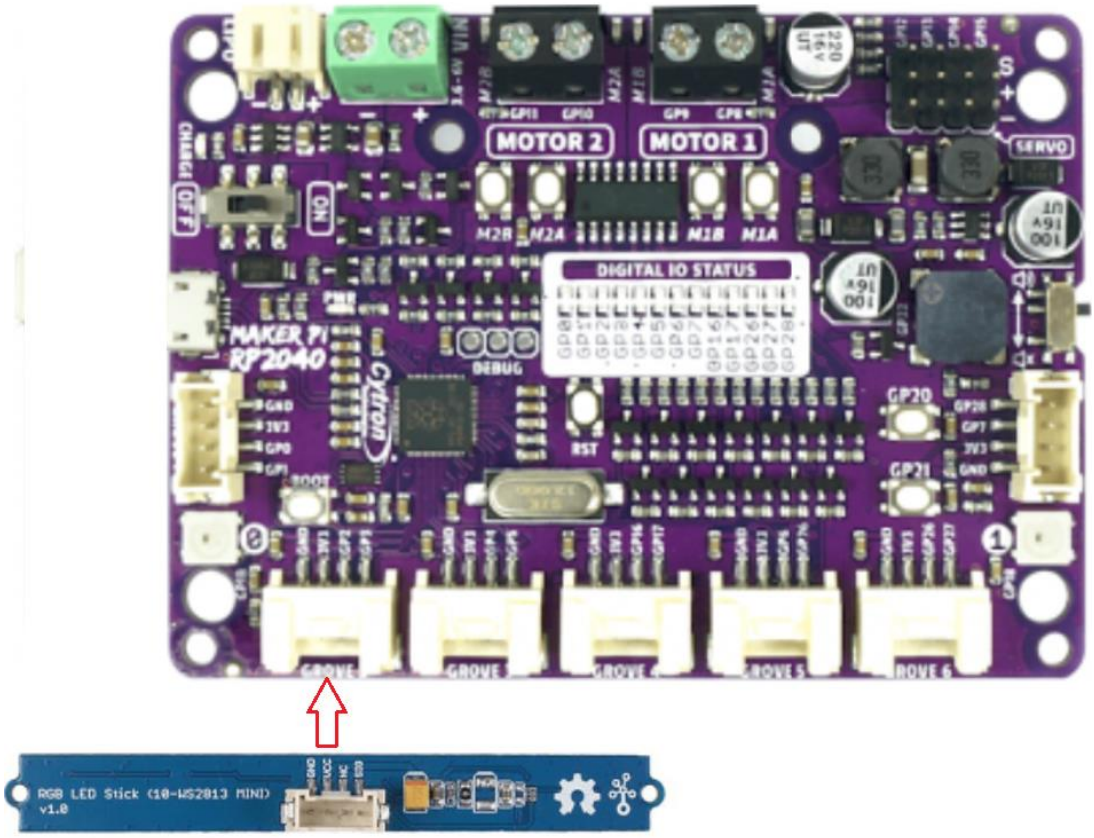
```
        verde = 0
        azul = (255-Posicion*3)
```

5. Finalmente en la parte del bucle **while True** se encadenan dos bucles anidados tipo **for**

```
#Efecto Arco de Iris con dispositivo NEOPIXEL de 10 LEDs
from machine import Pin
from neopixel import myNeopixel
import time

#Defincion del Objeto Neopixel
#(nº de leds, pin de conexion)
np = myNeopixel(10, 3)

rojo = 0  #rojo
verde = 0 #verde
azul = 0  #azul

def iris(pos):
    global rojo, verde, azul
    Posicion = pos % 255
    if Posicion < 85:
        rojo = (255-Posicion*3)
        verde = (Posicion*3)
        azul=0
    elif Posicion >= 85 and Posicion < 170:
        Posicion -= 85;
        rojo = 0
        verde = (255 - Posicion*3)
        azul = (Posicion*3)
    else :
        Posicion -= 170;
        rojo = (Posicion*3)
        verde = 0
        azul = (255-Posicion*3)

np.brightness(20)
while True:
    for i in range(0, 255):
        for j in range(0, 10):
            iris(i + j*255 // 8)
            np.set_pixel(j, rojo, verde, azul)
        np.show()
        time.sleep_ms(1)
```

Montaje de la practica

El montaje es el de la siguiente imagen. Una vez cargado el programa se procederá a evaluar el comportamiento del circuito implementado



Análisis y propuesta de actividades

Una vez montado el ejercicio puedes comprobar el funcionamiento y realizar modificaciones en el programa, tales como:

- Cambiar el brillo de los LEDs
- Cambiar el tiempo de cada color en la secuencia

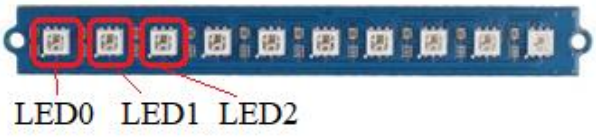
9.4. Semáforo

Objetivo

Disponiendo de un dispositivo **Neopixel RGB** resulta muy sencillo poder simular con el un semáforo. En esta práctica lo vamos a hacer

Funcionalidad

Utilizamos el dispositivo Neopixel de la practica anterior.



Utilizaremos los tres primeros LEDs del dispositivo.

- LED0 Lampara LED Rojo
- LED1 Lámpara LED Ámbar
- LED2 Lámpara LED Verde

Los tiempos entre cada fase serán de 500 ms.

Programa

1. En primer lugar, como siempre, cargamos las librerías.

```
import time
from machine import Pin
from neopixel import myNeopixel
```

2. Seguidamente definimos nuestro objeto Neopixel conectándolo en el pin GP3

```
semaforo = myNeopixel(NUM_LEDS, 3)
```

3. Fijamos el brillo y aseguramos que los tres leds, al inicio del programa, esten apagados

```
semaforo.brightness(200) #brillo: 0 ~ 255
semaforo.set_pixel(0, 0,0,0)
semaforo.set_pixel(1, 0,0,0)
semaforo.set_pixel(2, 0,0,0)
```

4. Finalmente establecemos la secuencia de activación de los LEDs cada uno en su color adecuado y con un tiempo de retardo de 500 ms.

```
semaforo.set_pixel(0, 255,0,0) #LED1 en Rojo
semaforo.set_pixel(2, 0,0,0) #LED2 Apagado
```

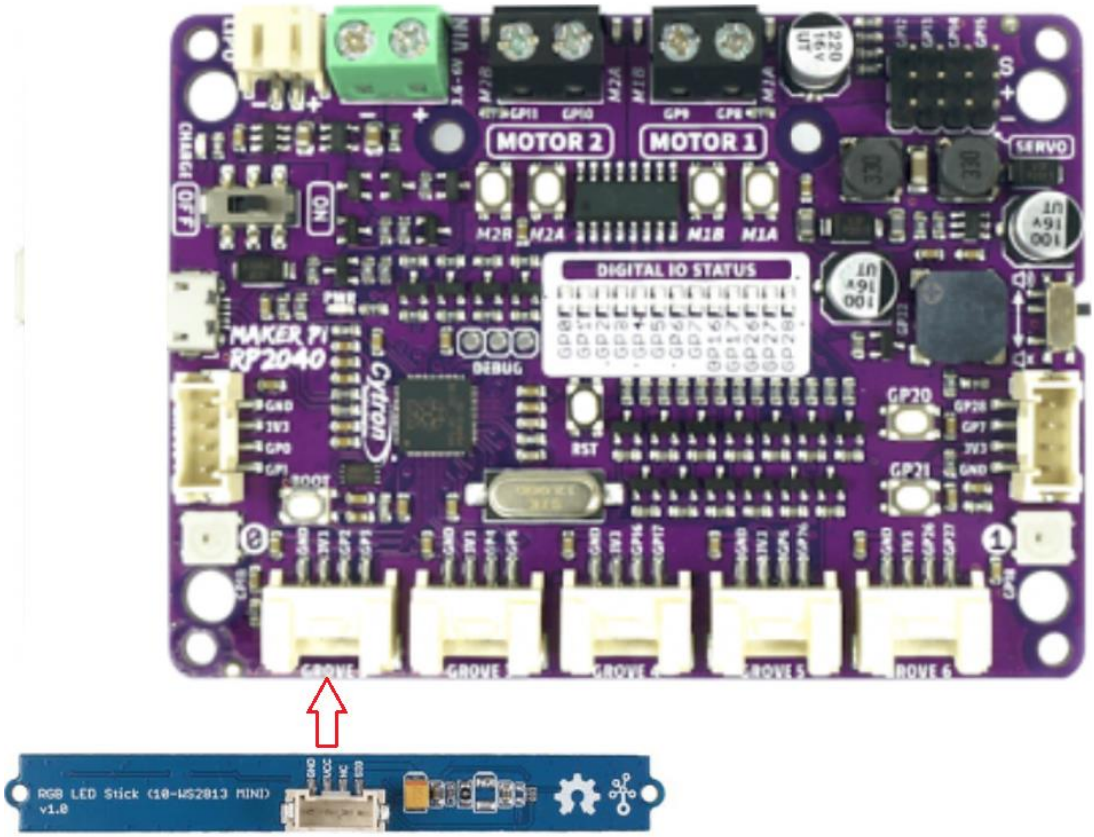
```
semaforo.show()
time.sleep(0.5)
semaforo.set_pixel(1, 100,100,0) #LED1 Amarillo
semaforo.set_pixel(0, 0,0,0) #LED0 Apagado
semaforo.show()
time.sleep(0.5)
semaforo.set_pixel(2, 0,255,0) #LED2 Verde
semaforo.set_pixel(1, 0,0,0) #LED1 en Apagado
semaforo.show()
time.sleep(0.5)
```

Este es el programa completo

```
"""
Neopixel Semaforo dispositivo 10 LEDs RGB
en la tarjeta MAKER PI RP2040
"""
import time
from machine import Pin
from neopixel import myNeopixel

NUM_LEDS = 10
#El pin GP18 es el que incluye dos RGBs en la tarjeta
semaforo = myNeopixel(NUM_LEDS, 3)
semaforo.brightness(200) #brillo: 0 ~ 255
semaforo.set_pixel(0, 0,0,0)
semaforo.set_pixel(1, 0,0,0)
semaforo.set_pixel(2, 0,0,0)

while True:
    semaforo.set_pixel(0, 255,0,0) #LED1 en Rojo
    semaforo.set_pixel(2, 0,0,0) #LED2 Apagado
    semaforo.show()
    time.sleep(0.5)
    semaforo.set_pixel(1, 100,100,0) #LED1 Amarillo
    semaforo.set_pixel(0, 0,0,0) #LED0 Apagado
    semaforo.show()
    time.sleep(0.5)
    semaforo.set_pixel(2, 0,255,0) #LED2 Verde
    semaforo.set_pixel(1, 0,0,0) #LED1 en Apagado
    semaforo.show()
    time.sleep(0.5)
```

Montaje de la practica

Este es el montaje de la practica



Análisis y propuesta de actividades

Te proponemos que modifiques:

- Los LEDs a usar (LED3, LED4,LED5)
- El pin designado de la tarjeta para el gobierno de la unidad Neopixel
- Los tiempos de cada secuencia.
- Crear una función que ejecute una secuencia

10. Usando sensores

En este capítulo vamos a trabajar con sensores. Son muy numerosos los sensores que se ofrecen en el mercado de los componentes electronicos.

Trabajaremos en la mayor parte de las practicas son sensores compatibles con conexión tipo Grove que como sabemos failita mucho el trabajo de conexionado con cualqueiera de las dos tarjetas con las que estamos trabajando de la firma **Cytron**.

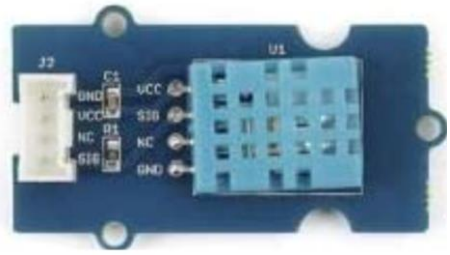
10.1. Medida de Temperatura y Humedad

Objetivo

Funcionalidad

Sensor de temperatura

Seeed Studio ofrece una gran colección de sensores.



El DHT11/DHT22 es un sensor que mide la humedad relativa y el sensor de temperatura. Proporciona una salida digital calibrada con un protocolo de 1 hilo. Ambos sensores son baratos. Son bastante similares entre sí con algunas diferencias de especificaciones.

DHT22 es casi similar a la DHT11, pero la primera mide la temperatura y la humedad con mayor precisión y admite un rango más amplio.

	DHT22	**DHT11**
Temperatura	-40 a 80 ºC +/- 0,5ºC	0 a 50 ºC +/-2 ºC
Humedad	0 a 100% +/- 2%	20 a 90% +/- 5%
Resolución	Humedad: 0.1%Temperatura: 0.1ºC	Humedad: 1%Temperatura: 1ºC
Voltaje de funcionamiento	3- 6 V DC (directamente alimentable desde Raspberry Pi Pico)	3-5.5 V DC (directamente alimentable desde Raspberry Pi Pico)
Tiempo de muestreo	2 segundos	1 segundo

Clasificación actual	$4 a $10	$1 a $5
Tipo de datos de salida	flotar	Int
Pinout	4 pines (igual que DHT11)	4 pines (igual que DHT22)
Precio	5	2

Como puede ver en la tabla de comparación anterior, DHT22 ofrece un rango de temperatura y resolución más amplios para la temperatura y la humedad. Pero es más caro que DHT11. Sin embargo, DHT11 tiene un mejor período de muestreo. Además, el rango de voltaje de funcionamiento para ambos sensores es casi y podemos alimentar directamente estos sensores desde los pines de alimentación de Raspberry Pi Pico.

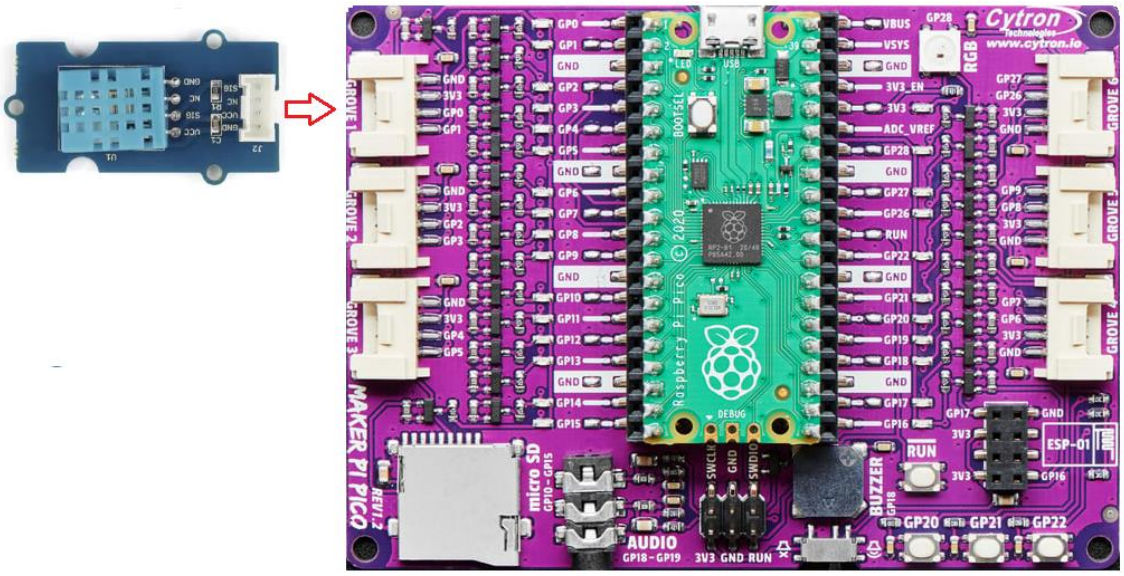
Independientemente de las diferencias anteriores, ambos sensores DHT tienen el mismo principio de funcionamiento y el mismo pinout. Podemos usar el mismo script de **MicroPython** para leer lecturas de temperatura y humedad seleccionando el tipo DHT dentro del código.

Los sensores DHT están pre calibrados. Podemos conectarlos directamente con Raspberry Pi Pico para obtener la lectura de salida del sensor. Están compuestos internamente por un sensor de detección de humedad y un termistor. Estos dos componentes miden la humedad y la temperatura.

Programa

```
'''
 * Medida de Temperatura y Humedad
 * Con MAKER PI PICO de Cytron
'''
from machine import Pin
from time import sleep
import dht

pin = Pin(1, machine.Pin.OUT, machine.Pin.PULL_DOWN)
sensor = dht.DHT11(pin)

while True:
    print("temperatura : {} °C  humedad : {} %".format(sensor.temperature,
sensor.humidity))
    sleep(1.5) #Este valor no puede ser menor dado que el sensor no trabaja bien
```

Montaje de la practica



Análisis y propuesta de actividades

```
"""
Medida de temperatura mediante un sensor resistivo
Colocado en un canal de entrada analógica
"""
from machine import Pin, ADC
import time
import math

#Set ADC
adc=ADC(27)

try:
    while True:
        adcValue = adc.read_u16()
        voltage = adcValue / 65535.0 * 3.3
#El valor 65535 es muy critico y se puede generar una divison por cero
# Si ocurre esto se puede subir a 65635
        Rt = 10 * voltage / (3.3-voltage)
        tempK = (1 / (1 / (273.15+25) + (math.log(Rt/10)) / 3950))
        tempC = int(tempK - 273.15)
        print("ADC value:", adcValue, "  Voltage: %0.2f"%voltage,
              "  Temperature: " + str(tempC) + "ºC")
        time.sleep(1)
except:
    pass
```

10.2. Termostato

Objetivo

Con esta práctica vamos a implementar un termostato que controlará dos equipos de salida, un calefactor y un refrigerador.

Funcionalidad

Estableceremos dos consignas para la activación de cada uno de los dos elementos actuadores, calefactor y refrigerador.

- Cc Temperatura de activación de calefacción =10º
- Cr Temperatura de activación de refrigeración =25

Las direcciones delos pines de la tarjeta implicadas serán las siguientes:

- GP3 Pin del relé de activación del calefactor
- GP5 Pin del relé de activación del refrigerador
- GP1 Pin de conexión del sensor DHT11 de temperatura

Relé1

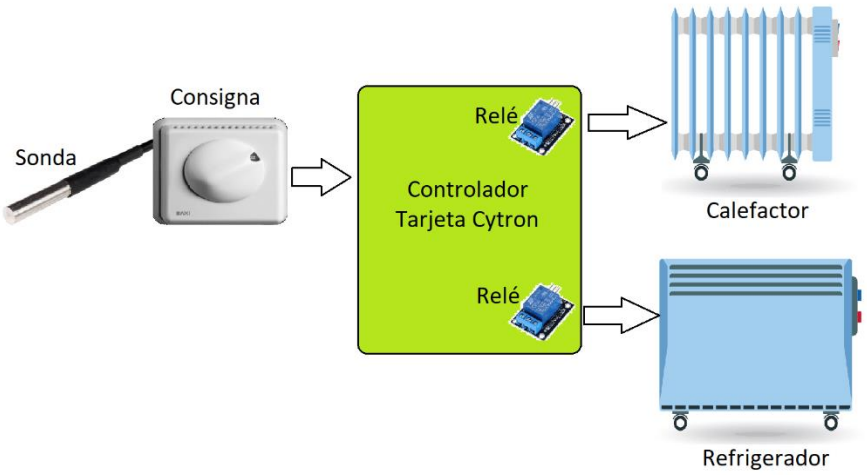
Relé2

Sensor DHT11

Nuestro sistema ha de diseñarse de acuerdo a la siguiente tabla de funcionamiento.

- Si temperatura < de Cc — Activa Relé Calefactor Relé 1
- Si temperatura > de Cr — Activa Relé Refrigeración Relé 2

Los estados de los elementos de salida queremos que se escriban en la consola de **Thonny**



Programa

1. En el programa lo primero que haremos será cargar las librerías

```
from machine import Pin
from utime
import dht
```

2. Seguidamente definimos los parámetros Cc y Cr

```
Cc= 10 #Consigna Calefacción
Cr=25  #Consigna Refigerador
```

3. Definimos los pines que se usarán

```
calefactor=Pin(3,Pin.OUT)
refrigerador=Pin(5,Pin.OUT)
pin = machine.Pin(1, machine.Pin.OUT, machine.Pin.PULL_DOWN)
sensor = dht.DHT11(pin)
```

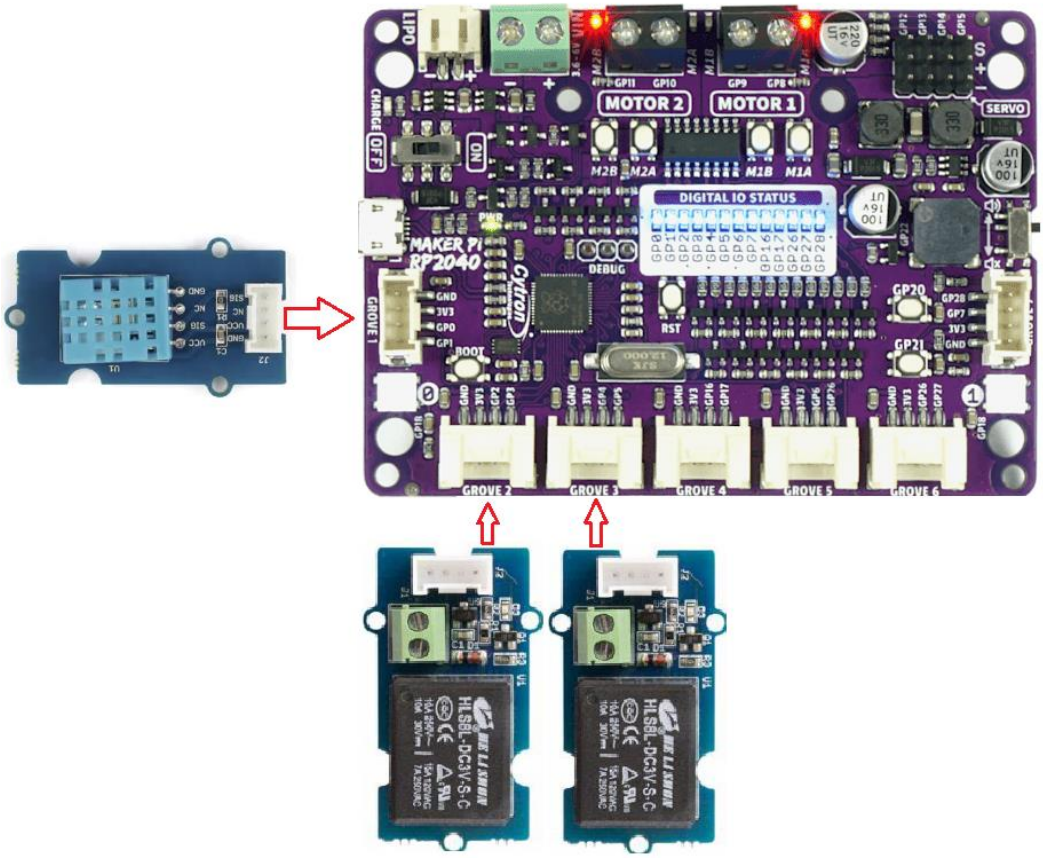
4. Establecemos el contenido del bucle **while**

```
    if sensor.temperature < Cc:
        calefactor.value(1)   # Se activa Calefacción
        print('Calefacción Encendido')
    else:
        calefactor.value(0)   # Se desactiva Calefacción
        print('Calefacción Apagada')
    if sensor.temperature > Cr:
        refrigerador.value(1)   # Se activa Refrigeración
        print('Refrigerador Encendido')
    else:
        refrigerador.value(0)   # Se desactiva Refrigeración
        print('Refrigerador Apagado')
```

Programa completo

```
'''
 * Termostato con gobierno sobre calefactor
 * y refrigerador
'''
from machine  import Pin
import utime
import dht

Cc= 10 #Consigna Calefacción
Cr=25  #Consigna Refigerador

calefactor=Pin(3,Pin.OUT)
refrigerador=Pin(5,Pin.OUT)
pin = machine.Pin(1, machine.Pin.OUT,
machine.Pin.PULL_DOWN)
sensor = dht.DHT11(pin)

while True:
    if sensor.temperature < Cc:
        calefactor.value(1)   # Se activa Calefacción
        print('Calefacción Encendido')
    else:
        calefactor.value(0)   # Se desactiva Calefacción
        print('Calefacción Apagada')
    if sensor.temperature > Cr:
        refrigerador.value(1)   # Se activa Refrigeración
        print('Refrigerador Encendido')
    else:
        refrigerador.value(0)   # Se desactiva Refrigeración
        print('Refrigerador Apagado')
```

Montaje de la practica



10.3. Fotoresistor: Medida de Luz

Objetivo

Trabajaremos con un sensor de medida de luz con el que realizaremos una practica

Funcionalidad

Sobre el Sensor de Luz Grove

El módulo **Grove - Light Sensor** incorpora una resistencia dependiente de la luz (LDR). Por lo general, la resistencia del LDR o fotorresistencia disminuirá cuando aumente la intensidad de la luz ambiental. Esto significa que la señal de salida de este módulo será ALTA en luz brillante y BAJA en la oscuridad.

Funciones

- Módulo sensor de luz fácil de usar
- La resistencia disminuye a medida que aumenta la luminancia
  - Baja resistencia (con luz brillante) dispara una señal ALTA hacia el módulo de salida
  - Alta resistencia (en la oscuridad) dispara una señal BAJA hacia el módulo de salida
- Se integra fácilmente con los módulos lógicos en el lado de entrada de los circuitos Grove
- Utiliza cables Grove estándar de 4 pines para conectarse a otros módulos Grove.

Características técnicas

- Voltaje: 3-5V
- Corriente de alimentación: 0.5-3mA
- Resistencia a la luz: 20KΩ
- Resistencia oscura: 1MΩ
- Tiempo de respuesta: 20-30 segundos
- Longitud de onda máxima: 540 nm
- Temperatura ambiente: -30 ~ 70 °C
- LDR utilizado: GL5528

Nuestro programa: Se trata de leer el valor que entrega el sensor de luz colocado en el pin **GP27** y este valor se imprimirá en la consola de **Thonny** y además se lleva a la salida **GP7** definida como salida PWM

El valor que imprimimos lo escalamos para que se refleje en la escala de **0 a 100**, por lo que necesitamos implementar la función **map** para realizar el reescalado del valor leído del puerto **GP27** que como sabemos adopta los valores comprendidos en el entorno de **0 a 65530**

El valor que sacamos por la salida PWM GP7 será el valor recogido directamente del puerto GP27 es decir de 0 a 65530

Programa

1. En el programa lo primero que haremos será cargar las librerías

```
from machine import Pin, ADC, PWM
import time
```

2. Seguidamente definimos los pines a usar

```
#Definición de pines adc y salidaPwm
adc = ADC(27)
salidaPwm = PWM(Pin(7))
salidaPwm.freq(10000)
```

3. Definimos la función de mapeado.

```
# Función mapeado
def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)
```

4. Establecemos la función de control de error

```
try:

……

except:
   salidaPwm.deinit()
```

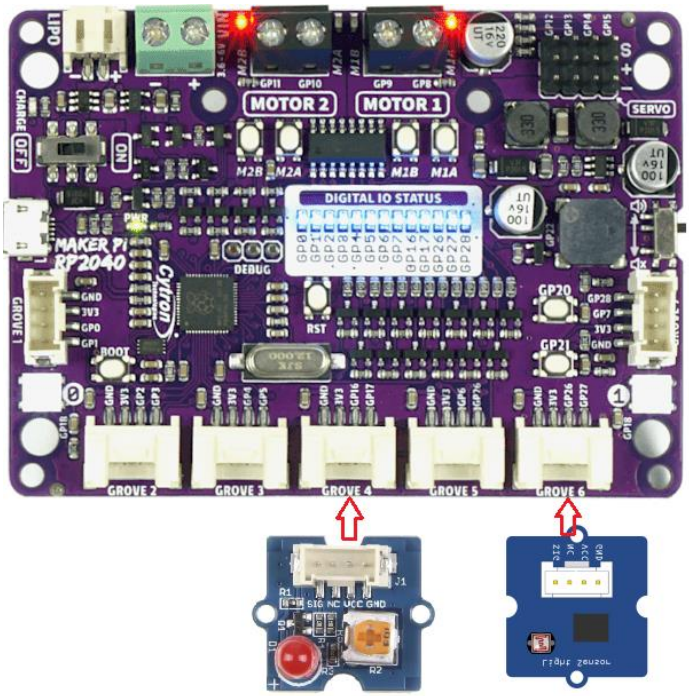
5. Escribir el contenido dentro de la función while

```
        valor=adc.read_u16()
        luz=int(map(valor,0,65530,0,100))
        salidaPwm.duty_u16(valor)
        time.sleep(0.1)
        print("Valor Leido", valor,"LUZ (valor mapeado)",luz)
```

El programa completo será el siguiente

```
#Test de una sonda de Luz (fotorresistencia)
from machine import Pin, ADC, PWM
import time
#Definición de pines adc y salidaPwm
adc = ADC(27)
salidaPwm = PWM(Pin(7))
salidaPwm.freq(10000)
# Funcion mapeado
def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)
try:
   while True:
      valor=adc.read_u16()
      luz=int(map(valor,0,65530,0,100))
      salidaPwm.duty_u16(valor)
      time.sleep(0.1)
      print("Valor Leido", valor,"LUZ (valor mapeado)",luz)
except:
   salidaPwm.deinit()
```

Montaje de la practica



11. Detector de Infrarrojos

11.1. Infrarrojos Básico

Objetivo

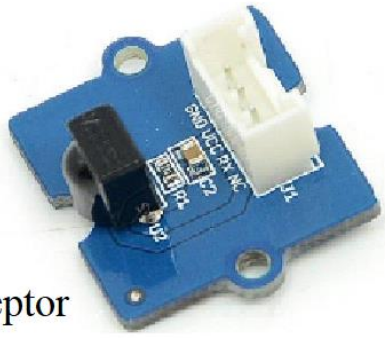
Con esta práctica vamos a estudiar el comportamiento de un sensor de infrarrojos.

Funcionalidad

El mando a distancia es una función muy importante en los desarrollos electrónicos. Son numerosos los sistemas que usamos que incorporan un mando para poder activar un determinado aparato.

Para trabajar este tema vamos a recurrir a dos dispositivos: Mando Emisor de infrarrojos y Receptor de señal.

Emisor


Receptor

La señal que recibe el receptor de IR esta codificada en hexadecimal de 32 bits. Los distintos fabricantes de sistemas de mando con infrarrojos adoptan sus propios códigos, por lo cual, es preciso adaptar nuestro sistema a los códigos del mando que manejamos.

Programa

1. El programa que vamos a escribir es muy sencillo básicamente lo que hacemos es cargar la librería "**irrecvdata**" y de esta la función "**irGetCMD**"

```
from irrecvdata import irGetCMD
```

2. Definimos nuestro dispositivo creando el objeto al que llamamos **recvPin** que se asociara al pin **irGetCMD**(3)

```
recvPin = irGetCMD(3)
```

3. El sensor será leído en todo momento mediante el bucle **while True**, el valor se recoge en la variable **irValue** que será lo que se lee en el pin **GP3**

```
irValue = recvPin.ir_read()
```

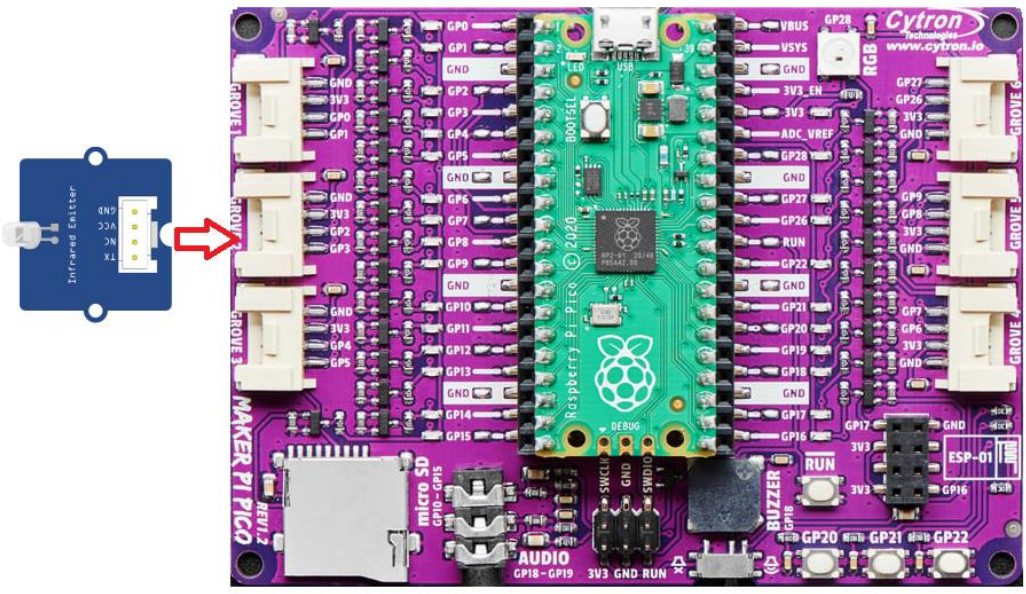
4. Si se detecta valor **irValue** adquiere un valor y en ese caso lo imprimimos

```
if irValue:
        print(irValue)
```

Este sería el programa

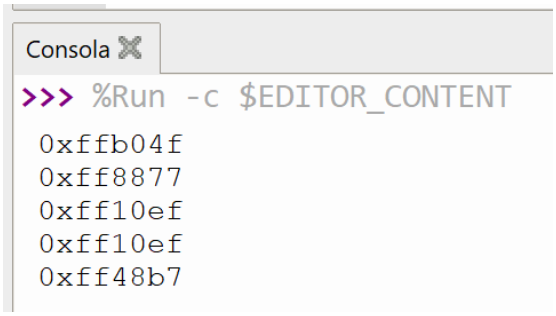
```
# Sensor detector de infrarrojos
from irrecvdata import irGetCMD
# Definicion del pin de conexión
recvPin = irGetCMD(3)
try:
   while True:
      irValue = recvPin.ir_read()
      if irValue:
         print(irValue)
except:
   pass
```

Montaje de la practica



Análisis y propuesta de actividades

Una vez que hemos cargado el programa podemos mandar datos desde el mando a distancia y en la consola de **Thonny** podemos leer los datos leidos correspondientes a cada tecla pulsada.



1. Te propongo que realices una ampliación de tu programa para que los códigos correspondientes a las teclas “0,1,2 y3” se transformen en los valores “0,1,2 y3). Para esto puedes usar condicionales. Queremos también que se imprima el código recibido convertido en número decimal print(int(irValue,16))

```
#Decodificando señal de Infrarrojos
from irrecvdata import irGetCMD
recvPin = irGetCMD(3)

while True:
    irValue = recvPin.ir_read()
    if irValue:
        print(irValue)
        print(int(irValue,16)) # Convierte hexadecimal en decimal
    if irValue == '0xffb04f':   #0
        print(0)
    elif irValue == '0xff08f7': #1
        print(1)
    elif irValue == '0xff8877': #2
        print(2)
    elif irValue == '0xff48b7': #3
        print(3)
```

11.2. Infrarrojos Decodificador

Objetivo

Vamos a implementar un sistema que, mediante un mando a distancia, permite activar y desactivar hasta tres salidas a las que colocamos un LED.

Funcionalidad

Usamos tres salidas GP1, GP3 y GP5 que cambiaremos de estado cada vez que se reciba el código correspondiente a los botones 0,1y2. En cada salida colocaremos un LED

Visualizaremos el estado de cada salida así como el código detectado

Programa

1. En primer lugar cargamos librerías

```
from machine import Pin
import utime
from irrecvdata import irGetCMD
```

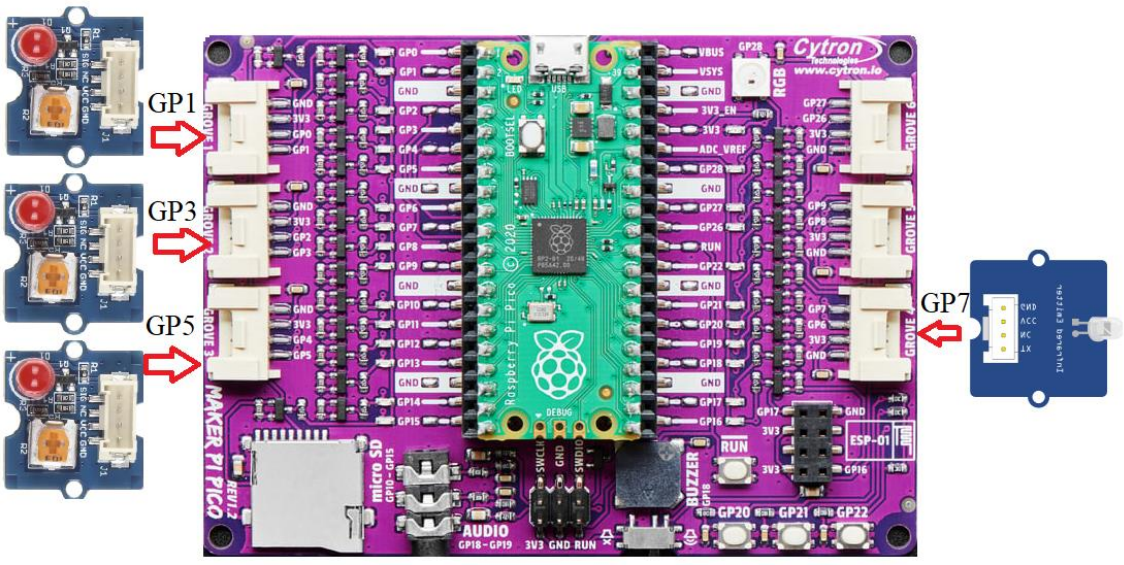
2. Seguidamente definimos los pines que intervienen en nuestra aplicación:

```
recvPin = irGetCMD(7)
led0 = Pin(1, Pin.OUT)
led1 = Pin(3, Pin.OUT)
led2 = Pin(5, Pin.OUT)
```

3. Dentro del bucle lo que haremos será asignar a la variable **irValue** el valor leído en el pin **GP7** en el que esta conectado nuestro detector de IR . Seguidamente imprimimos el valor en hexadecimal de la entrada detectada y seguidamente el estado de las salidas pines **GP1, GP3 y GP5**. Seguidamente establecemos los condicionales que revisan los valores de los códigos correspondientes a la teclas 0,1 y 2 para activar/desactivar las salidas con la función **ledxx.toggle()**

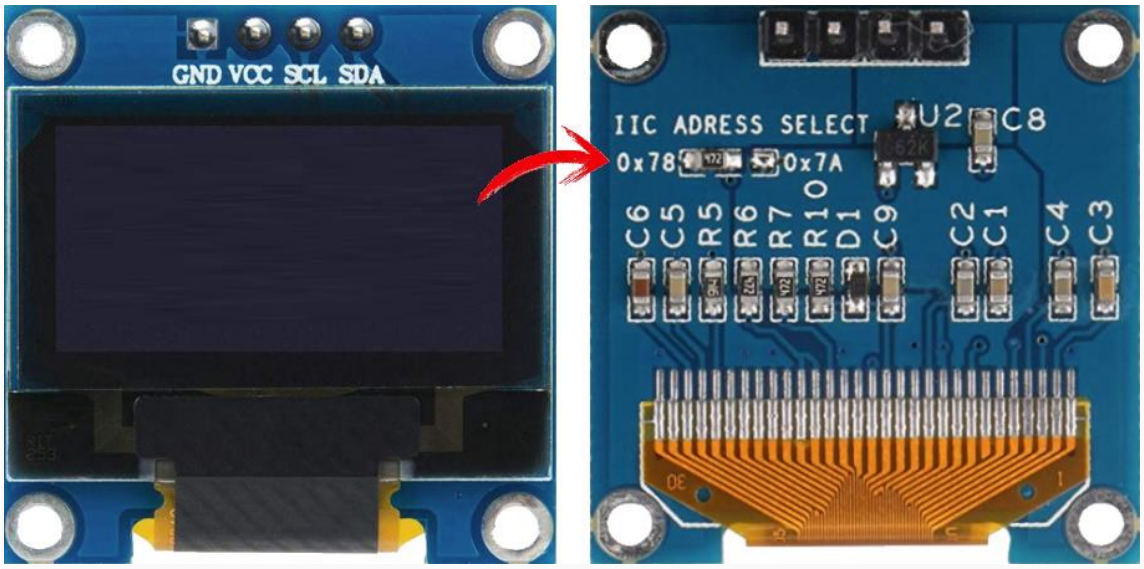
```
#Gobierno de 3 salidas GP0, GP1 y GP2 desde
#un mando a distancia por infrarrojos

from machine import Pin
import utime
from irrecvdata import irGetCMD

recvPin = irGetCMD(7)
led0 = Pin(1, Pin.OUT)
led1 = Pin(3, Pin.OUT)
led2 = Pin(5, Pin.OUT)

while True:
    irValue = recvPin.ir_read()
    print("Codigo detectado=",irValue,"//","Salida0=",
          led0.value(),"Salida1=",led1.value(),
          "Salida1=",led2.value())
    if irValue == '0xffb04f':   #0
        led0.toggle()     # conmuta LED
    elif irValue == '0xff08f7': #1
        led1.toggle()
    elif irValue == '0xff8877': #2
        led2.toggle()
```

Montaje de la practica



Análisis y propuesta de actividades

12. Display OLED



Pantallas OLED SSD1306

Las pantallas OLED (Organic light-emitting diode o diodos emisores de luz orgánicos) son una excelente opción para mostrar datos en los proyectos de electrónica.
Son pantallas formadas por diodos (orgánicos) individuales. que emiten su propia luz. Esto las permite tener un alto contraste, un gran ángulo de visión y un reducido espesor.

MicroPython dispone de una ***librería*** **(no estándar) para pantallas monocromas OLED con el controlador SSD1306 y bus de comunicación I2C o SPI**.

Dado que el tipo más habitual de este modelo de pantallas que podemos encontrar en el mercado, son las **de 128×64 pixels de resolución – 0.96″ (25×14 mm) y bus de comunicaciones el I2C**, se va a desarrollar su uso para poder conocer las opciones de texto y gráficas (puntos, líneas, rectángulos e imágenes) que están disponibles en la *librería*.

LA LIBRERÍA SSD1306

La librería (no estándar) para pantallas monocromas OLED con el controladores SSD1306 y bus de comunicación I2C o SPI para ***MicroPython*** se puede localizar en GitHub:

https://github.com/MicroPython/MicroPython/blob/master/drivers/display/ssd1306.py

Se utilizará grabándola con el el nombre **ssd1306.py** en el directorio **/lib**. Este directorio está destinado *–por defecto–* a guardar los módulos y librerías complementarias de una forma organizada

LA PROGRAMACIÓN

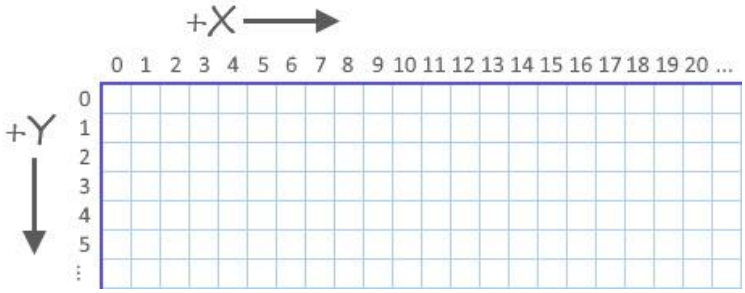
La librería **ssd1306.py** utiliza la *librería* **FrameBuff** de ***MicroPython***, ya que ésta permite crear un búferes eficientes de memoria para trabajar con vistas gráficas simples, por lo que es necesario desarrollar su funcionamiento.

FrameBuff dispone de los *métodos* para dibujar puntos, líneas, rectángulos, imágenes o escribir texto, así como desplazar imágenes por la pantalla.

Un ***framebuffer*** (porción de memoria –*buffer*- reservada para mantener temporalmente una imagen –*frame*-.

Antes de comenzar…

- El **sistema de coordenadas** que utiliza la librería para localizar las representaciones es el cartesiano (unidades en –*píxels*–). Su **eje horizontal** es es el *X* y el **eje vertical** el *Y*.
  - El **origen** (0,0) está en la *esquina superior izquierda.*
  - El **vértice opuesto** en una pantalla de 128×64 *píxels* estará en la coordenada (127×63).



- Las pantallas OLED son monocromas y el **color** se especifica como «0» (*inactivo – color de la pantalla recién inicializada*) o «**1**» (*activo – iluminado*).

Es necesario utilizar .**show()** para mostrar en la pantalla lo que se haya dibujado. Hasta que no se haga no mostrará ningún cambio.

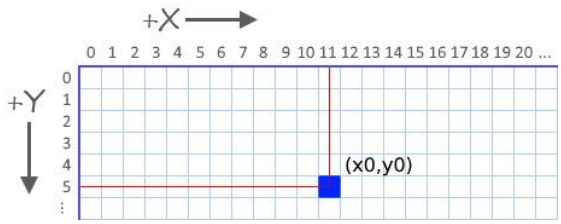
Cargar librerias y crear el objeto oled

```
from machine import Pin, I2C
import ssd1306
from time import sleep
i2c = I2C(-1, scl=Pin(22), sda=Pin(21))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)
```

Instrucciones basicas:

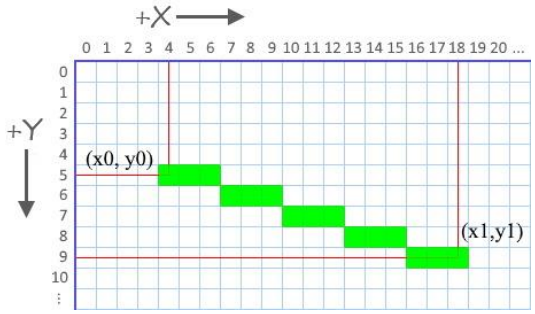
```
oled.fill(1)        # Rellena la pantalla (la ilumina entera)
oled.fill(0)        # Rellena la pantalla (la apaga entera)
oled.show()           # Muestra el resultado
oled.contrast(50)   # Regula el contraste al 50% de la intensidad
oled.inver(1)         # Invierte lo que se va a mostrar
.poweron()       .poweroff() La pantalla se
puede apagar  y encender utilizando .poweron() y poweroff() respectivamente.
```

Objetos que se pueden dibujar en la pantalla:



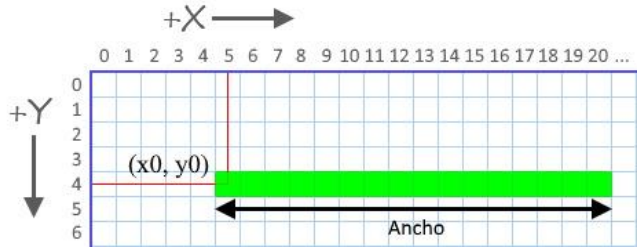
.pixel(x0, y0 [, color])

```
oled.pixel(11, 5, 1)  # Dibuja un píxel iluminado en (5,10)
```



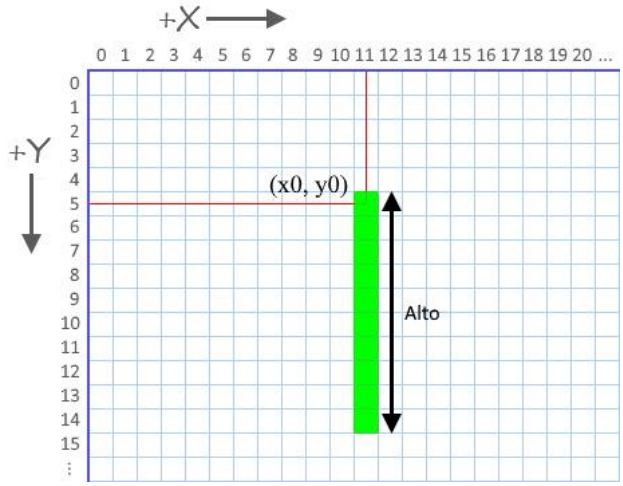
.line(x0, y0, x1, y1, color)

```
oled.line(4, 5, 18, 9, 1)      # Dibuja una línea iluminada. Origen (4, 5) y final (18, 9)
```



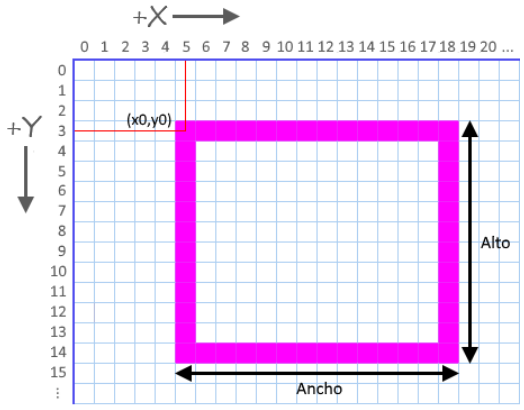
.hline(x0, y0, ancho, color)

```
oled.hline(5, 4, 16, 1)      # Dibuja una línea horizontal iluminada. Origen (5, 4) anchura 16 píxels
```



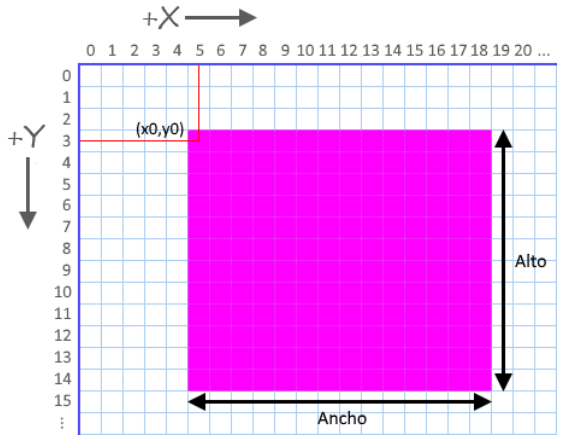
.vline(x0, y0, alto, color)

```
oled.vline(11, 5, 10, 1)      # Dibuja una línea vertical iluminada. Origen (11, 5) altura 10 píxels
```



.rect(x0, y0, ancho, alto, color)

```
oled.rect(5, 3, 14, 12, 1)     # Dibuja un rectángulo iluminado. Origen (5, 3)y anchura x altura 14x12
```



.fill_rect(x0, y0, ancho, alto, color)

```
oled.fill_rect(5, 3, 14, 12, 1)     # Dibuja un rectángulo relleno iluminado. Origen (5, 3)y anchura x altura 14x12 píxels
```

.text(texto, x0, y0 [, color])

```
oled.text("w", 6, 1, 1)              # Escribe "w" iluminado. Origen (6, 1).
oled.text("Hola mundo!", 16, 28, 0)    # Escribe "Hola mundo!" apagado. Origen (16, 28).
```

.scroll(x0, y0)

```
oled.scroll(20,10)
```

12.1. OLED Dibuja Texto

Objetivo

Vamos a probar el funcionamiento del dispositivo OLED 1306

Funcionalidad

En la aplicación manejaremos la librería "ssd1306" que incorpora las funciones necesarias para poder manejar este dispositivo.

La comunicación con el OLED es a través del puerto I2C de Raspberry Pi Pico.

Lo que mandaremos al display para ser mostrado será:

- La letra "w" durante 2 seg. En posición (6,1) y color blanco
- La frase "Hola mundo" en video inverso durante 2 seg.
- La letra "w" desplazándose cinco veces por la pantalla

Programa

El programa se realizará en los siguientes pasos.

1. Cargar las librerías

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf
import utime
```

2. Definimos el ancho y alto en pixels de la pantalla y también el I2C

```
WIDTH  = 128    # Define tamaño de pantalla width
HEIGHT = 64

i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
```

3. Mostramos "w" en pantalla, posición (6,1) y temporizamos 2 seg.

```
oled.text("w", 6, 1, 1)              # Escribe "w" iluminado. Origen (6, 1).
oled.show()                          # Muestra el resultado
utime.sleep(2)                       # Espera 2 segundos
```

4. Ponemos la pantalla en video inverso (blanco) y escribimos "HOLA MUNDO" en la posición (16,28) en color negro. Esperamos 2 seg.

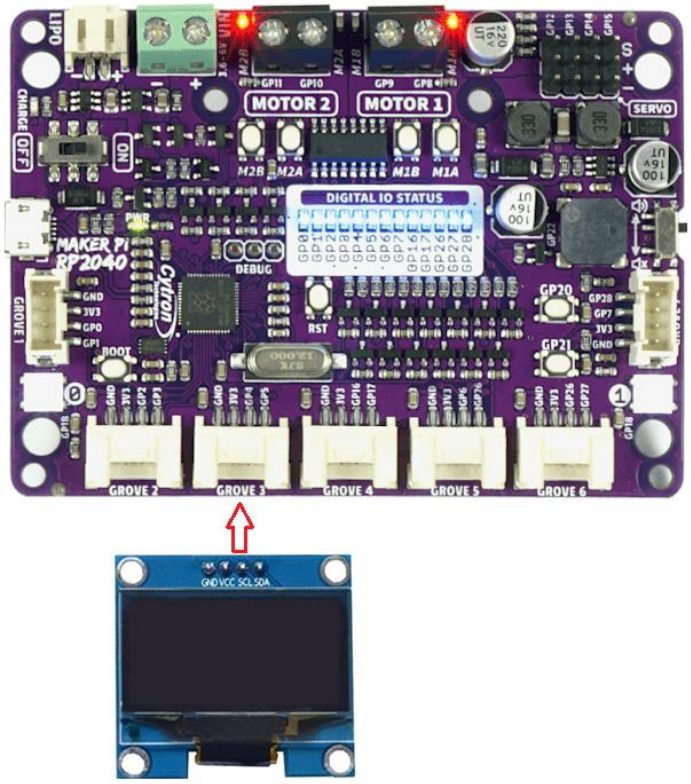
```
oled.fill(1)      # Rellena la pantalla (la ilumina entera)
# Escribe "Hola mundo!" apagado. Origen (16, 28).
oled.text("Hola mundo!", 16, 28, 0)
oled.show()    # Muestra el resultado
utime.sleep(2)
```

5. Muestra “w” y hace un scroll de la letra en la pantalla

```
oled.text("W", 2, 2, 1)    # Escribe "W" iluminado. Origen (2, 2).
oled.show()              # Muestra el resultado
for x in range (0, 5):     # Desplaza la imagen a (20, 10) 5 veces.
   oled.scroll(20,10)
   utime.sleep(0.5)
   oled.show()           # Muestra el resultado
```

```
#Manejo del dispositivo OLED 1306
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf
import utime

WIDTH  = 128    # Define tamaño de pantalla width
HEIGHT = 64

i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
#Dibuja texto
oled.text("w", 6, 1, 1)              # Escribe "w" iluminado. Origen (6, 1).
oled.show()                        # Muestra el resultado
utime.sleep(2)                           # Espera 2 segundos
oled.fill(1)                      # Rellena la pantalla (la ilumina entera)
oled.text("Hola mundo!", 16, 28, 0)    # Escribe "Hola mundo!" apagado. Origen (16, 28).
oled.show()                        # Muestra el resultado
utime.sleep(2)
#Scroll
oled.text("W", 2, 2, 1)    # Escribe "W" iluminado. Origen (2, 2).
oled.show()              # Muestra el resultado
for x in range (0, 5):     # Desplaza la imagen a (20, 10) 5 veces.
   oled.scroll(20,10)
   utime.sleep(0.5)
   oled.show()           # Muestra el resultado
```

Montaje de la practica



Análisis y propuesta de actividades

Este debería ser el aspecto de la pantalla OLED en la ejecución de la aplicación

12.2. OLED Señal Analógica

Objetivo

Queremos registrar en la pantalla OLED el valor analógico leído desde un canal

Funcionalidad

Designaremos un cana analógico en el pin GP27 y tomaremos el valor leído y lo pondremos en el display OLED que, como en el caso anterior, esta conectado en el I2C

Programa

El programa se realizará en los siguientes pasos.

1. Cargar las librerías

```
from machine import Pin, I2C,ADC
from ssd1306 import SSD1306_I2C
import framebuf
import utime
```

2. Definimos el pin de entrada de la señal analógica GP27, el ancho y alto en pixels de la pantalla y también el I2C

```
analogica=ADC(27)

WIDTH  = 128   # Pixels de la Base
HEIGHT = 64    # Pixels de la Altura

# Inicia I2C usando GP4 y GP5 (I2C0)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c) # Inicializa el OLED
```

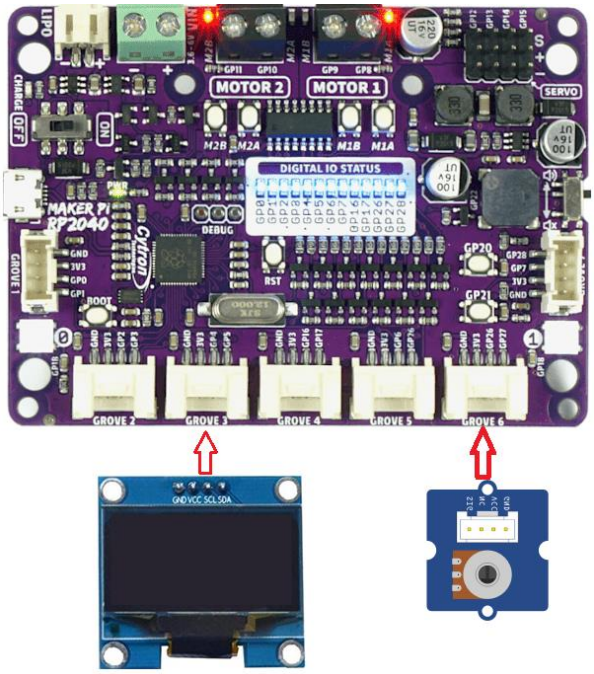
3. En la parte **while** dl programa escribiremos en la pantalla el valor de la señal analógica leída en el pin GP27

```
while True:
    oled.fill(0) # Borra el display OLED.
    adcValue = analogica.read_u16()    # Añade texto
    oled.text("Valor= "+ str(adcValue),0,10) # Coordenada x=0, y=10
    oled.show()# Descarga el texto en la pantalla y lo muestra
```

Programa completo

```
# Muestra el valor del canal analogico GP27 con
# OLED conectado en el bus I2C con la librería ssd1306
from machine import Pin, I2C,ADC
from ssd1306 import SSD1306_I2C
import framebuf
import utime

analogica=ADC(27)

WIDTH  = 128   # Pixels de la Base
HEIGHT = 64    # Pixels de la Altura

i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)# Inicia I2C usando GP4 y GP5 (I2C0)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c) # Inicializa el OLED

while True:
    # Borra el display OLED.
    oled.fill(0)
    # Añade texto
    adcValue = analogica.read_u16()
    oled.text("Valor= "+ str(adcValue),0,10) # Coordenada x=0, y=10
    # Descarga el texto en la pantalla y lo muestra
    oled.show()
```

Montaje de la practica



Análisis y propuesta de actividades

Este sería el aspecto de la pantalla de salida

12.3. OLED Dibuja Varias cosas

Objetivo

Con esta práctica vamos a seguir probando el display con las distintas funciones que nos permiten escribir y/o dibujar varias cosas.

Funcionalidad

La construcción del programa es similar a las anteriores

El programa se realizará en los siguientes pasos.

1. Cargar las librerías

```
from machine import Pin, I2C,ADC
from ssd1306 import SSD1306_I2C
import framebuf
import utime
```

2. Definimos el ancho y alto en pixels de la pantalla y también el I2C e iniciamos el conexionado del display en el puerto I2C

```
WIDTH  = 128   # Pixels de la Base
HEIGHT = 64    # Pixels de la Altura

# Inicia I2C usando GP4 y GP5 (I2C0)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c) # Inicializa el OLED
```

3. Realizamos las distintas operaciones para :

   Cada una de estas entidades de dibujo se definen con la correspondiente función

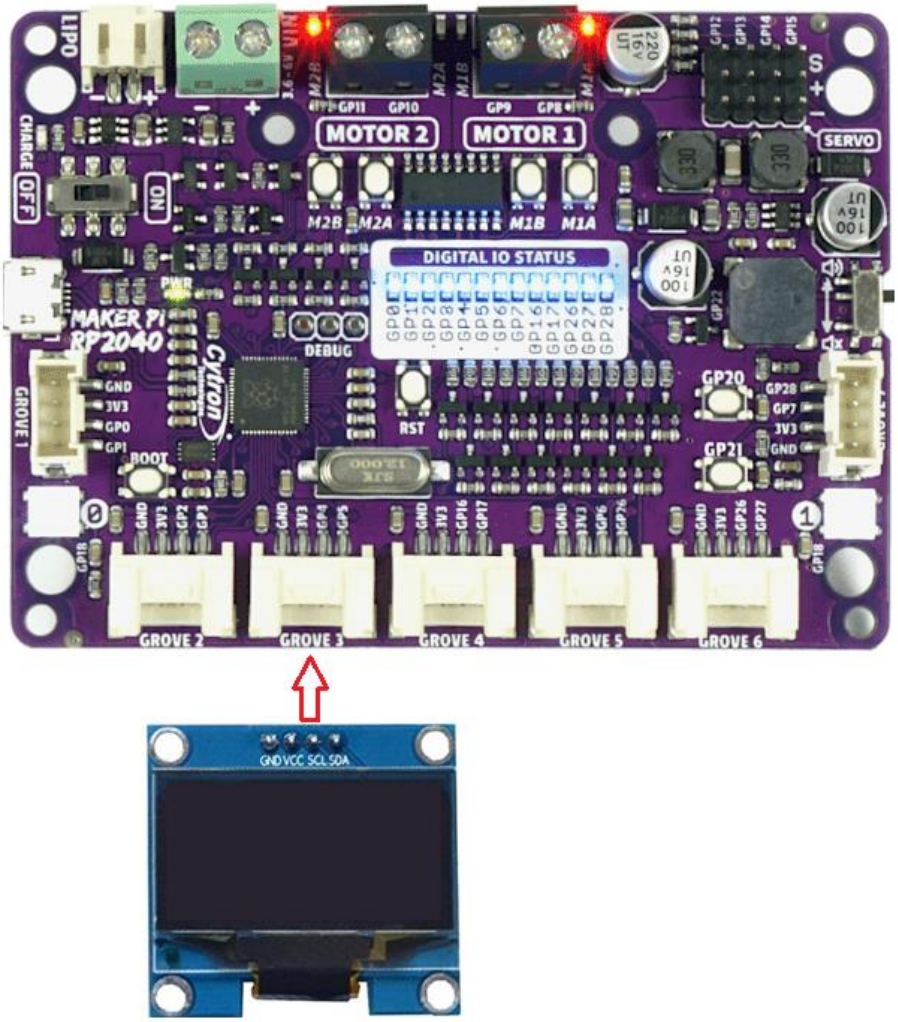
   - Apagar y encender pantalla. **oled.fill**(0) , **oled.fill**(1)
   - Muestra un Pixel. **oled.pixel**(11, 5, 1)
   - Muestra un Icono
   - Muestra líneas. **oled.line**(4, 5, 18, 9, 1)#
   - Línea Vertical. **oled.vline**(11, 5, 10, 1)
   - Línea Horizontal. **oled.hline**(11, 5, 10, 0)
   - Cuadrado. **oled.fill_rect**(5, 3, 14, 12, 1)#
   - Rectángulo relleno. **oled.fill_rect**(5, 3, 14, 12 ,0)

Programa

```
#Dibujar en un dispositivo OLED distintos objetos
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import framebuf
import utime

WIDTH  = 128    # oled display width
HEIGHT = 64

i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
# Apaga y enciende pantalla
oled.fill(1)
oled.show()     # Rellena la pantalla (la ilumina entera)
utime.sleep(2)  # Espera 2 segundos
oled.fill(0)    # Rellena la pantalla (la apaga entera)
oled.show()

#Muestra un Pixel
oled.fill(0)
oled.pixel(11, 5, 1)  # Dibuja un píxel iluminado en (5,10)
oled.show()           # Muestra el resultado
utime.sleep(2)              # Espera 2 segundos
oled.pixel(11, 5, 0)  # Dibuja un píxel apagado en (10,5)
oled.show()           # Muestra el resultado

#Muestra un Icono
from machine import Pin, I2C

ICONO = [                       # Matriz de puntos
    [ 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [ 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [ 0, 1, 1, 1, 1, 1, 1, 1, 0],
    [ 1, 1, 0, 0, 1, 0, 0, 1, 1],
    [ 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [ 1, 0, 1, 0, 0, 0, 1, 0, 1],
    [ 1, 0, 1, 1, 1, 1, 1, 0, 1],
    [ 0, 0, 1, 1, 0, 1, 1, 0, 0],
    [ 0, 1, 1, 1, 0, 1, 1, 1, 0],
]
oled.fill(0)
for y, fila in enumerate(ICONO): # Dibuja los puntos de la matriz
    for x, c in enumerate(fila):
        oled.pixel(x, y, c)
oled.show()
utime.sleep(2)  # Espera 2 segundos
# Muestra el resultado
```

```
#Muestra lineas
oled.fill(0)
oled.line(4, 5, 18, 9, 1)# Dibuja una línea iluminada. Origen (4, 5) y final (18, 9)
oled.show()# Muestra el resultado
utime.sleep(2)# Espera 2 segundos
oled.fill(0) # Rellena la pantalla (la ilumina entera)
oled.line(4, 5, 18, 9, 0)# Dibuja una línea apagada. Origen (4, 5) y final (18, 9)
oled.show()

#Linea Vertical
oled.vline(11, 5, 10, 1) # Dibuja una línea vertical iluminada. Origen (11, 5) altura 10
píxels
oled.show()            # Muestra el resultado
utime.sleep(2)          # Espera 2 segundos
oled.fill(0)           # Rellena la pantalla (la ilumina entera)

#Linea Horizontal
oled.hline(11, 5, 10, 0) # Dibuja una línea vertical apagada. Origen (11, 5) altura 10
píxels
oled.show()            # Muestra el resultado
utime.sleep(2)

#Cuadrado
oled.fill_rect(5, 3, 14, 12, 1)# Dibuja un rectángulo relleno iluminado. Origen (5, 3)y
anchura x altura 14x12 píxels
oled.show()                # Muestra el resultado
utime.sleep(2)              # Espera 2 segundos

#Rectángulo relleno
oled.fill_rect(5, 3, 14, 12 ,0) # Dibuja un rectángulo relleno apagado. Origen (5, 3)y
anchura x altura 14x12 píxels
oled.show()
```

Montaje de la practica



12.4. OLED Senoidal

Objetivo

Vamos experimentar con el dispositivo OLED mostrando el gráfico de una señal senoidal

Funcionalidad

Con este ejemplo solo queremos demostrar las posibilidades de dispositivo OLED, se entiende que la complejidad del algoritmo se escapa al nivel de este libro por lo que solo nos dedicamos a escribir l programa para que el lector lo pruebe y si acaso lo investigue.

Programa

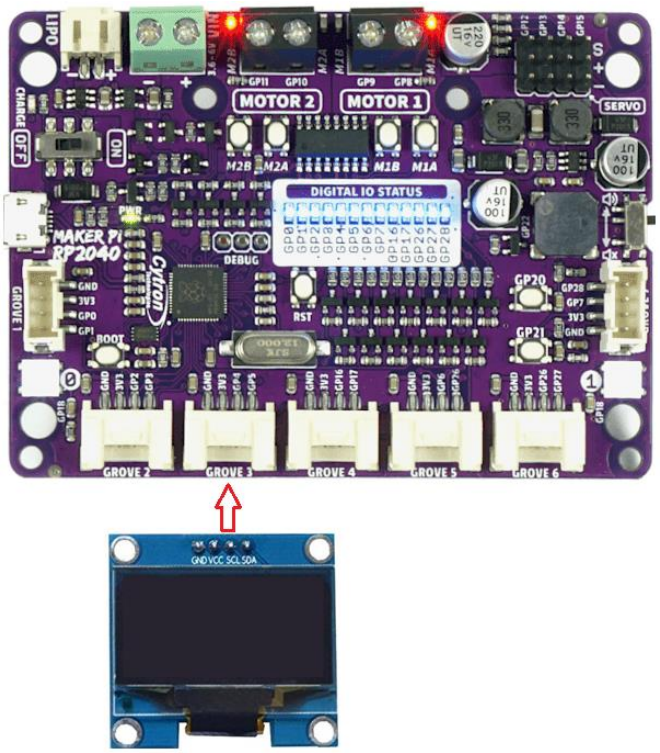
```
#Onda Senoidal
from machine import Pin, I2C
import ssd1306
from time import sleep
import math
i2c = I2C(0, scl=Pin(5), sda=Pin(4))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

while True:
    for anguloGrados in range(0, 360, 5):                # Ángulo de giro de la línea del eje (de 0º a 360º en intervalos de 5º)
        anguloRadianes = (math.pi*anguloGrados)/180      # Ángulo de giro de la línea del eje en radianes
        xLinea = int(math.cos(anguloRadianes)*24)        # Coordenada x línea del eje
        yLinea = int(math.sin(anguloRadianes)*24)        # Coordenada y línea del eje
                                                         # Circunferencia. Radio 25 pixels. Centro (24, 32)
        for anguloGrados in range(0, 360, 5):            # Ángulo de giro de cada pixel de la circunferencia en grados
            anguloRadianes = (math.pi*anguloGrados)/180  # Ángulo de giro de cada pixel de la circunferencia en radianes
            x = int(math.cos(anguloRadianes)*24)         # Coordenada x pixel circunferencia
            y = int(math.sin(anguloRadianes)*24)         # Coordenada y pixel circunferencia
        oled.pixel(x+24, y+32, 1)                        # Dibuja cada pixel de la circunferencia
        oled.hline(0, 32, 128, 1)                        # Dibuja la línea horizontal central
        oled.vline(49, 0, 64, 1)                         # Dibuja la línea vertical
        oled.line(24 , 32, xLinea+24, yLinea+32, 1)      # Dibuja la línea del eje. De (24, 32) a (xLinea, yLinea)
        oled.hline(xLinea+24, yLinea+32, 27-xLinea,1)    # Dibuja la línea horizontal giro
        oled.show()                                      # Muestra el resultado
        oled.scroll(1, 0)                                # Desplaza imagen un pixel a la derecha
        oled.fill_rect(0, 0, 51, 64, 0)                  # Tapa desde origen hasta raya vertical...
                                                         # ...toda la altura de la pantalla
```

Página Editor de Thonny

```
#Onda Senoidal
from machine import Pin, I2C
import ssd1306
from time import sleep
import math
i2c = I2C(0, scl=Pin(5), sda=Pin(4))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

while True:
      # Ángulo de giro de la línea del eje (de 0º a 360º en intervalos de 5º)
   for anguloGrados in range(0, 360, 5):
      # Ángulo de giro de la línea del eje en radianes
      anguloRadianes = (math.pi*anguloGrados)/180
      xLinea = int(math.cos(anguloRadianes)*24)   # Coordenada x línea del eje
```

```
        yLinea = int(math.sin(anguloRadianes)*24)    # Coordenada y línea del eje

# Circunferencia. Radio 25 pixels. Centro (24, 32)
# Ángulo de giro de cada pixel de la circunferencia en grados
    for anguloGrados in range(0, 360, 5):

# Ángulo de giro de cada pixel de la circunferencia en radianes
      anguloRadianes = (math.pi*anguloGrados)/180
      x = int(math.cos(anguloRadianes)*24)   # Coordenada x pixel circunferencia
      y = int(math.sin(anguloRadianes)*24)    # Coordenada y pixel circunferencia
      oled.pixel(x+24, y+32, 1) # Dibuja cada pixel de la circunferencia
      oled.hline(0, 32, 128, 1)    # Dibuja la línea horizontal central
      oled.vline(49, 0, 64, 1)      # Dibuja la línea vertical

# Dibuja la línea del eje. De (24, 32) a (xLinea, yLinea)
      oled.line(24 , 32, xLinea+24, yLinea+32, 1)

 # Dibuja la línea horizontal giro
        oled.hline(xLinea+24, yLinea+32, 27-xLinea,1)
        oled.show()                    # Muestra el resultado
        oled.scroll(1, 0)               # Desplaza imagen un pixel a la derecha
        oled.fill_rect(0, 0, 51, 64, 0)  # Tapa desde origen hasta raya vertical...
                                         # ...toda la altura de la pantalla
```

Montaje de la practica



13.Dispositivo LCD

Grove - 16x2 LCD

Grove - 16 x 2 LCD es una pantalla LCD I2C perfecta para Arduino y Raspberry Pi con alto contraste y fácil implementación. 16x2 significa dos líneas y cada línea tiene 16 columnas, 32 caracteres en total. Con la ayuda del conector Grove I2C, solo se necesitan 2 pines de señal y 2 pines de alimentación. Ni siquiera necesita preocuparse por cómo conectar estos pines. Simplemente conéctelo a la interfaz I2C en Seeeduino o Arduino / Raspberry Pi + base-shield a través del cable Grove. No habrá cableado complicado, soldadura, preocuparse por quemar la pantalla LCD causada por la resistencia limitadora de corriente incorrecta.

Versiones

Las versiones son las siguientes.

The Grove - LCD de 16 x 2 (negro sobre amarillo)
The Grove - LCD 16 x 2 (Negro sobre Rojo)
The Grove - LCD de 16 x 2 (blanco sobre azul) *(el que usaremos*

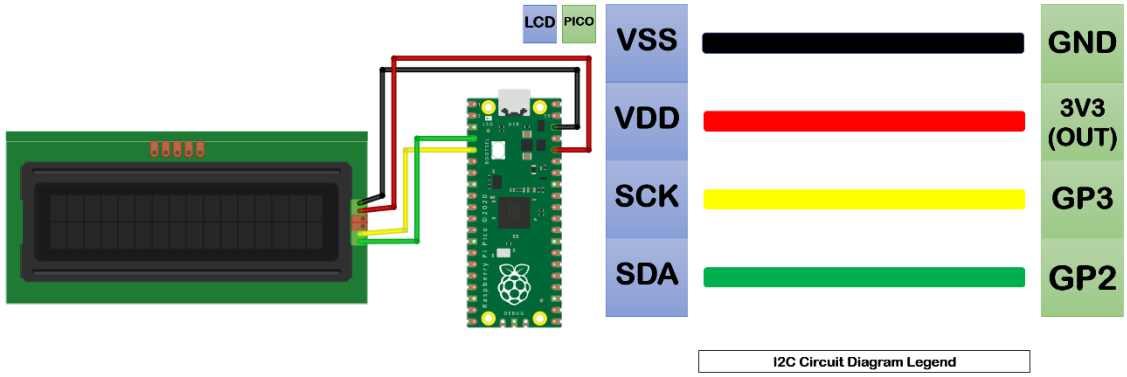
Especificaciones

Voltaje de funcionamiento	3.3V/5V
Temperatura de funcionamiento	0 a 50°C
Temperatura de almacenamiento	-10 a 60°C
Método de conducción	1/16 de deber, <>/<> de sesgo
Interfaz	Yo²C
Yo²Dirección C	0X3E

El Protocolo

Pantalla LCD funciona con el protocolo de comunicación serie I2C.
Que es el protocolo de comunicación que sólo utiliza dos hilos para la comunicación que son **los datos (SDA)** y otro, el **reloj (SCL).** La dirección de comunicación entre esta

pantalla LCD y el dispositivo es **62 (DEC)** o **0x3E**. A continuación se muestra el diagrama del circuito para configurar la pantalla LCD con protocolo I2C en Pi Pico:



Librería que usaremos

Para el manejo de este dispositivo vamos a usar la librería que nos recomienda **Cytron**:

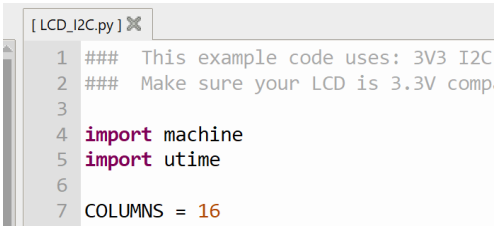
GitHub - Bhavithiran97/LCM1602-14_LCD_Library: This LCD module can be connected using SPI or I2C protocol

Esta librería permita el conexionado en dos modos SPI e I2C, nosotros vamos a trabajar con el protocolo I2C. Documento tutorial de **Cytron**

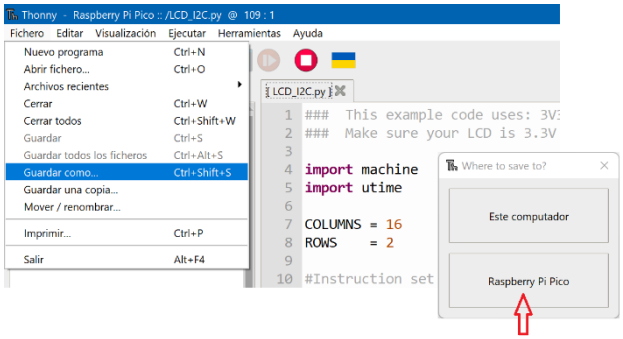
Utilizacion de la librería para I2C:

Paso 1: Cargar la librería LCD_I2C.py

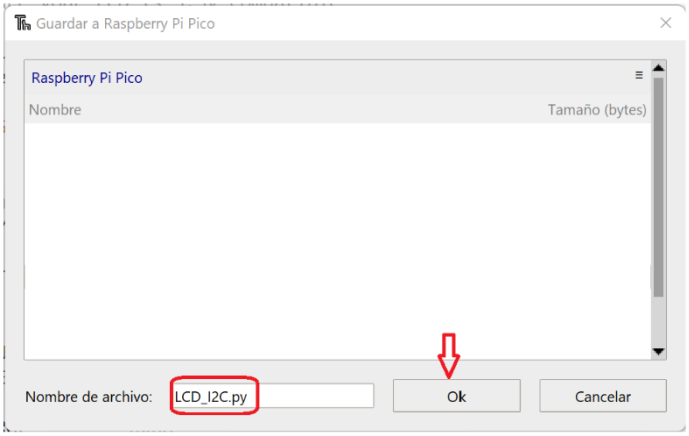
Paso 2: Inicie la aplicación **Thonny** y abra el archivo de la biblioteca

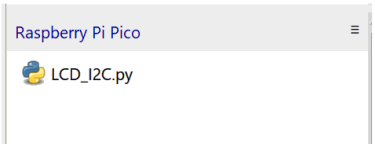


Paso 3: Guarde el archivo de la biblioteca en Pi Pico



Paso 4: Guarde la biblioteca con el mismo nombre que descargó con (LCD_I2C.py)
Debe agregar .py en la parte posterior





Haga clic en Aceptar y la biblioteca se agregará a su Pi Pico

Instrucciones

Importar LCD_I2C Library

```
from LCD_I2C import *
```

Crear el Objeto para I2C

```
lcd = LCD(sda=4, scl=5)
```

Posiciona cursor

Establecer el cursor en una posición específica. El primer parámetro establece la columna, el segundo parámetro establece la fila. Coloque el cursor en la columna 0 (primera columna) y la fila 0 (primera fila)

```
lcd.set_cursor(0,0)
```

On LCD Display

Encienda la pantalla LCD sin borrar los datos. El cursor y el parpadeo están desactivados de forma predeterminada.

```
lcd.on()
```

- Cursor encendido, parpadeo apagado

```
lcd.on(cursor=True, blink=False)
```

- Cursor apagado, parpadeo encendido

lcd.on(cursor=False, blink=True)

Cursor encendido, parpadeo encendido

lcd.on(cursor=True, blink=True)

Apagado Pantalla LCD

- Apaga la pantalla LCD sin borrar los datos

lcd.off()

Write to LCD Display

- Escribir cadena en la pantalla LCD

lcd.write("Hello World")

Borra la pantalla LCD

- Borra el dato en el display

lcd.clear()

13.1 LCD Básico

Objetivo

Este primer ejemplo de manejo del dispositivo LCD vamos a ver como conectarlo y poner en el un texto y una cifra.

Funcionalidad

Este ejercicio resulta muy sencillo. Bastará con conectar el dispositivo **LCD** que como hemos visto en el tema anterior se comunica con **Raspberry Pi Pico** a través del puerto **I2C.**

En el dispositivo aparecerá:

"HOLA MUNDO"
123

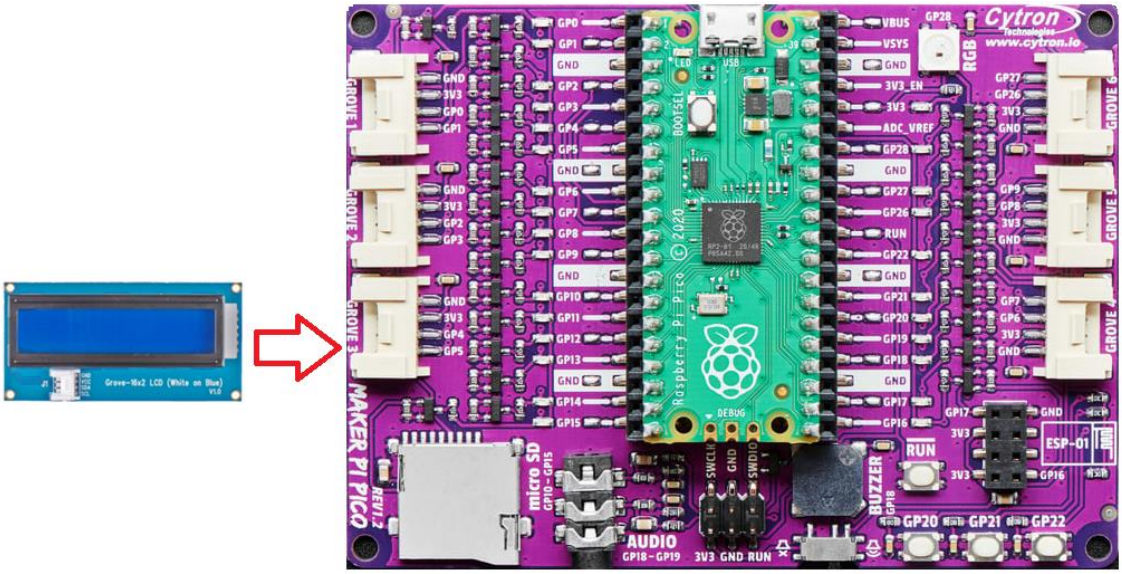
Programa

Nuestro programa se construye siguiendo los pasos siguientes:

1. Importación de librerías
2. Definimos el objeto lcd poniendo los números de los pines asociados a **sda**=4 y **scl**=4
3. Situamos el cursor en la **fila 0** y la **columna 0** (al principio del área de escritura)
4. En el bucle **while** escribimos el texto "*HOLA MUNDO*" y la cifra "**123**" que previamente hemos de convertir en una cadena de texto con la función "**str**" de **MicroPython**

```
###  Mostrar un Texto y un numero
###  Asegúrate que tu LCD es compatible con 3.3V
###  La comunicación del LCD es via I2C
import machine
import time
from LCD_I2C import *    #import LCD_I2C library

lcd = LCD(sda=4, scl=5)  #Deficion de los pines de conexión
lcd.set_cursor(0,0)  #Situación del cursor para el comienzo de la escritura

while True:
    lcd.set_cursor(0,0)
    lcd.write("HOLA MUNDO")# imprime una cadena de texto
    lcd.set_cursor(0,1)
    lcd.write(str(123)) # Se convierte en string una cifra y se imprime
```

Montaje de la practica



13.2 LCD Lectura de valor analógico

Objetivo

En esta práctica vamos a imprimir un valor analógico que leeremos en uno de los canales de Raspberry dispuesto para este tipo de señal.

Funcionalidad

La configuración de nuestra aplicación es similar a la llevada a cabo en el anterior ejemplo.

En este caso el valor a escribir será un valor adquirido en tiempo real a través de un potenciómetro que conectaremos en el pin GP27 de la tarjeta **Cytron**.

Imprimiremos también el valor analógico en la consola de **Thonny**.

El valor que entrega el sensor lo convertimos en un valor **entero** y después en un **string.**

Se van a imprimir dos líneas en la primera un texto "Valor Analógico" y en la segunda el valor analógico.

Valor Analógica
123

El refresco de la pantalla es muy importante dado que al clocar la orden de impresión dentro de un bucle al realizar las sucesivas impresiones que ordena el bucle podremos un retardo para conseguir que el efecto visual sea lo mas cómodo posible aunque no podremos evitar un cierto "**parpadeo**" del valor. Seguidamente se borrara el display.

Programa

Nuestro programa se construye:

1. Carga de las librerías.

2. Definición del dispositivo LCD designando los pines de conexión. Definición de la variable "potenciómetro" que recogerá el valor analógico.

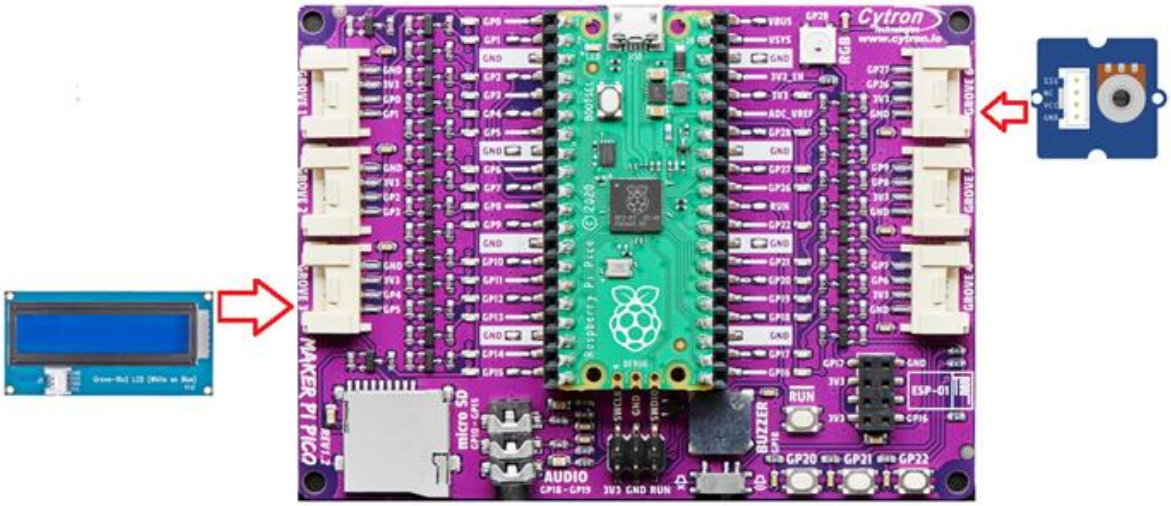
3. En el bucle **while** pondremos en primer lugar la impresión del valor en el terminal de **Thonny**. Seguidamente posicionamos el curos e imprimimos el texto "**Valor Analógico**".

4. El valor analógico lo extraemos del pin **GP27** y lo convertimos en un valor de tipo **int** (entero) y seguidamente lo convertimos en un **str** string o cadena de texto.

5. Finalmente incluimos un retardo de 50 ms. Y borramos la pantalla para conseguir el refresco del valor a imprimir.

```
###  Mostrar un valor analógico sacado de GP27
###  Asegúrate que tu LCD es compatible con 3.3V
###  La comunicación del LCD es vía I2C
import machine
import time
from LCD_I2C import *    #importa LCD_I2C library

lcd = LCD(sda=4, scl=5) #Define el objeto "lcd"
lcd.set_cursor(0,0)  #Situa el cursor en (0,0)
potentiometro = machine.ADC(27) # Potenciómetro conectado en GP(27)

while True:
    print(potentiometro.read_u16())
    lcd.set_cursor(0,0)
    lcd.write("Valor Analogico")# imprime el valor analógico
    lcd.set_cursor(0,1)
    lcd.write(str(int(potentiometro.read_u16())))
    utime.sleep_ms(500)
    lcd.clear()# retardo de 50ms..
```

Montaje de la practica



13.3 LCD Termostato

Objetivo

Vamos a crear una aplicación que emule el funcionamiento de un termostato

Funcionalidad

Dispondremos de un potenciómetro con el que simularemos el valor de un rango de variación de temperatura que conseguiremos mapeando el valor leído a un valor entre 20ºC y 60 ºC.

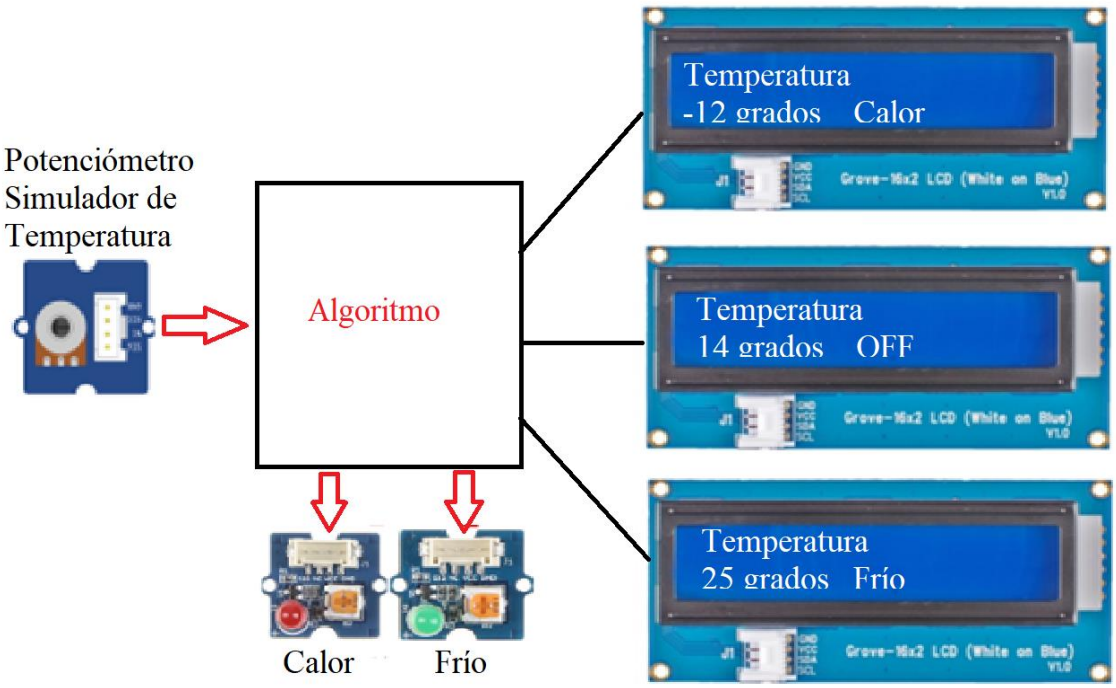
Nuestro algoritmo de control establecerá tres rangos de valor que se adecuaran a los tres niveles que queremos controlar: Calor, OFF y Frío

- **Calor** significará que la temperatura es menor de 10 ºC y se activara el Calefactor que nosotros emulamos con un LED de color Rojo

- **OFF** significará que la temperatura esta comprendida entre 10ºC y 20ºC.En este caso permanecerán desactivados los elementos Calefactor y Refrigerador. Ambos LEDs se mantendrán apagados.

- **Frio** significará que la temperatura es mayor de 20 ºC lo que hará que se conecte el equipo de Refrigeración. El LED verde se activara

- En cada uno de los estados se mostrará en el LCD el texto correspondiente junto con el valor leído en tiempo real.



Programa

La construcción del programa se llevará a cabo de acuerdo a la secuencia siguiente.

1. Importación e las librerías como en todos los programas. La librería especia en esta ocasión es:

   ```
   from LCD_I2C import *
   ```

2. Definimos las variables: **Calor** (digital) **Frio** (Digital) y **lcd** (objeto de la librería LCD) y **pot** (potenciómetro para simular las variaciones de temperatura)

3. Definimos la conocida función **map** que nos servirá para cambiar la escala de la señal analógica leída en el puerto GP27 en el que tenemos que conectar el potenciómetro.

   ```
   def map(s, a1, a2, b1, b2):
       return b1 + (s - a1) * (b2 - b1) / (a2 - a1)
   ```

4. La siguiente fase es la que se relaciona con el bucle **while True** dentro del cual tendremos que:
   - Leer la señal del pin GP27
   - Mapear la señal igualando a una nueva variable local llamado “valor” que imprimiremos dando formato.
   - Posicionamos el curos del LCD en el punto (0,0), escribimos la cadena de texto “Temperatura” y en la segunda línea escribiremos la variable valor (ya mapeada) y leemos el valor del potenciómetro, seguida de la cadena de texto “Grados”

5. Establecemos los condicionales para distinguir en que zona de la escala de temperatura estamos y en función de ello mostraremos un texto y activaremos o no una de las dos salidas digitales “Frio” y “Calor” o el estado “OF10F”

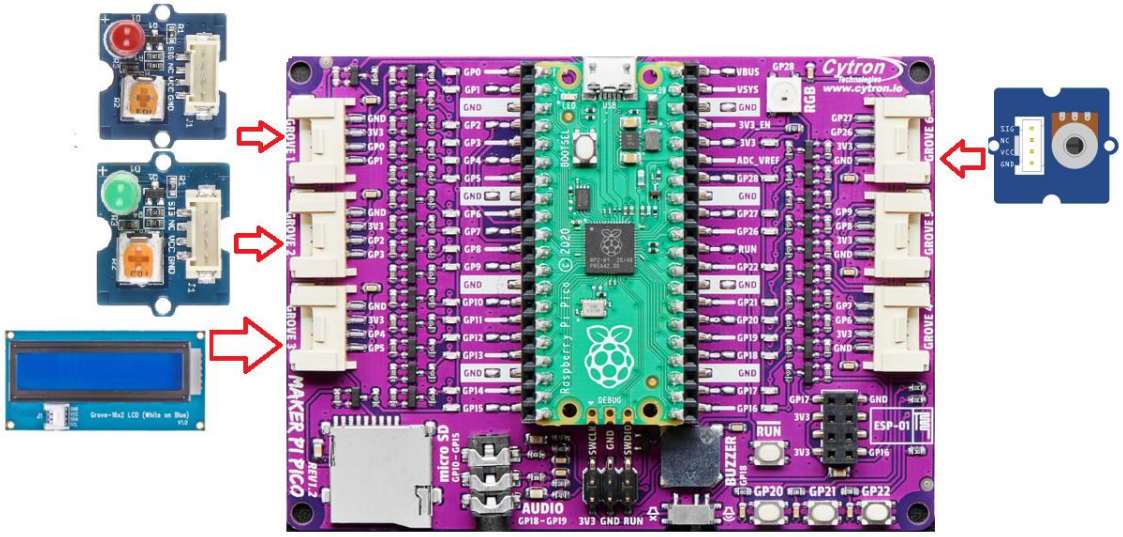
   En el apartado de funcionamiento hemos escrito estas condiciones que se resumen aquí:

   - Si valor<10 entonces Escribe “Frio” y activa la salida Frio (pin GP3)
   - Si valor>20 entonces Escribe “Calor” y activa la salida Calor (pin GP1)
   - Si 10<valor<20 entonces Escribe “OFF” se desactivan las salidas GP1 y GP3

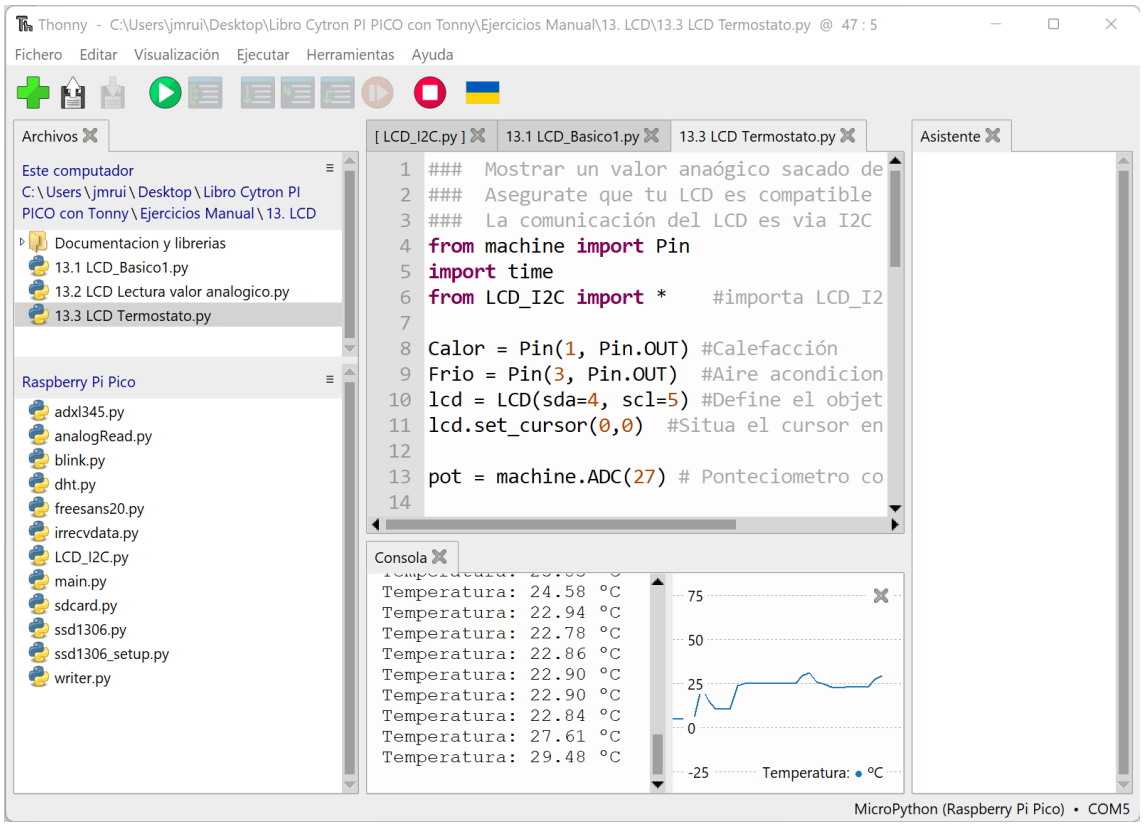
A continuación vemos el programa completo

```
###  Mostrar un valor analógico sacado de GP27
###  Asegúrate que tu LCD es compatible con 3.3V
###  La comunicación del LCD es vía I2C
from machine import Pin
import time
from LCD_I2C import *    #importa LCD_I2C library

Calor = Pin(1, Pin.OUT) #Calefacción
Frio = Pin(3, Pin.OUT)  #Aire acondicionado
lcd = LCD(sda=4, scl=5) #Define el objeto "lcd"
lcd.set_cursor(0,0)  #Situa el cursor en (0,0)
pot = machine.ADC(27) # Potenciómetro conectado en GP(27)

def map(s, a1, a2, b1, b2):
return b1 + (s - a1) * (b2 - b1) / (a2 - a1)

while True:
lectura = pot.read_u16()
valor = map(lectura, 0, 65535, -20, 60)
#Imprimimos solo dos decimales "{:.2f}".format(x)
print("Temperatura:", "{:.2f}".format(valor)+ " ºC")
lcd.set_cursor(0,0)
lcd.write("Temperatura")# print analog value to serial
lcd.set_cursor(0,1)
lcd.write(str(int(valor))+" Grados")
if valor>20:
lcd.set_cursor(27,1)
lcd.write("Frio")
Frio.value(1)
Calor.value(0)
elif valor<10:
lcd.set_cursor(27,1)
lcd.write("Calor")
Frio.value(0)
Calor.value(1)
elif (10 < valor <20):
lcd.set_cursor(27,1)
lcd.write("OFF")
Frio.value(0)
Calor.value(0)
utime.sleep_ms(300)
lcd.clear()  # retado de 50ms.
```

Montaje de la practica



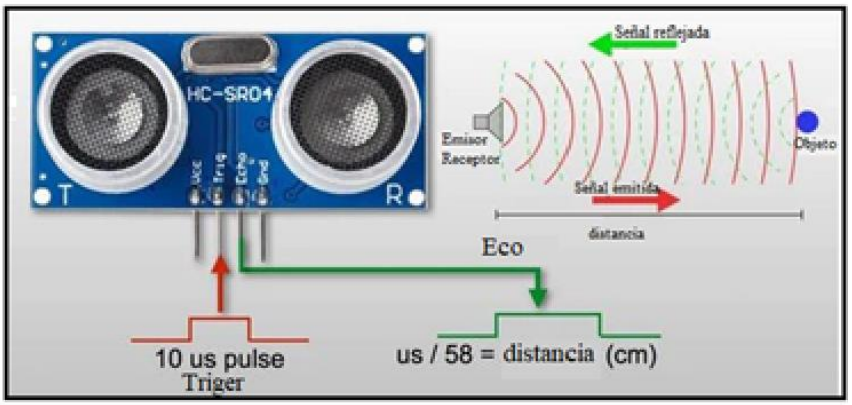
Análisis y propuesta de actividades



Vemos en la imagen la posibilidad de representar gráficamente el valor de la señal.

14 Midiendo con Ultrasonidos

Medir distancias es una tarea muy propia de los sistemas de control y de los dispoitivos roboticos, por ese motivo vamos a inlcuir en este libros unos nejemplos de manejo de un sensor de medidor de distancia mediante ultrasonidos.



Vamos a usar un sensor comoel mostrado en la imagen anterior con dos pines activos: **Eco** y **Trigger**

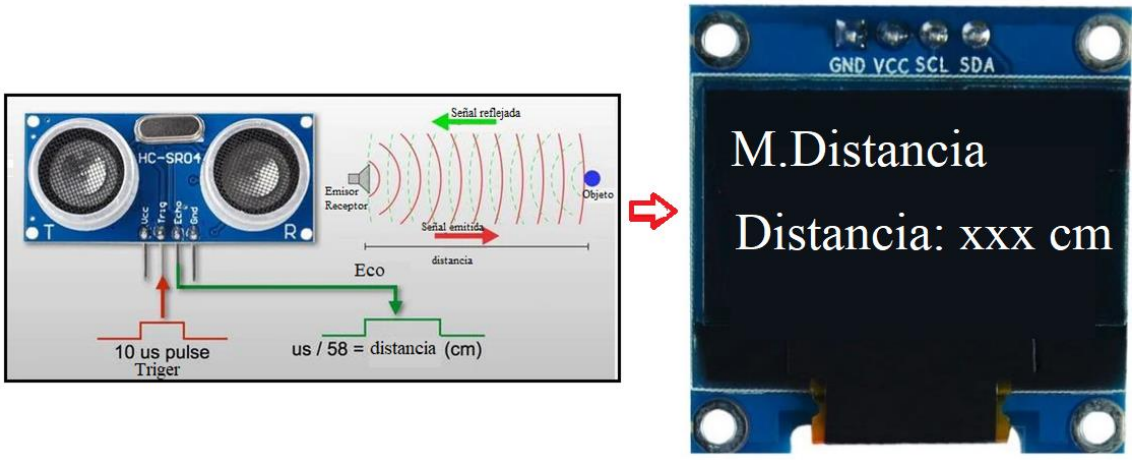
14.1. Medidor básico de ultrasonidos

Objetivo

Se pretende conocer el funcionamiento de un dispositivo para medir distancias basado en ultrasonidos.

Funcionalidad:

Vamos a manejar el sensor HC-SR04 que conectaremos a nuestra tarjeta. La medida que realicemos la mostraremos en un display OLED



Conectaremos el dispositivo sensor a nuestra tarjeta de acuerdo a la designación de las entradas salidas:

Los dispositivos que usaremos

Sus conexiones a la tarjeta

- **GP4** **SDA** del dispositivo **OLED**
- **GP5** **SCL** del dispositivo **OLED**
- **GP0** **Trig** del Dispositivo **HC SR04**
- **GP1** **Eco** del dispositivo **HC SR04**

Programa

El programa sería el de la siguiente figura.

Básicamente hemos seguido los pasos:

- Cargamos las librerías

```
from machine import Pin, I2C,ADC
from ssd1306 import SSD1306_I2C
import framebuf
import utime
```

- Definimos los pines de conexión del dispositivo de ultrasonidos

```
Trig = machine.Pin(0,machine.Pin.OUT)
Echo = machine.Pin(1,machine.Pin.IN)
```

- Definimos la conexión I2C del dispositivo OLED.

```
WIDTH  = 128   # Pixels de la Base
HEIGHT = 64    # Pixels de la Altura
# Inicia I2C usando GP4 y GP5 (I2C0)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
```

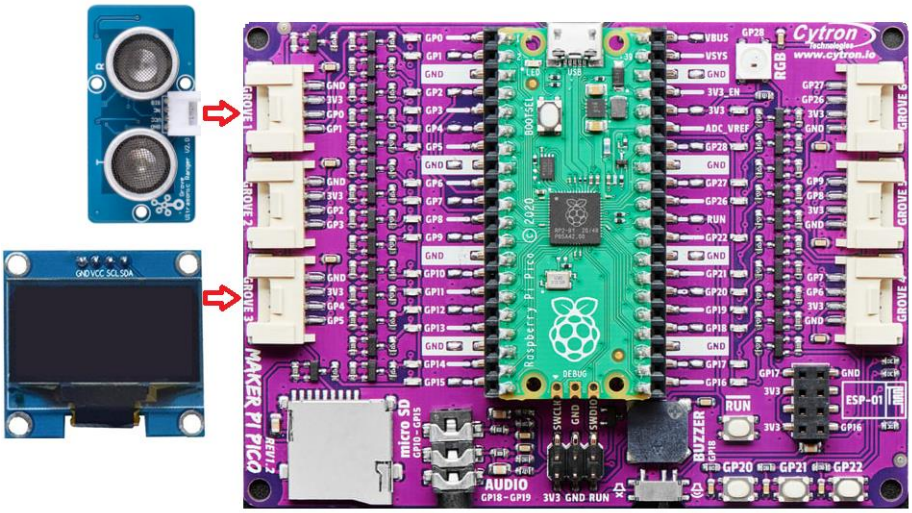
- Creamos la función de control del dispositivo de ultrasonidos para generar los impulsos de salida (Trig) y la recepción de los impulsos de (Echo).
- Dentro del bucle **while** se enviara al display el texto de cada una de las dos líneas que indicaran

M.Distancia
Distancia: *distancia* **cm**

Programa completo

```
# Medimos el valor de la distancia  con un sensor HC-SR04
# Muestra el valor leído en un dispositivo OLED conectado
# en el bus I2C con la librería ssd1306
from machine import Pin, I2C,ADC
from ssd1306 import SSD1306_I2C
import framebuf
import utime
# Conectar pin Trig con GP0 como SALIDA
# Conectar pin Echo con GP1 como ENTRADA

Trig = machine.Pin(0,machine.Pin.OUT)
Echo = machine.Pin(1,machine.Pin.IN)

WIDTH  = 128   # Pixels de la Base
HEIGHT = 64    # Pixels de la Altura
# Inicia I2C usando GP4 y GP5 (I2C0)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
# Inicializa el OLED
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
# Función "Leer Ultrasonidos"
def read_ultrasonic():
    Trig.value(0)
    utime.sleep_us(2)
    Trig.value(1)
    utime.sleep_us(10)
    Trig.value(0)

    while Echo.value() == 0:
        pulse_start = utime.ticks_us()

    while Echo.value() == 1:
        pulse_end = utime.ticks_us()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration//58
    return distance

while True:
    # Borra el display OLED.
    oled.fill(0)
    # Añade texto
    oled.text("M.Distancia",0,0)
    oled.text("Distancia:"+ str(read_ultrasonic())+"cm",0,32) # Coordenada x=0, y=10
    # Descarga el texto en la pantalla y lo muestra
    oled.show()
```

Esquema de montaje:



14.2. Medidor de ultrasonidos con emision de tonos

Objetivo

Aplicar el uso de un **detector de distancia** para la emisión de un sonido en función de la distancia. Esta es la emulación de los sistemas de aparcamiento asistido en los automóviles.

Funcionalidad:

Vamos a montar el mismo ejemplo anterior, pero en este nuevo, incorporaremos una función sonora. Se trata de usar la librería "**Tonos**" con el fin de usar el bloque de generación de un sonido "**toca la frecuencia…**" cuyo valor de frecuencia será variable estando en función de la distancia que nos mide el sensor de distancia.

Multiplicamos el valor de la "Distancia" para conseguir una variación de frecuencias audibles, también podríamos haber usado la función "**rescale**"

Entradas salidas:

Los dispositivos que usaremos

Sus conexiones a la tarjeta

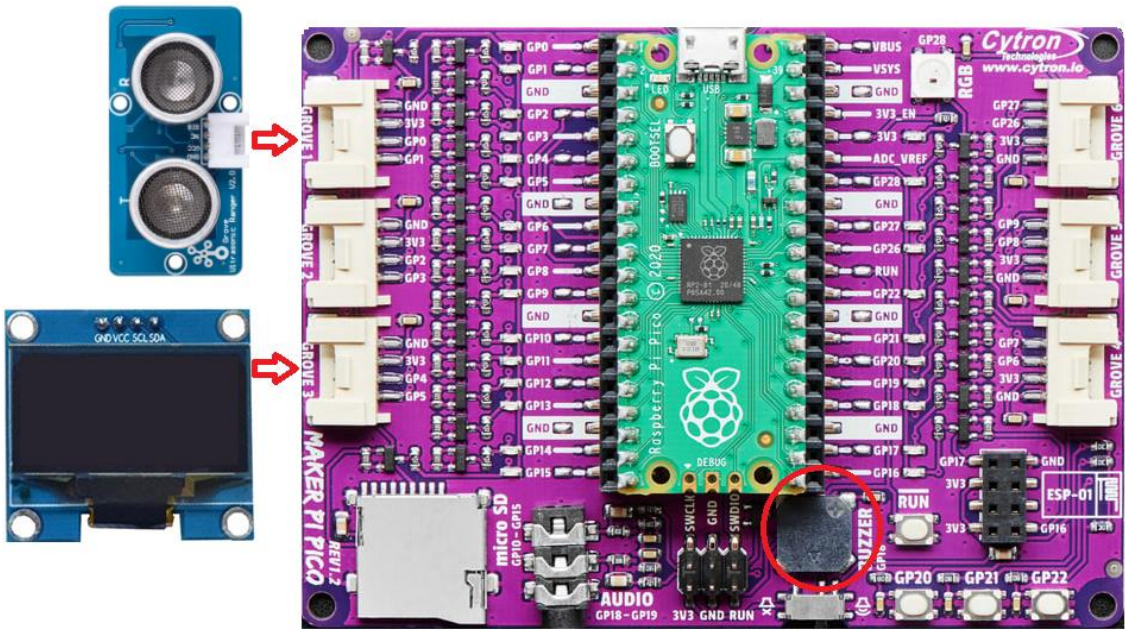
- **GP4 SDA** del dispositivo **OLED**
- **GP5 SCL** del dispositivo **OLED**
- **GP0 Trig** del Dispositivo **HC SR04**
- **GP1 Eco** del dispositivo **HC SR04**
- **GP18 Buzer** de la tarjeta

Programa:

```
# Medimos el valor de la distancia  con un sensor HC-SR04
# Muestra el valor leído en un dispositivo OLED conectado
# en el bus I2C con la librería ssd1306
# Emite un tono en función de la distancia
from machine import Pin, I2C,ADC,PWM
from ssd1306 import SSD1306_I2C
import utime
# Conectar pin Trig con GP0 como SALIDA
# Conectar pin Echo con GP1 como ENTRADA
```

```
Trig = machine.Pin(0,machine.Pin.OUT)
Echo = machine.Pin(1,machine.Pin.IN)
buzzer = PWM(Pin(18)) #Define pin de salida

WIDTH  = 128   # Pixels de la Base
HEIGHT = 64    # Pixels de la Altura
# Inicia I2C usando GP4 y GP5 (I2C0)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
# Inicializa el OLED
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
# Función "Leer Ultrasonidos"
def read_ultrasonic():
    Trig.value(0)
    utime.sleep_us(2)
    Trig.value(1)
    utime.sleep_us(10)
    Trig.value(0)

    while Echo.value() == 0:
        pulse_start = utime.ticks_us()

    while Echo.value() == 1:
        pulse_end = utime.ticks_us()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration//58
    return distance

while True:
    valor=read_ultrasonic()
    buzzer.freq(valor*100) #Frecuencia del tono
    buzzer.duty_u16(8000) #Volumen del sonido
    utime.sleep_us(10)
    oled.fill(0)
    # Añade texto
    oled.text("M.Distancia",0,0)
    oled.text("Distancia:"+ str(valor)+"cm",0,20) # Coordenada x=0, y=10
    oled.text("Frec.:"+ str(valor*100)+" hz",0,40) # Coordenada x=0, y=10
    # Descarga el texto en la pantalla y lo muestra
    oled.show()
```

Esquema de montaje:

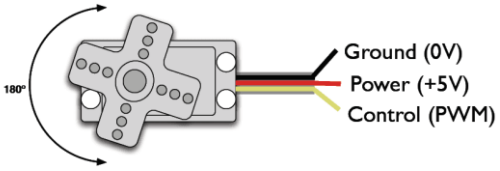


Al conectarse el sistema y ejecutarse la aplicación en la pantalla se nos muestra.

15 Controlando Servos

¿Qué es un Servomotor?

Un servomotor es un motor con un mecanismo y algoritmo de control, es decir este dispositivo mecánico posee internamente **un controlador que posiciona** precisamente el rotor del motor en un ángulo especificado por la señal entrante de control. Sin embargo el servomotor que posee este controlador es el de 90 ° o el de 180°




Servo

Para entender esto, veamos que físicamente el servomotor se compone de tres cables: GND, Power (5v) y Control. Es decir que los dos primeros se usan como alimentación en cuanto el **cable de control** es usado para enviar la señal de posicionamiento del motor con una secuencia de pulsos PWM.

Un servo con **MicroPython** implementado en una Raspberry Pi Pico o en un ESP8266 podrá tener una potencialidad tremenda, dado que la propia placa será la encargada de enviar esa señal PWM para posicionar el motor en una posición en grados en específico.

Servos de rotación continua

Los servo motores de 360° son servos Radio control estándar que se han modificado para ofrecer un **control de velocidad** en lazo abierto en lugar de su **control habitual de posición** de lazo cerrado, por lo tanto aquí ya no puedes especificar con tu raspberry pi pico la posición especifica que deseas mover el rotor del motor, sino que por el contrario estableces la velocidad de giro.

La modificación los convierte en **motores con un sistema de motoreducción** que le permite aumentar su **fuerza** y **disminuir la velocidad**, donde adicionalmente se puede controlar el **sentido de giro**, todo eso integrado en un paquete compacto y económico.

Señal de Control de un ServoMotor con MicroPython

El control de un servomotor lo podemos lograr de forma sencilla con nuestro microcontrolador RP2040 de la Raspberry Pi Pico o el ESP, usando los módulos PWM programados a través de **MicroPython** tal y como lo estudiamos en entradas pasadas.

Importamos la clase **Pin** y **PWM** del módulo machine y el módulo **utime**.

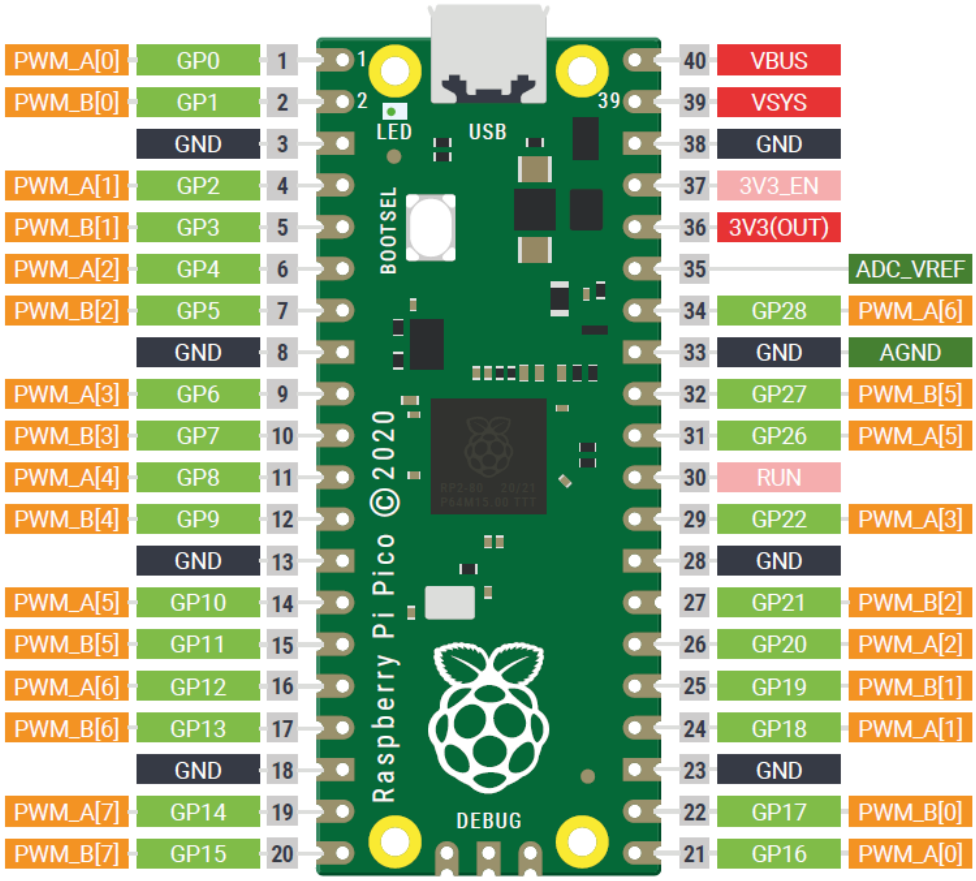
```
from machine import Pin, PWM
```

```
import utime
```

Generalmente los servomotores comerciales procesan un pulso con un **periodo de 20ms** para conseguir efectuar todo el movimiento del servo adecuadamente bien sea de 90° o 180° para posición o de 360° para velocidad.

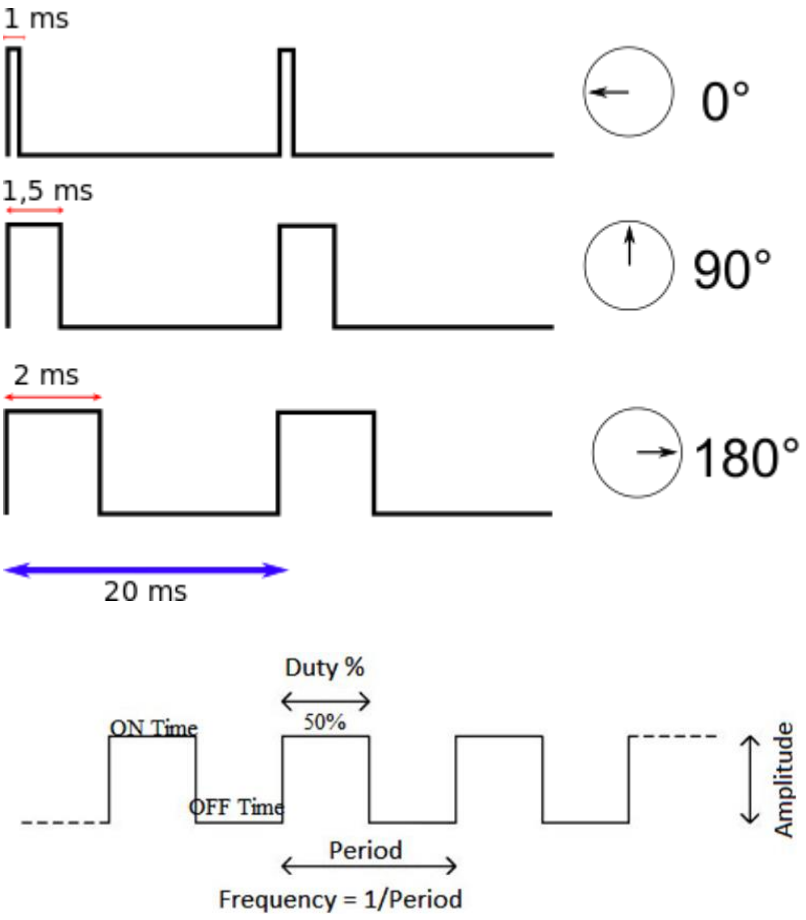
En otras palabras debemos establecer una frecuencia de **50Hz** en nuestro servomotor en cualquiera de los Pines PWM de nuestro microcontrolador.

¿A que pines podemos conectar nuestro servo?



Modulación de ancho de pulso PWM

Los servomotores funcionan por medio de modulación de ancho de pulso –*pulse-width modulation* (PWM)- Para los servos para modelismo, la frecuencia usada para mandar la secuencia de pulsos al servomotor es de 50 Hz -esto significa que cada ciclo dura 20 ms- Las duraciones de cada pulso se interpretan como comandos de posicionamiento del motor, mientras que los espacios entre cada pulso son despreciados. En la mayoría de los servomotores los anchos de pulso son de 1 ms a 2 ms, que cuando son aplicados al servomotor generan un desplazamiento de -90° a +90° por lo que, de una manera más sencilla, el ángulo de giro está determinado por el ancho de pulso; si el ancho de pulso fuera de 1.5 ms, el motor se posicionará en la parte central del rango – a 0°-



La siguiente imagen representa una señal PWM con un ciclo de trabajo del 50%. Podemos controlar el "a tiempo" de la señal PWM variando el ciclo de trabajo de 0% a 100%. Vamos a controlar un servomotor variando el ciclo de trabajo de una señal PWM en particular. Un período es el tiempo completo de encendido y apagado de una señal PWM como se muestra en la figura anterior. La **frecuencia de una señal PWM determina** qué tan rápido un PWM completa un período. Las fórmulas para calcular la frecuencia se dan a continuación

Para eso usamos la siguiente instrucción de configuración de PWM en **MicroPython** que nos permite establecer el periodo en encendido para el control de nuestros servomotores:

```
servo_180.duty_ns(100000) # 1ms -> 0 grados (180) o CW (360)
utime.sleep_ms(500)  #Tiempo para el movimiento
servo_180.duty_ns(150000) # 1.5ms -> 90 grados (180) o Center (360)
utime.sleep_ms(500)  #Tiempo para el movimiento
servo_180.duty_ns(200000) # 2ms -> 180 grados (180) o CCW (360)
utime.sleep_ms(500)  #Tiempo para el movimiento
```

Servomotores de rotación continua

A diferencia de los servomotores que rotan en un rango acotado, los servomotores de rotación continua son completamente diferentes porque los motores pueden girar 360° y

en ellos no se puede controlar la posición ni el rango de giro del motor ya que solamente puede controlarse la velocidad y el sentido de giro.

Si se quisiera modificar un servomotor de un rango limitado para que sea de rotación continua, lo primero que debe hacerse es quitar la carcasa y, mediante una secuencia de pulsos de 1.5 ms, mover el eje para que quede posicionado en la parte central y posteriormente fijarlo para que no gire más; en caso de que el servomotor tenga elementos mecánicos que acoten el giro, éstos deben ser cortados. Ahora la pregunta es, ¿por qué debe mantenerse fijo el potenciómetro? La razón por la que el potenciómetro debe estar fijo es porque el comparador siempre detectará que el eje se mantiene en la posición central todo el tiempo, y si se manda un pulso para que gire en alguno de los dos sentidos, el motor girará por alcanzar esa posición y dado que el potenciómetro ya no proporciona la señal de retroalimentación para dicha posición, el motor seguirá girando infinitamente.

Ejemplo Servomotor 180° y 360° Raspberry Pi Pico

A través de la shell de **Thonny** en **MicroPython**, vamos a pedirle al usuario el valor en grados al cual desea desplazar el servomotor y dicha señal vamos a aplicarla a un servomotor de 180° y también a un servomotor de giro completo de 360°. Este último deberá girar en sentido CW, CCW o detenerse.

Código:

```
from machine import Pin, PWM
import utime
def main():
    #Configura el Servo de 180
    servo_180 = PWM(Pin(15))
    servo_180.freq(50)

    #Configura el Servo de 360
    servo_360 = PWM(Pin(14))
    servo_360.freq(50)

    while True:
        angulo = float(input('Ingrese un ángulo: '))
        if angulo >= 0 and angulo <= 180:
            duty = int((12.346*angulo**2 + 7777.8*angulo + 700000))
            servo_180.duty_16(duty)
            servo_360.duty_ns(duty)
        else:
            print('Digite un ángulo entre 0 y 180')

if __name__ == '__main__':
    main()
```

Ejemplo Servomotor 180° y 360° ESP8266

Repetiremos el mismo ejemplo pero ahora modificaremos el código para trabajar con la NodeMCU ESP8266. En este caso necesitamos configurar el PWM para trabajar con la función **duty()** y por lo tanto debemos transformar el ciclo de trabajo en bits de 0 – 1023 en el periodo de los 20ms.

Código:

```
from machine import Pin, PWM
import utime
def main():
    #Configura el Servo de 180
    servo_180 = PWM(Pin(15))
    servo_180.freq(50)

    #Configura el Servo de 360
    servo_360 = PWM(Pin(14))
    servo_360.freq(50)

    while True:
        angulo = float(input('Ingrese un ángulo: '))
        if angulo >= 0 and angulo <= 180:
            #nanosegundos
            duty = ((12.346*angulo**2 + 7777.8*angulo + 700000))
            #milisegundos
            duty /= 1000000
            #bits
            duty = int(duty*1023/20)

            servo_180.duty(duty)
            servo_360.duty(duty)
        else:
            print('Digite un ángulo entre 0 y 180')

if __name__ == '__main__':
    main()
```

15.1. Control de Servo Basico

Objetivo

Vamos a controlar un servo mediante la implementación de una salida PWM

Funcionalidad

En la siguiente imagen se muestra el ciclo de trabajo de la señal **PWM** con la que actuamos sobe el servo. Los tiempos de encendido **Ton** y de apagado **Toff** configuran el ciclo completo, la relación entre ellos nos dará una idea del ángulo a girar

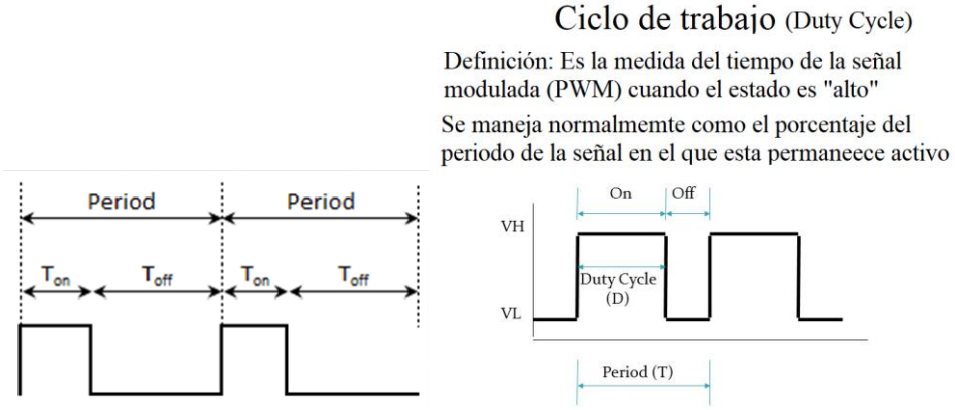


Diagrama del ciclo de trabajo El ciclo de trabajo de la señal PWM se calcula mediante la siguiente ecuación.

Período = 1/Frecuencia
Período= T encendido + T apagado
Ciclo de trabajo= T encendido / (T encendido +T apagado) * 100 (porcentaje)

Definiremos nuestro dispositivo servo con el nombre "**servo**" establecemos la frecuencia base de la señal PWM en **50 Hz** y los valores de MIN_DUTY = 1600 MAX_DUTY = 8400

Programa

Queremos que el servo gire de manera alternativa entre el ángulo equivalente a MIN_DUTY y MAX_DUTY con un retardo para cada posición de 1000 ms.

1. Cargamos las librerías en primer lugar

```
from machine import Pin, PWM
import time
```

2. Seguidamente definimos el nombre del servo y su pin, así mismo definimos la frecuencia de trabajo y los valores MIN_DUTY=1600 y MAX_DUTY=8400

```
MIN_DUTY = 1600
```

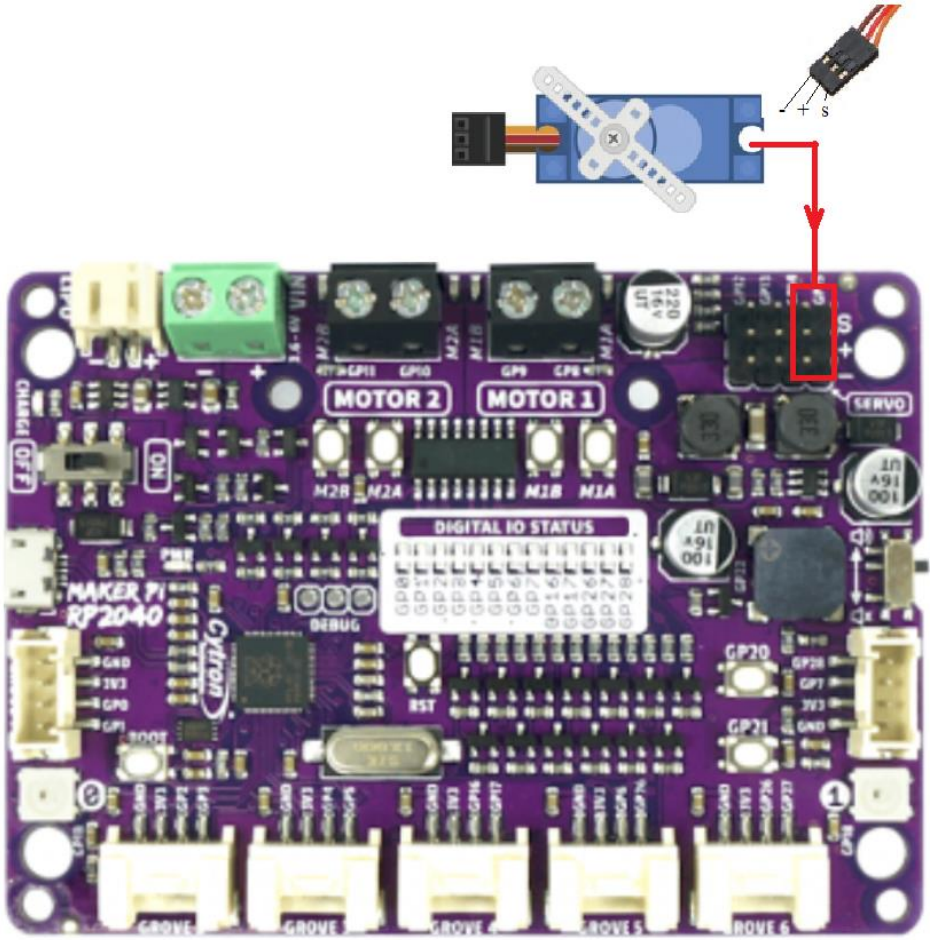
```
MAX_DUTY = 8400

servo = PWM(Pin(15))  # Conectar el servo al pin GP15
        servo.freq(50)
```

3. Dentro de **while** colocamos las dos posiciones y los retardos de espera

```
servo.duty_u16(MIN_DUTY) # Establecer ciclo de trabajo
   time.sleep_ms(1000)
   servo.duty_u16(MAX_DUTY)
   time.sleep_ms(1000)
```

```
"""
Este código de ejemplo usa: Maker Pi Pico
Referencia: www.Cytron.io/p-maker-pi-pico
Este código de ejemplo usa: Analog Micro Servo 9g (3V-6V)
Referencia: www.cytron.io/p-analog-micro-servo-9g-3v-6v
"""
from machine import Pin, PWM
import time

# ajuste los valores del ciclo de trabajo que funcionan para su
servomotor
MIN_DUTY = 1600
MAX_DUTY = 8400

servo = PWM(Pin(15))  # Conectar el servo al pin GP15
servo.freq(50)   # Configura la frecurnvia del PWM a 50Hz

while True:
   servo.duty_u16(MIN_DUTY) # Establecer ciclo de trabajo
   time.sleep_ms(1000)
   servo.duty_u16(MAX_DUTY)
   time.sleep_ms(1000)
```

Montaje de la practica



15.2. Control de Servo mediante librería “Servo”

Objetivo

Vamos a controlar un servo pero esta vez lo haremos usando una librería que nos proporcionan.

Funcionalidad

Lo que haremos básicamente e recurrir a la llamada de las funciones implementadas en la librería para controlar nuestro servo.

Usaremos la tarjeta **MAKER PI RP2040** y de las salidas de control de servos usamos la salida **GP15**.

Nuestro servo ejecutara una secuencia cíclica de 4 posiciones angulares que nosotros introduciremos mediante un array de 4 datos

La librería “**myservo”** la cargamos en el principio de nuestro programa y usamos de ella la función “**xx.servoAngle(a)**” que recoge de parámetro el valor del ángulo

Programa

1. En primer lugar cargamos las librerías, entre ellas la librería **myservo**

```
from myservo import Servo
from machine import Pin
import array
import time
```

2. Configuramos el pin del servo y lo posicionamos en 0º

```
servo=Servo(15)
servo.ServoAngle(0)
```

3. Escribimos el array de ángulos.

```
int_array = array.array('i', [0, 20, 120, 60])
```

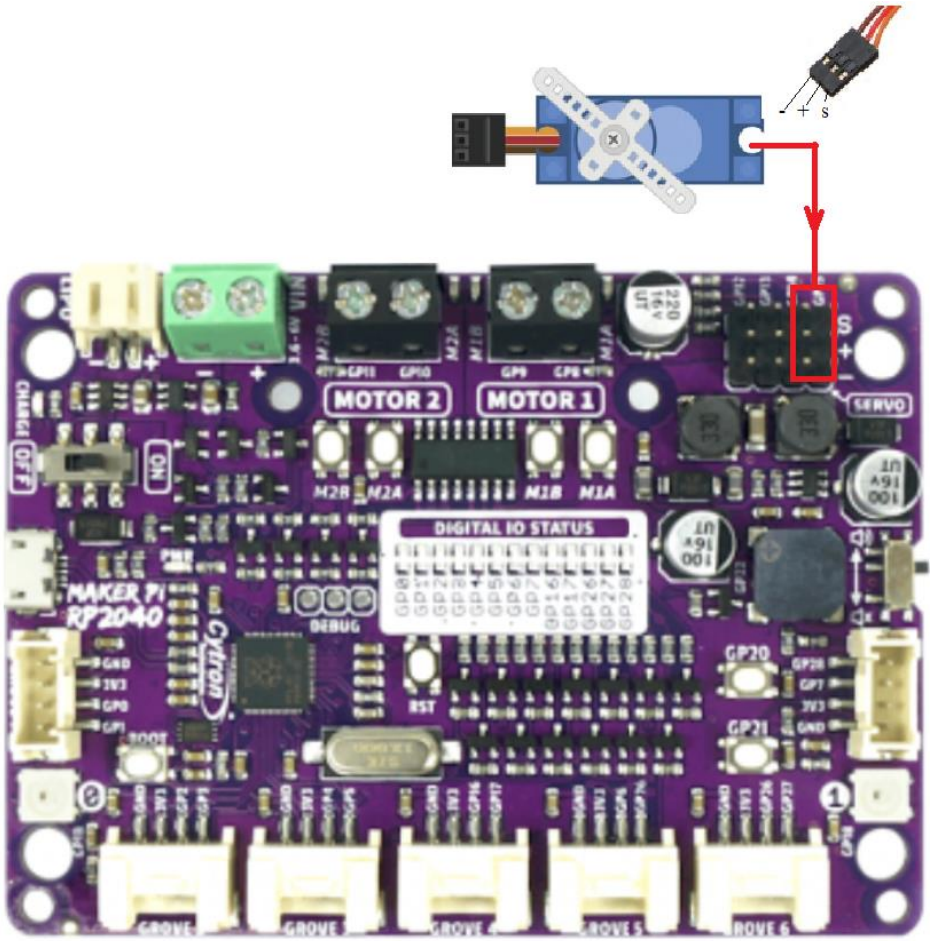
4. Seguidamente definimos la estructura **while** con control de error y dentro de ella establecemos un bucle de lectura del array con la instrucción for. Despues de posicionado en cada angulo hacemos que el servo se detenga un tiempo de 4 seg.

```
try:  (control de error)
    while True:
        for a in int_array:
            servo.ServoAngle(a)
            time.sleep_ms(4000)
except:
```

servo.deinit()

```
#Secuencia de control de posiciones de un SERVO con la librería myservo
from myservo import Servo
from machine import Pin
import array
import time
servo=Servo(15) #Servo conectado en el pin GP15
servo.ServoAngle(0) #Posición de inicio angulo=0
#Lista de posiciones
int_array = array.array('i', [0, 20, 120, 60])

try:  (control de error)
   while True:
      for a in int_array:
         servo.ServoAngle(a)
         time.sleep_ms(4000)
except
```

Montaje de la practica



Análisis y propuesta de actividades

Proponemos una variante del ejemplo visto. La variación consiste en que a la vez que se realizan los giros de posición podamos imprimir el ángulo girado en la consola de **Thonny**

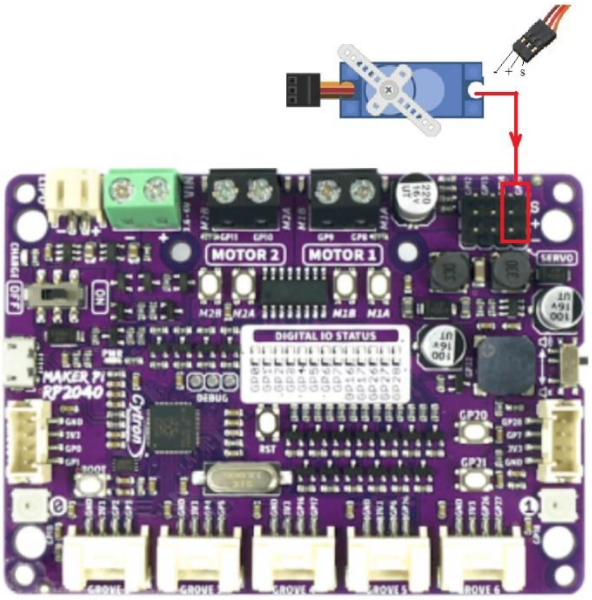
Vamos a realizar una secuencia con los siguientes valores

[0, 10, 20, 30, 40, 50, 180]

Trabajaremos con el servo conectado a **GP15** y el tiempo entre cada posición lo fijaremos a 1000 ms

Este sería el **programa**.

```
#Secuencia de control de posiciones de un SERVO con la libreria myservo
from myservo import Servo
from machine import Pin
import array
import time

servo=Servo(15) #Servo conectado en el pin GP15
servo.ServoAngle(0) #Posición de inicio angulo=0
angulos = array.array('i', [0, 10, 20, 30, 40, 50, 180])  #Lista de posiciones

try:
   while True:
      for a in angulos:
         servo.ServoAngle(a)
         print("angulo =",a)
         time.sleep_ms(1000)

except:
   servo.deinit()
```



15.3. Servo Control con Maker Pi RP2040

Objetivo

Con este ejemplo queremos controlar cuatro servos mediante la tarjeta MAKER PI RP2040

Funcionalidad

Vamos a conectar cuatro servos a los pines **GP15**, **GP14**, **GP13** y **GP12** que vamos a controlar a la vez.

Queremos controlar el giro mediante dos pulsadores, los que están en la propia tarjeta **GP20** y **GP21** de tal manera que con cada uno de ellos hacemos que los servos se muevan en ambos sentidos de giro.

El giro será de 1º por cada vez que pulsemos el botón correspondiente. El tiempo de retardo en el pulsador para evitar rebotes será de 100 ms (**time.sleep(0.1)**)

Programa

1. En primer lugar cargamos las librerías, entre ellas la librería **myservo**

```
import machine
import utime
import time
```

2. Configuramos el pin del servo y los botones 1 y 2

```
button1 = machine.Pin(20, machine.Pin.IN, machine.Pin.PULL_DOWN)
button2 = machine.Pin(21, machine.Pin.IN, machine.Pin.PULL_DOWN)
# Configuracion de pines para los Servos
pwm1 = machine.PWM(machine.Pin(12))
pwm2 = machine.PWM(machine.Pin(13))
pwm3 = machine.PWM(machine.Pin(14))
pwm4 = machine.PWM(machine.Pin(15))
```

3. Configuramos los parámetros de los servos frecuencia, y ciclo de trabajo.

```
pwm1.freq(50)
pwm2.freq(50)
pwm3.freq(50)
pwm4.freq(50)

angle = 0.0
min_dutycycle = 2200
max_dutycycle = 8300
dutycycle = 0
```

4. Seguidamente definimos la estructura **while** en el que lo que hacemos es testear la pulsación d ellos botones **Boton1 GP20 y Boton2 GP21** con la ayuda de sendas instrucciones condicionales tipo **if.** La variable que se incrementara o decrementara será la variable **“angle”.** Para convertir este valor en la auténtica variable que recogerá el servo **dutycycle** lo hacemos medinte la fórmula:

dutycycle *= int(((max_dutycycle - min_dutycycle) / 180) * angle) + min_dutycycle*

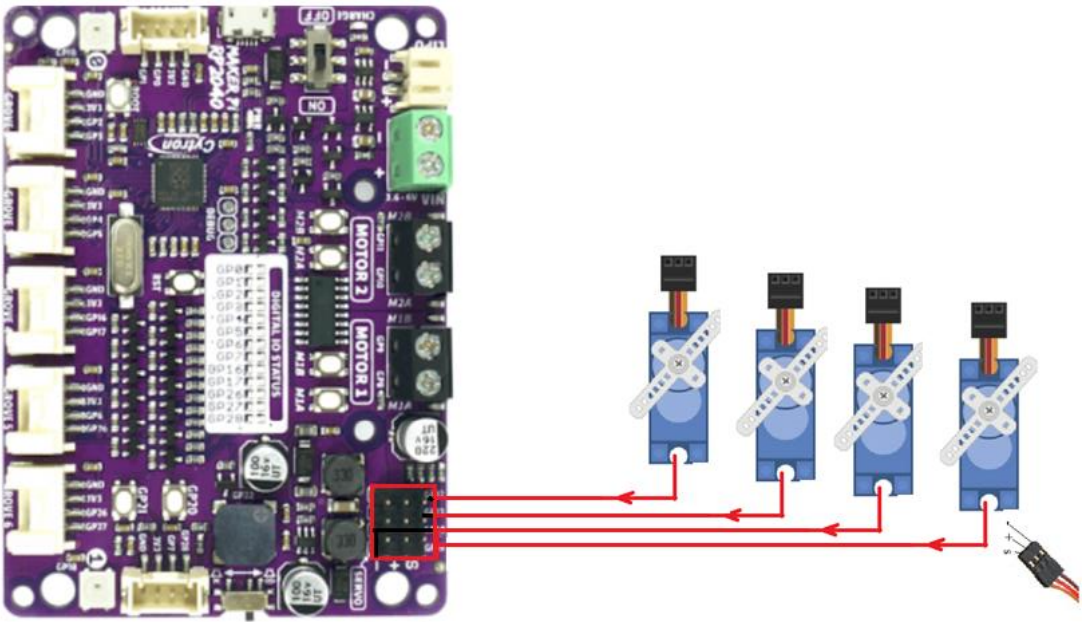
En el principio de las instrucciones incluidas en el bucle while pondremos una isntruccion para imprimir el valor del ángulo

print("Angulo",angle,"Ciclo de trabajo", dutycycle)

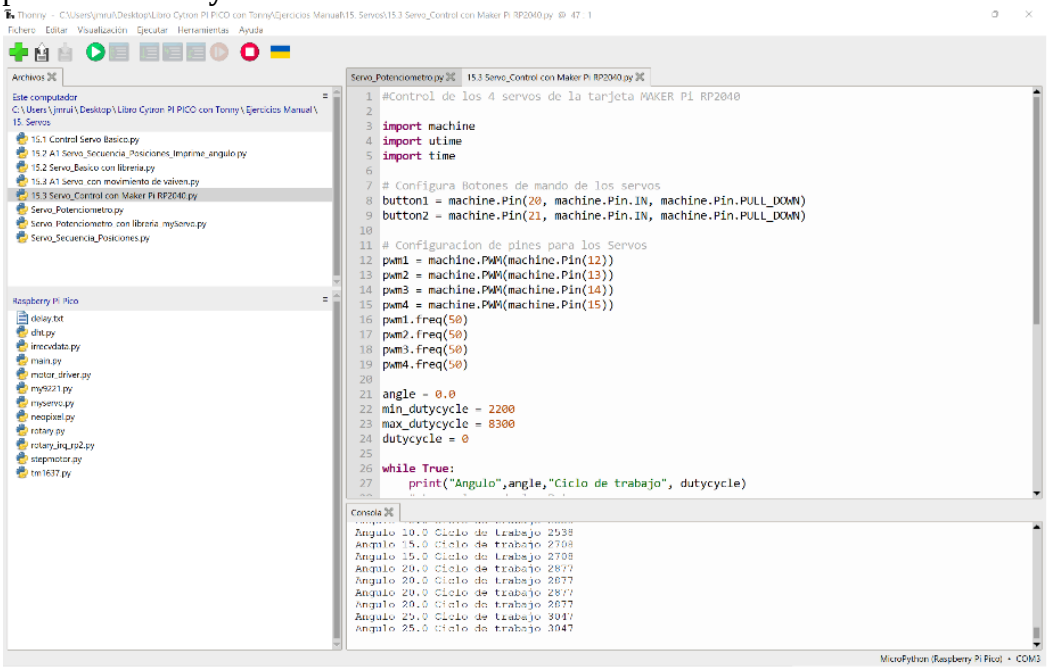
```
#Control de los 4 servos de la tarjeta MAKER Pi RP2040

import machine
import utime
import time

# Configura Botones de mando de los servos
button1 = machine.Pin(20, machine.Pin.IN, machine.Pin.PULL_DOWN)
button2 = machine.Pin(21, machine.Pin.IN, machine.Pin.PULL_DOWN)

# Configuracion de pines para los Servos
pwm1 = machine.PWM(machine.Pin(12))
pwm2 = machine.PWM(machine.Pin(13))
pwm3 = machine.PWM(machine.Pin(14))
pwm4 = machine.PWM(machine.Pin(15))
pwm1.freq(50)
pwm2.freq(50)
pwm3.freq(50)
pwm4.freq(50)

angle = 0.0
min_dutycycle = 2200
max_dutycycle = 8300
dutycycle = 0

while True:
    print("Angulo",angle,"Ciclo de trabajo", dutycycle)
    # Lee valores de los Botones.
    if button1.value():
        angle += 5
        time.sleep(0.1)
    if button2.value():
        angle -= 5
        time.sleep(0.1)
    # Limita el angulo al rango 0 a 180 grados.
    if angle > 180:
```

```
    angle = 180.0
if angle < 0:

    angle = 0.0


dutycycle = int(((max_dutycycle - min_dutycycle) / 180) * angle) + min_dutycycle
pwm1.duty_u16(dutycycle) #Mueve Servo conectado en el GP12
pwm2.duty_u16(dutycycle) #Mueve Servo conectado en el GP13
pwm3.duty_u16(dutycycle) #Mueve Servo conectado en el GP14
pwm4.duty_u16(dutycycle) #Mueve Servo conectado en el GP15
utime.sleep(0.01)
```

Montaje de la practica



Aspecto de **Thonny**

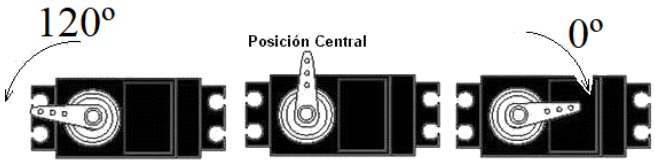


Análisis y propuesta de actividades

Se propone una variacion del programa anterior en el que queremos que el servo realice un movimiento de vaivén recorriendo un arco equivalente a 120º.

Queremos que el servo gire en sentido de las agujas del reloj en pasos de 1º hasta llegar a 120º. Seguidamente queremos que el giro sea igual pero en sentido contrario.

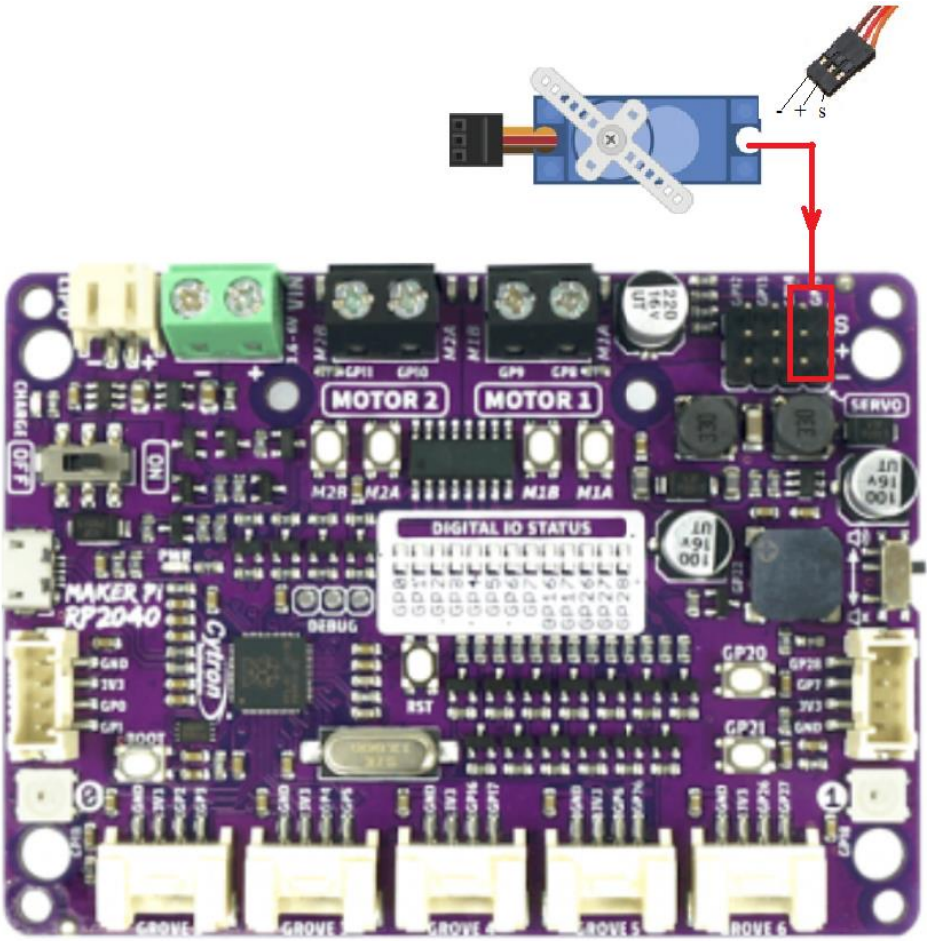
El tiempo entre paso y paso queremos que sea de 15 ms (**time.sleep_ms(15)**)



El programa.

```
from myservo import Servo
import time

servo=Servo(15)
servo.ServoAngle(0)
time.sleep_ms(1000)

try:
   while True:
      for i in range(0, 120, 1):
         servo.ServoAngle(i)
         time.sleep_ms(15)
      for i in range(120, 0, -1):
         servo.ServoAngle(i)
         time.sleep_ms(15)
except:
   servo.deinit()
```

Montaje


S
MOTOR 2
MOTOR 1
SERVO
DIGITAL IO STATUS
MAKER PI
RP2040
DEBUG
RST
GP20
GP21
GROVE

15.4. Control Servo con un Potenciómetro

Objetivo

Controlar el ángulo de giro de un servo con la ayuda de un potenciómetro.

Funcionalidad

Se trata de convertir el giro de nuestro potenciómetro en un valor que enviaremos a la función de giro del servo como parámetro. Usaremos un potenciómetro conectado en el pin GP27 de la tarjeta MAKER PI RP2040. El servo se conectará a los terminales correspondientes a la salid GP15.

Para el control del servo haremos uso de la librería "myservo"

El valor leído de nuestro canal analógico GP27 se debe mapear al valor de 0 a 180 ya que haremos girar el servo en ese rango de valor, para ello recurrimos a un sencillo cálculo:

```
angle = (adcValue * 180) / 65535
```

La función que mueve el angulo del serbvo es:

```
servo.ServoAngle(0)
```

Programa

La realización del programa se ajustará al siguiente orden de escritura del programa.

1. Cargamos las librerías

```
from myservo import Servo
from machine import ADC,Pin
import time
```

2. Definimos los pines que usaremos y situamos la posición angular en 0º

```
servo=Servo(15)
adc=ADC(27)
servo.ServoAngle(0)
```

3. Controlaremos errores con la estructura:

```
try:

….

Except:
```

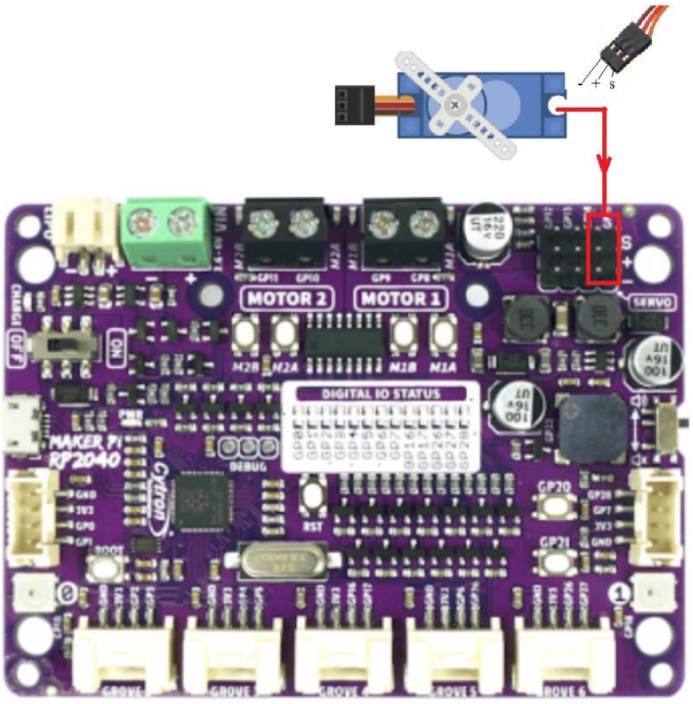
Servo.deinit()

4. En la estructura **while** leeremos el valor del canal analógico y lo asociamos al valor "**angle**" para realizar el encalado del ángulo de 0º a 180º. Incluimos un retardo y la impresión del valor de la variable ángulo en la consola de **Thonny**

Programa completo

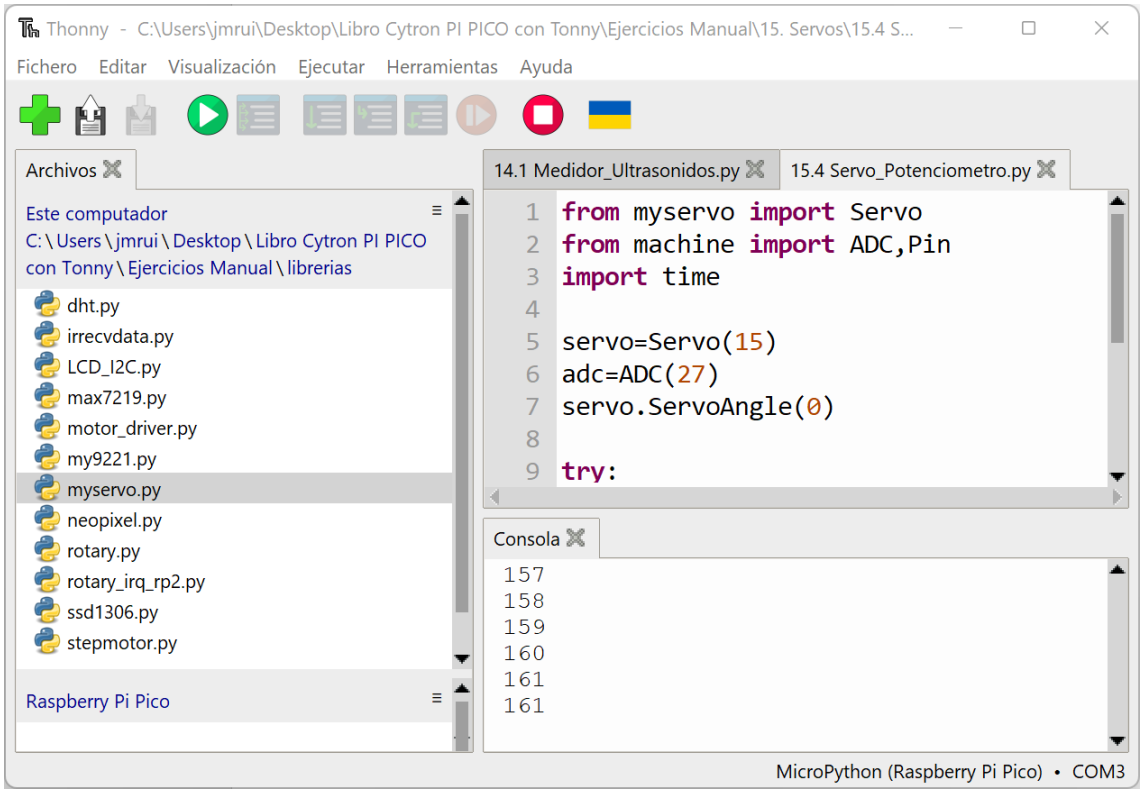
```
from myservo import Servo
from machine import ADC,Pin
import time

servo=Servo(15)
adc=ADC(27)
servo.ServoAngle(0)

try:
    while True:
        adcValue = adc.read_u16()
        angle = int((adcValue * 180) / 65535)
        servo.ServoAngle(int(angle))
        time.sleep_ms(50)
        print(angle)
except:
    servo.deinit()
```

Montaje de la practica



Análisis y propuesta de actividades

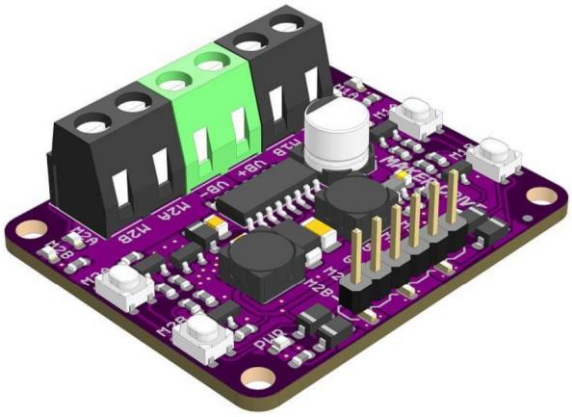
Cuando ejecutamos el programa podemos ver en la consola de **Thonny** los valores angulares



16 Motores de cc

Para el control de los motores de cc necesitamos utilizar una unidad controladora que permita el consumo de corriente de estos dispositivos a la vez que el control de velocidad y sentido de giro de estos.

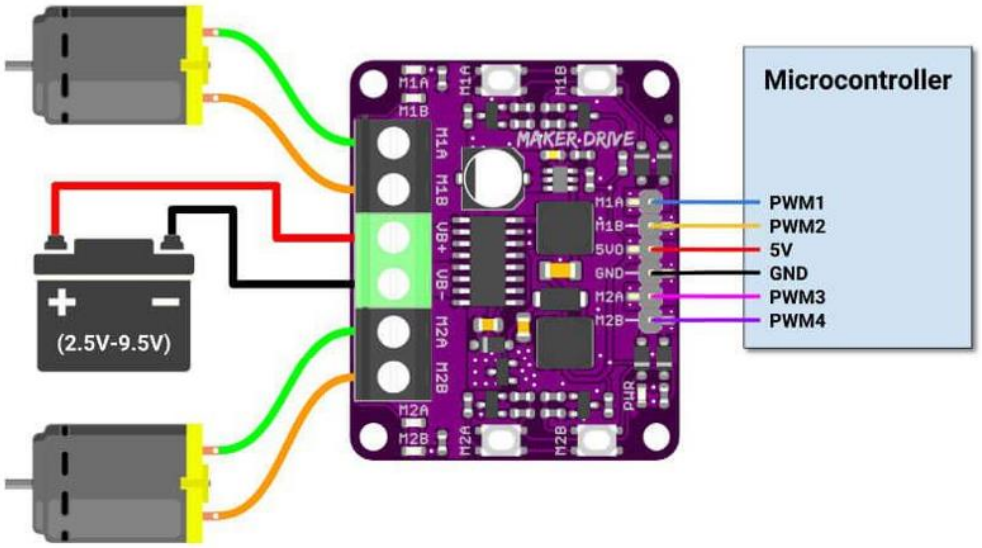
Cytron tiene una unidad controladora que podemos usar de manera autonoma y a traves de los pines de salida de la tarjeta Raspbery Pi Pico.



Maker Drive: Simplifying H-Bridge Motor Driver for Beginner

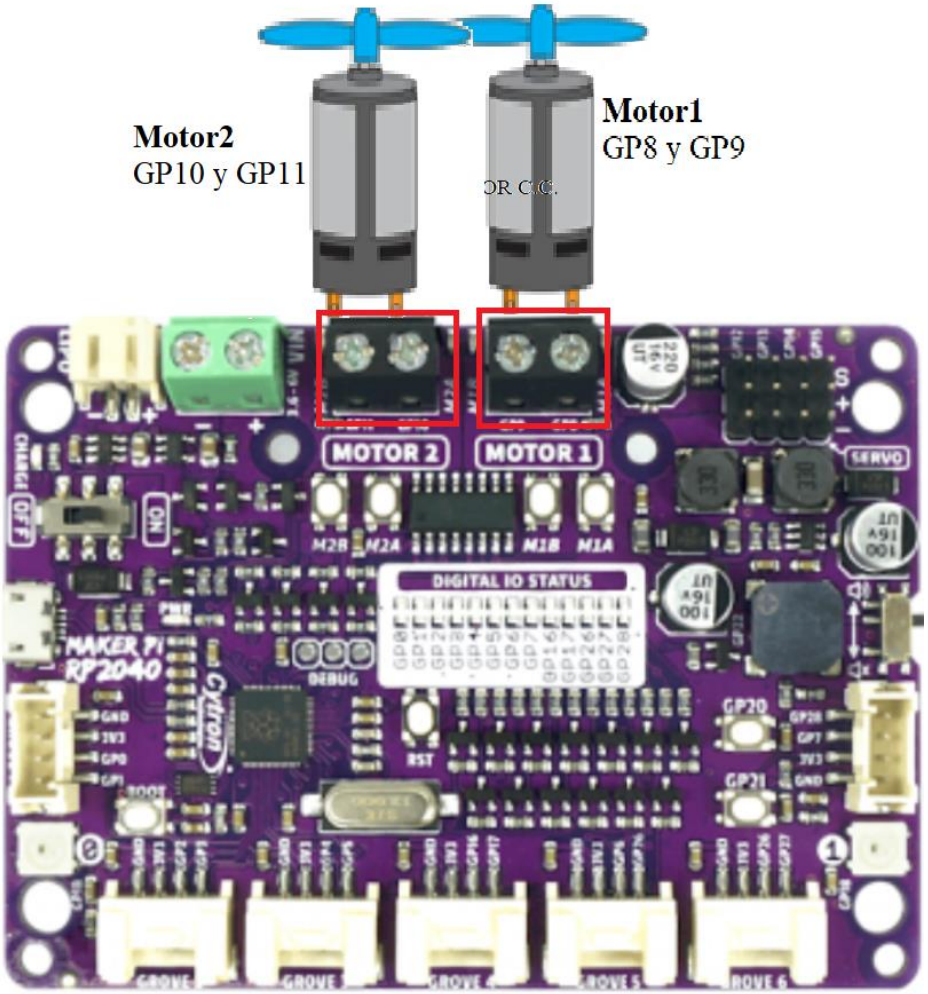
En la siguiente figura vemos el conexionado de la etapa de potencia , con motores y alimentación de esta unidad.

La parte de control como vemos estará en manos de los pines de la derecha que se llevaran a la unidad MAKER PI PICO



Si trabajamos con la tarjeta MAKER PI PICO podemos ver algunos eemplos en el siguiente LINK. Entre ellos hay uno que nos muestra como controlar motores con esta unidad de.

Nuestros ejemplos los vamos a implementar con la tarjeta MAKER PI RP2040 que trae ya implementado el controlador de motores cc.



16.1 Control de dos Motores de cc con la tarjeta Maker Pi RP2040

Objetivo

Controlar un motor de cc a través de la tarjeta MAKER PI RP2040

Funcionalidad

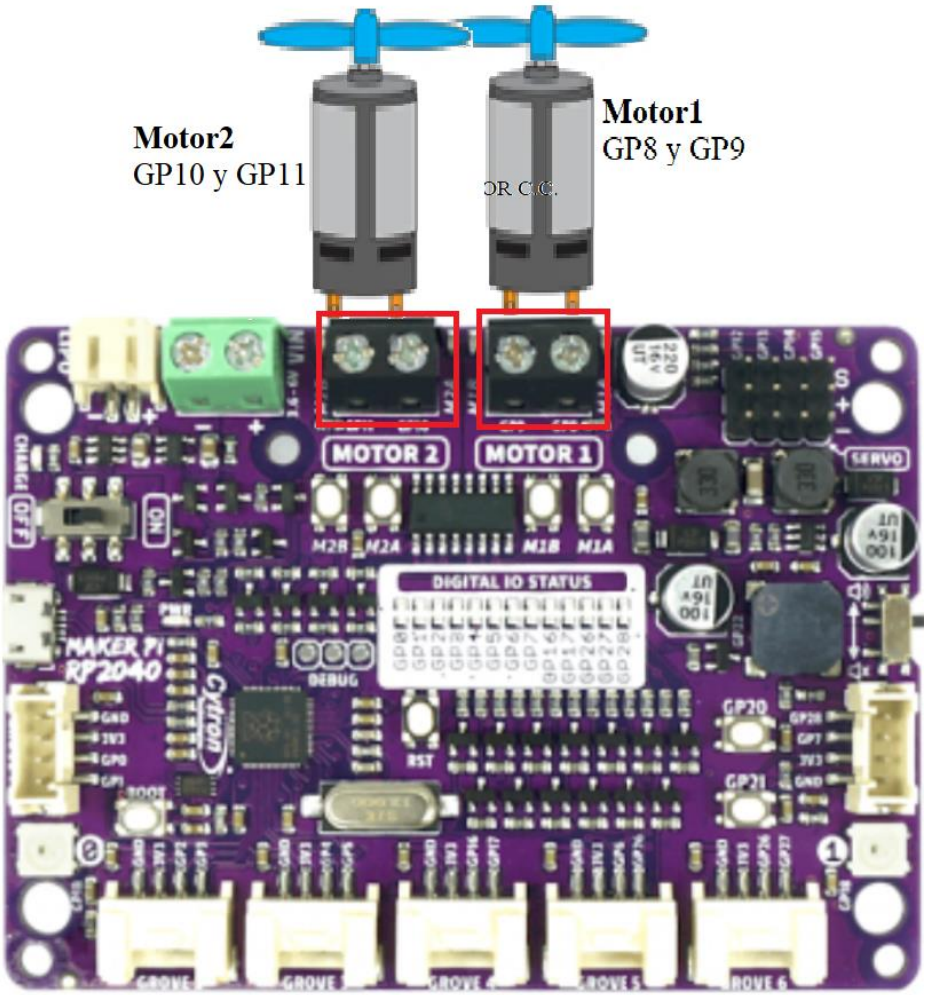
Para realizar este programa usaremos la librería "**motor**" que nos permitirá controlar los dos motores acoplables a la tarjeta.

Con la función "**speed**" podemos mover los motores modificando los parámetros de la función. La función "**brake**" detiene el motor

Programa

```
"""
Control de dos motores MOTOR1 y MOTOR2
Con la tarjeta MAKER PI RP2640
Usando una librería
"""

from motor_driver import * # importar librería motor_driver

# Activación de dos Motores Motor1 y Motor2
# M1A = 8, M1B = 9, M2A = 10, M2B = 11
motor = motor_driver(8,9,10,11)
# Funcion speed(self,speedLeft, speedRight). La máxima velocidad 100

motor.speed(50,50)    # mueve adelante motores a velocidad 50
utime.sleep(5)       # se detiene 5 segundos

motor.speed(0,50)     # Gira izquierda a una velocidad de 50
utime.sleep(5)

motor.speed(50,0)    # Gira derecha a una velocidad 50
utime.sleep(5)

motor.brake()        # Frena motores
```

Montaje de la practica



16.2 Control de un Motor de cc. sin usar librería

Objetivo

Escribir un programa que nos permita controlar el movimiento de un motor de cc.

Funcionalidad

Se trata de escribir una secuencia de movimientos que se ajuste a la tabla siguiente:

1	Mover hacia adelante lentamente	Durante 1 seg
2	Parar	Durante 1 seg.
3	Mover hacia adelante	Durante 1 seg.
4	Parar	Durante 1 seg.
5	Mover hacia Atrás	Durante 1 seg.
6	Parar	Durante 1 seg.
7	Mover atrás lentamente	Durante 1 seg.
8	Frenar	Durante 1 seg.

Usaremos un motor conectado a la correspondiente salida para motor de la tarjeta MAKER PI RP2040 Motor 1. Lo haremos se usar ninguna librería expresamente de control de motores.

Haremos uso de las instrucciones:

```
M1A.duty_u16(xx)
M1B.duty_u16(xx)
```

cuyo parámetro “xx” es el valor comprendido entre 0 y 65535 que se encargara de enviar una tensión variable en modo PWM a las salidas de control:

```
M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))
```

Programa

La construcción del programa se realizará de acuerdo a los siguientes pasos:

1. Cargar las librerías

```
import machine
import utime
```

2. Definir los pines de conexión M1A y M1B y consignar la frecuencia de la señal base de los PWM

```
M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))
M1A.freq(50)
M1B.freq(50)
```

3. Escribir cada uno de los movimientos con el formato repetitivo siguiente:

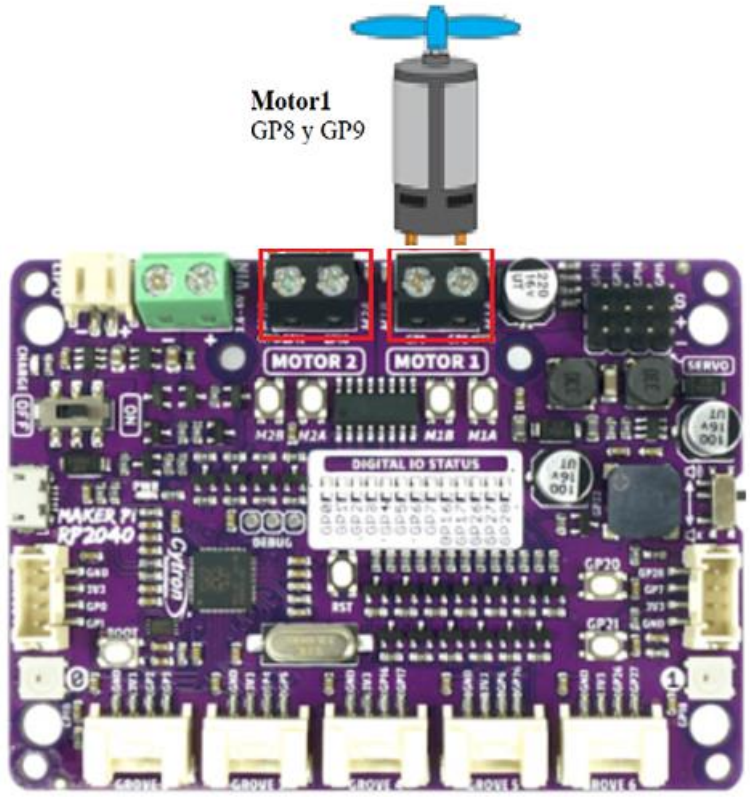
```
    print("Adelante lentamente")
    M1A.duty_u16(0)
    M1B.duty_u16(30000)
    utime.sleep(1)
```

Listado completo del programa

```
#Controlando Motor1 sin usar librería especial
import machine
import utime

# Setup DC Motor pins
M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))

M1A.freq(50)
M1B.freq(50)

while True:
    print("Adelante lentamente")
    M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
    M1B.duty_u16(30000)
    utime.sleep(1)

    print("Parar")
    M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
    M1B.duty_u16(0)
    utime.sleep(1)

    print("Adelante")
    M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
    M1B.duty_u16(65535)
    utime.sleep(1)

    print("Parar")
    M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
```

```
M1B.duty_u16(0)
utime.sleep(1)

print("Atras")
M1A.duty_u16(65535)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
utime.sleep(1)

print("Parar")
M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
utime.sleep(1)

print("Atras Lentamente")
M1A.duty_u16(30000)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
utime.sleep(1)

print("Freno")
M1A.duty_u16(65535)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(65535)
utime.sleep(1)
```

Montaje de la practica



Análisis y propuesta de actividades

- Probar el funcionamiento del programa observando como en la consola de **Thonny** aparece el movimiento que se realiza.

- Modificar los tiempos entre cada movimiento así como comprobar los cambios cuando se modifican los valores de la velocidad

```
M1A.duty_u16(xx)
M1B.duty_u16(xx)
utime.sleep(1)
```

16.3 Control de dos Motores

Objetivo

Controlar dos motores de un robot para situarlo en distintas posiciones usando el mismo programa anterior

Funcionalidad

Se trata de realizar el mismo programa anterior pero esta vez usando dos motores que conectamos a los terminales Motor1 y Motor 2 de la tarjeta

1	Mover M1 y M2 hacia adelante lentamente	Durante 1 seg
2	Parar M1 y M2	Durante 1 seg.
3	Mover M1 y M2 hacia adelante	Durante 1 seg.
4	Parar M1 y M2	Durante 1 seg.
5	Mover M1 y M2 hacia Atrás	Durante 1 seg.
6	Parar M1 y M2	Durante 1 seg.
7	Mover M1 y M2 atrás lentamente	Durante 1 seg.
8	Frenar M1 y M2	Durante 1 seg.

Programa

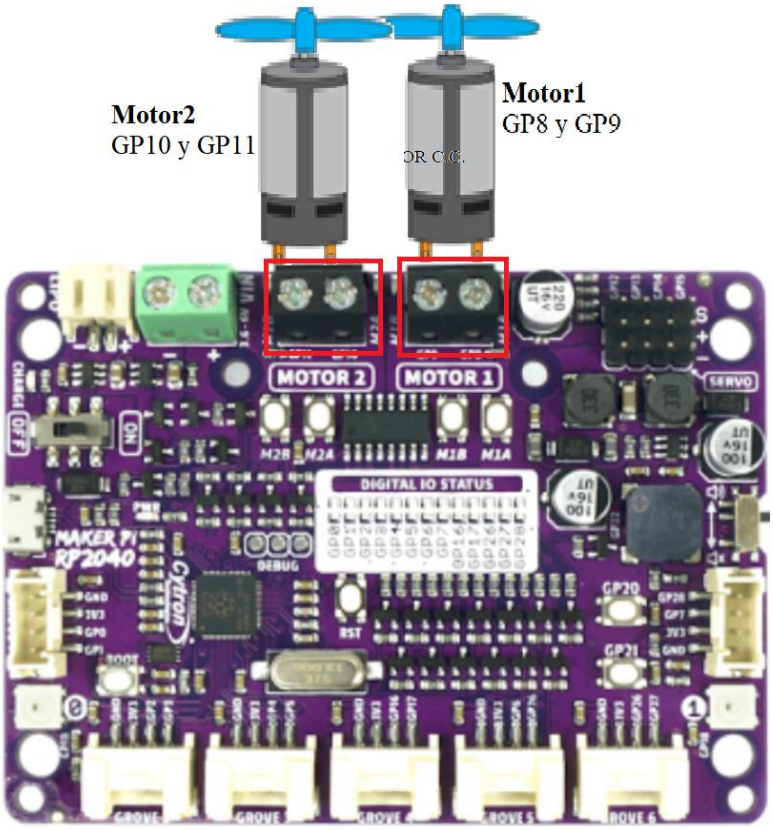
La explicación del programa la omitimos dado que es igual que el anterior

```
#Control Motores de la tarjeta MAKER Pi RP2040
#Controlando los motores Motor1 y Motor2
import machine
import utime

# Configuracion de los pines
M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))
M2A = machine.PWM(machine.Pin(10))
M2B = machine.PWM(machine.Pin(11))
M1A.freq(50)
M1B.freq(50)
M2A.freq(50)
M2B.freq(50)

while True:
```

```
print("Adelante Lento")
M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(30000)
M2A.duty_u16(0)
M2B.duty_u16(30000)
utime.sleep(1)

print("Parar")
M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
M2A.duty_u16(0)
M2B.duty_u16(0)
utime.sleep(1)

print("Adelante")
M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(65535)
M2A.duty_u16(0)
M2B.duty_u16(65535)
utime.sleep(1)

print("Parar")
M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
M2A.duty_u16(0)
M2B.duty_u16(0)
utime.sleep(1)

print("Atras")
M1A.duty_u16(65535)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
M2A.duty_u16(65535)
M2B.duty_u16(0)
utime.sleep(1)

print("Parar")
M1A.duty_u16(0)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
M2A.duty_u16(0)
M2B.duty_u16(0)
utime.sleep(1)

print("Atras Lento")
M1A.duty_u16(30000)     # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(0)
M2A.duty_u16(30000)
M2B.duty_u16(0)
utime.sleep(1)

print("Frenar")
```

```
M1A.duty_u16(65535)    # El ciclo de trabajo debe estar entre 0 y 65535
M1B.duty_u16(65535)
M2A.duty_u16(65535)
M2B.duty_u16(65535)
utime.sleep(1)
```

Montaje de la practica



Análisis y propuesta de actividades

- Probar el funcionamiento del programa observando como en la consola de **Thonny** aparece el movimiento que se realiza.

- Modificar los tiempos entre cada movimiento así como comprobar los cambios cuando se modifican los valores de la velocidad

```
M1A.duty_u16(xx)
M1B.duty_u16(xx)
M2A.duty_u16(xx)
M2B.duty_u16(xx)
utime.sleep(1)
```

16.4. Control de velocidad de un motor con un potenciómetro

Objetivo

Con esta práctica vamos a experimentar con la variación de la velocidad de un motor.

Funcionalidad

Se trata de conectar un motor **M1** a los terminales de la tarjeta **MAKER PI RP2040** que se gobierna en los pines **M1A GP8** y **M1B GP9.**

Trabajamos con señales PWM modificando el valor de salida de los mencionados pines M1A y M1B con la ayuda de la señal analógica que estableceremos en el pin ADC(27) mediante la conexión de un potenciómetro

Programa

Seguiremos los siguientes pasos

1. Importar las librerías

```
from machine import ADC, Pin
import time
```

2. Designar la entrada analógica ADC(27) y los pines de salida PWM M1A y M1B

```
pot = ADC(27) #Lectura de señal analógica
M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))
```

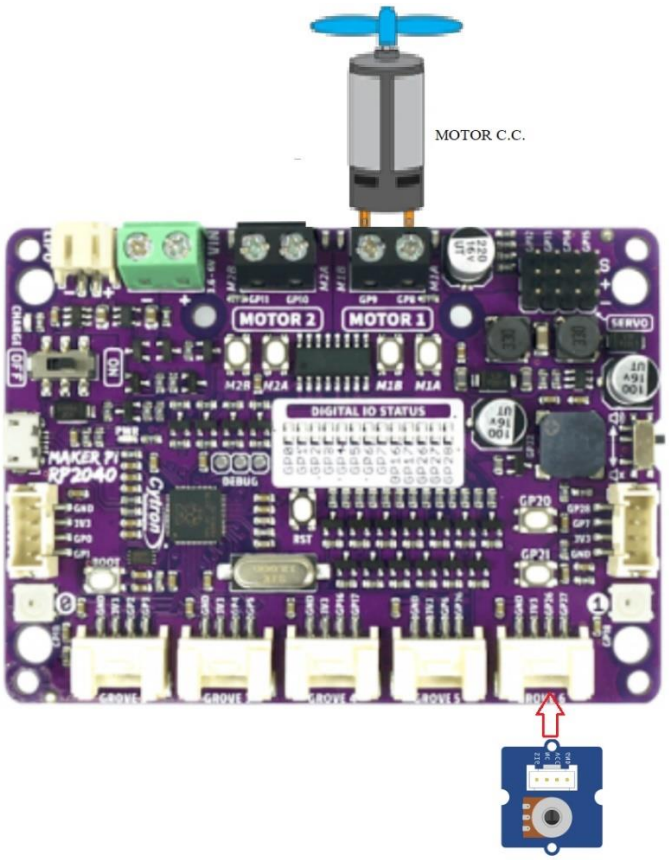
3. Fijamos la frecuencia de trabajo de la señal PWM

```
M1A.freq(50)
M1B.freq(50)
```

4. En la parte **while** asociamos la variable velocidad al valor leído del canal analógico, imprimimos la velocidad en la consola de **Thonny** enviamos el valor de velocidad al pin de salida M1B y ponemos un retardo.

```
#Control de la velocidad de un Motor
from machine import ADC, Pin
import time
pot = ADC(27) #Lectura de señal analógica

M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))

M1A.freq(50)
M1B.freq(50)

while True:
    velocidad = pot.read_u16()
    print("velocidad", velocidad)
    # El ciclo de trabajo debe estar entre 0 y 65535
    M1A.duty_u16(0)
    M1B.duty_u16(velocidad)
    time.sleep(0.1)
```

Montaje de la practica



16.5. Control de velocidad y sentido de giro de un motor

Objetivo

Vamos a controlar la velocidad y el sentido de giro de un motor de cc.

Funcionalidad

Usaremos un potenciómetro que conectaremos en el pin GP27 en el que obtendremos una señal analógica variable que nos servirá para modificar la velocidad del motor.

Para el control del sentido de giro lo haremos con un pulsador conectado en el GP17

Queremos monitorizar también el sentido de giro mediante el uso de dos salidas digitales a las que conectaremos unos LEDs, GP3 y GP5

Imprimiremos el valor de la velocidad en % de tal manera que para ello tendremos que mapear el valor de la "velocidad"

Programa

El programa se construirá de acuerdo a las siguientes etapas

1. Cargar las librerías

```
from machine import ADC, Pin
import time
```

2. Definir las entradas y salidas implicadas

```
sentido_giro = Pin(17,Pin.IN)
derecha = Pin(3, Pin.OUT) # GP3 Derecha
izquierda = Pin(5, Pin.OUT) # GP5 Izquierda
pot = ADC(27) #Lectura de señal analógica
M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))
```

3. Fijar condiciones de inicio

```
M1A.freq(50)
M1B.freq(50)

derecha.value(0)
izquierda.value(0)
```

4. Definir la función map para escalar el valor de la velocidad entre 0 y 100

```
def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)
```

5. En el bucle **while** lo que haremos será leer el valor de la velocidad y mapearlo seguidamente testeamos el valor de la entrada sentido de giro y en función de este valor colocamos los valores "**0**" o "**velocidad**" en las salidas M1A y M1B de control del motor M1 de la tarjeta. Además de activar la salida de sentido de giro GP3 y GP5 escribiremos la velocidad en % y el sentido en la consola de **Thonny**.

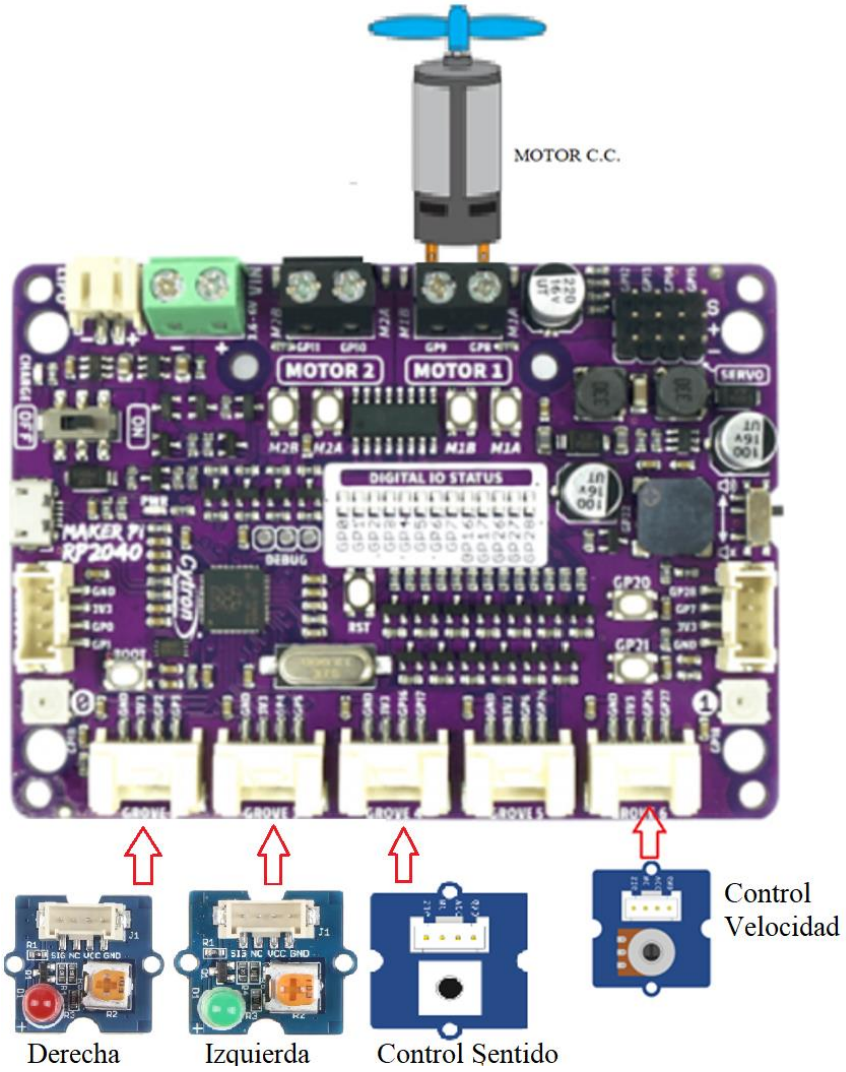
```
velocidad = pot.read_u16()
   velocidad_porcentual = map(velocidad, 0, 65535, 0, 100)

   if sentido_giro.value() == 1:
      # El ciclo de trabajo debe estar entre 0 y 65535
      M1A.duty_u16(0)
      M1B.duty_u16(velocidad)
      derecha.value(0)
      izquierda.value(1)
      print('Velocidad {:.0f}'.format(velocidad_porcentual), "Derecha")
      time.sleep(0.1)
   else:
      M1A.duty_u16(velocidad)
      M1B.duty_u16(0)
      time.sleep(0.1)
      derecha.value(1)
      izquierda.value(0)
      print('Velocidad {:.0f}'.format(velocidad_porcentual), "Izquierda")
      time.sleep(0.1)
```

El programa completo sería el siguiente

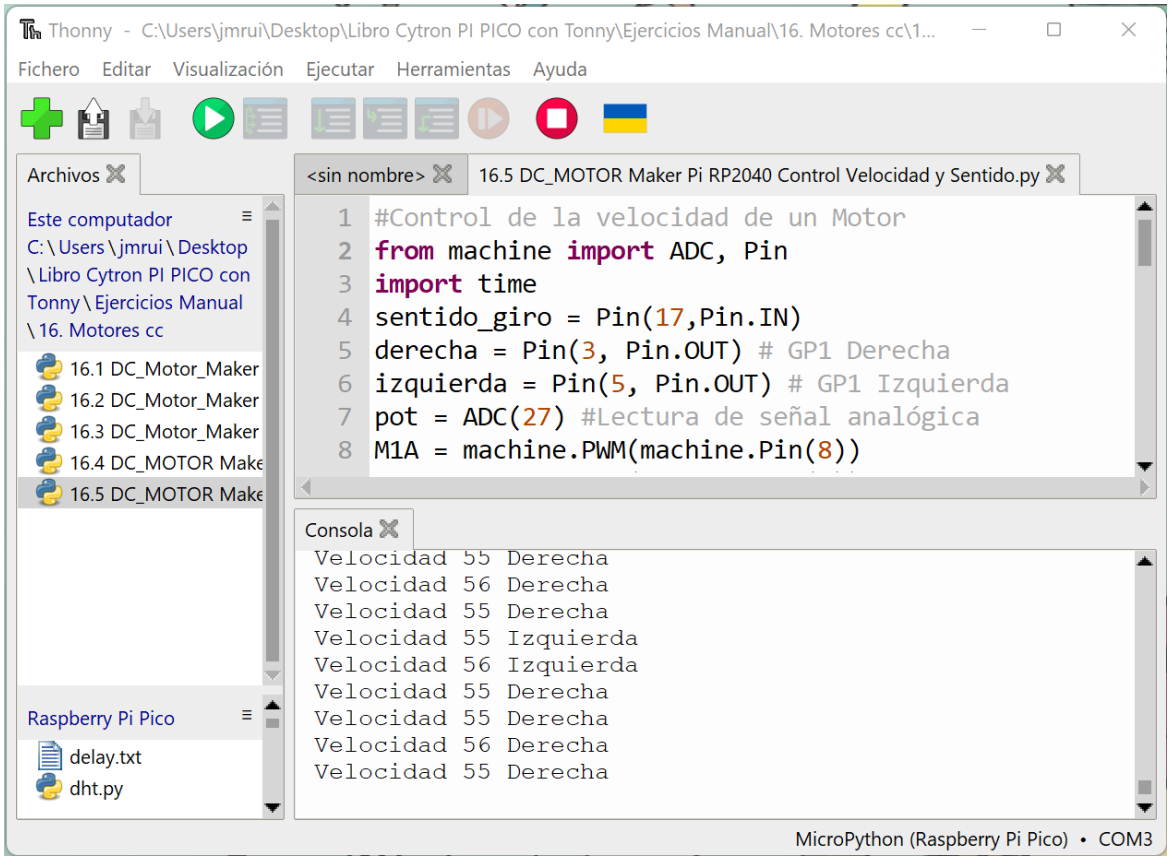
```
# Control de la velocidad de un Motor
from machine import ADC, Pin
import time
sentido_giro = Pin(17,Pin.IN)
derecha = Pin(3, Pin.OUT) # GP3 Derecha
izquierda = Pin(5, Pin.OUT) # GP5 Izquierda
pot = ADC(27) #Lectura de señal analógica
M1A = machine.PWM(machine.Pin(8))
M1B = machine.PWM(machine.Pin(9))

M1A.freq(50)
M1B.freq(50)

derecha.value(0)
izquierda.value(0)

def map(s, a1, a2, b1, b2):
    return b1 + (s - a1) * (b2 - b1) / (a2 - a1)

while True:
   velocidad = pot.read_u16()
   velocidad_porcentual = map(velocidad, 0, 65535, 0, 100)
```

```
if sentido_giro.value() == 1:
    # El ciclo de trabajo debe estar entre 0 y 65535
    M1A.duty_u16(0)
    M1B.duty_u16(velocidad)
    derecha.value(0)
    izquierda.value(1)
    print('Velocidad {:.0f}'.format(velocidad_porcentual), "Derecha")
    time.sleep(0.1)
else:
    M1A.duty_u16(velocidad)
    M1B.duty_u16(0)
    time.sleep(0.1)
    derecha.value(1)
    izquierda.value(0)
    print('Velocidad {:.0f}'.format(velocidad_porcentual), "Izquierda")
    time.sleep(0.1)
```

Montaje de la practica



Este es aspecto de **Thonny** cuando el programa esta funcionando



17. Otros dispositivos

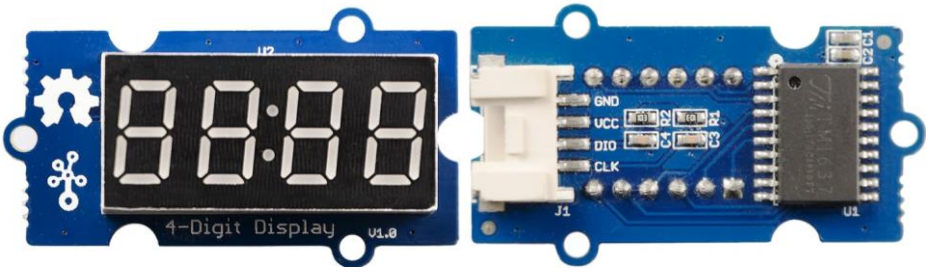
17.4. Control de Display Numérico de 4 Cifras

Objetivo

El dispositivo que vamos a usar en esta práctica es muy útil para mostrar valores numéricos en nuestras aplicaciones.

Funcionalidad

Se trata de un display formado por 4 dígitos de 7 segmentos



Los pines de conexión de este dispositivo son: Vcc, GND, DIO y CLK

En nuestro caso vamos a usar una librería que nos facilitará el manejo del dispositivo. Se trata de la librería "**tm1637**".

Para la prueba configuraremos el pin GP27 (entrada analógica) para que a través de un potenciómetro envíe una señal numérica al display.

La señal analógica la mapearemos convirtiendo su rango de 0 a 65530 a 0 a 1000 y lo haremos como siempre con una función que definimos al empezar.

El valor que mostramos en el display también lo imprimiremos en la consola de **Thonny**

Programa

Nuestro programa se compondrá de los siguientes pasos a realizar:

1. Carga las librerías

```
from machine import Pin, ADC
from time import sleep_us, sleep_ms
import tm1637
```

2. Definir los elementos display y canal analógico en sus correspondientes pines

```
display = tm1637.TM1637(clk=Pin(5), dio=Pin(4))
```

```
adc = ADC(27)
```

3. Definimos la función de mapeado a la que ponemos el nombre de "**convert**"

```
def convert(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) // (in_max - in_min) +
out_min
```

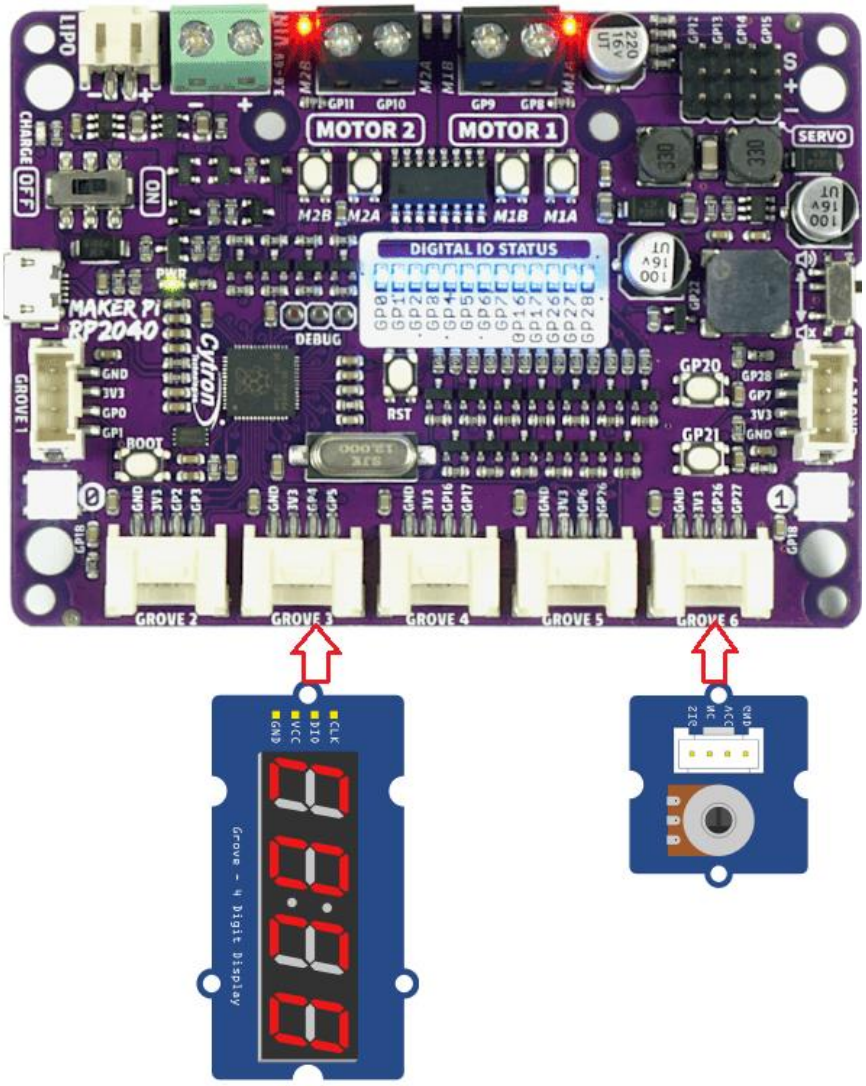
4. Montamos la parte del bucle incluida en una estructura de control de error "**try… except**".

```
        z=convert(adc.read_u16(), 0, 65535, 0, 1000)
        display.number(z)
        print("ADC Valor:", z, "V")
        sleep_ms(100)
```

El programa completo seria el siguiente

```
#Mostrar el valor de un canal de entrada analógica en un display de 4 dígitos
from machine import Pin, ADC
from time import sleep_us, sleep_ms
import tm1637 #Libreria del dispositivo tm1637

#definimos la conexión del objeto tm
display = tm1637.TM1637(clk=Pin(5), dio=Pin(4))
adc = ADC(27)  #Leer pin GP27 analógico (0 a 65535)
# Mapea devolviendo un entero
def convert(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) // (in_max - in_min) + out_min

try:
    while True:
        #mapeamos el valor del canal GP28
        z=convert(adc.read_u16(), 0, 65535, 0, 1000)
        #Mostramos el valor en el display de 4 dígitos
        display.number(z)
        print("ADC Valor:", z, "V")
        sleep_ms(100)
except:
    pass
```

Montaje de la practica

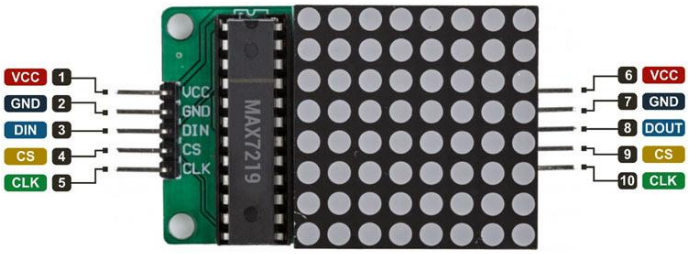


17.5. Control de matriz de LEDs Max7219

Objetivo

Vamos a realizar varios ejemplos de control de la matriz de LEDs de 8x8 con los que probaremos sus posibilidades

Funcionalidad



Nuestro dispositivo es una matriz de LEDs que podemos controlar direccionando cada uno de los LEDs que la componen de manera individual. Este elemento es muy usado para montar anuncios luminosos o indicadores.

Para su gobierno usamos además de los pines Vcc y GND tres pines mas: DIN (entrada de datos), CS (habilitación) y CLK (reloj).

Podríamos encadenar varios de estos dispositivos uniendo los pines de salida con los de entrada del anterior.

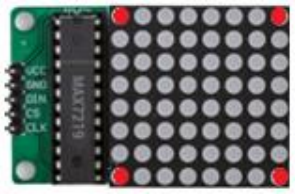
Programas

Vamos a realizar varios programas de ejemplo en esta practica.

Ejemplo1

La programación tiene una parte común que será la de configuración del dispositivo en la que designaremos los pines y funcionalidad del la matriz 8x8

Se trata de encender y apagar los LEDs (0,0), (0,7),(7,) y (7,7) a modo de Blink



1. **Importación de librerías**

   Importaremos la biblioteca para la clase **Pin** y la clase **SPI** desde el módulo **machine**. Para interactuar con los GPIO de entrada/salida importaremos el módulo de máquina que contiene clases para interactuar con los GPIOs. También debemos importar el módulo de tiempo para insertar el **sleep** retraso en nuestro script de **MicroPython**. Además, también importaremos la librería **max7219** para trabajar con la pantalla LED.

```
import max7219
from machine import Pin, SPI
from time import sleep
```

2. **Inicialización de SPI**

   El siguiente paso es inicializar la interfaz SPI con los parámetros requeridos, incluido el número de canal SPI y los pines SPI, creando un objeto **'spi'**. Mediante el uso de SPI() especificamos el número de canal SPI como primer parámetro y los pines SPI **sck** y **mosi** como el resto de los parámetros.

```
spi = SPI(0,sck=Pin(2),mosi=Pin(3))
```

   A continuación configuraremos el pin GPIO conectado con el pin CS del módulo como pin de salida. El primer argumento de la clase Pin() es el número pin en el que estamos configurando la salida. La salida está en GP5 que está conectado a CS. El segundo argumento muestra el modo pin, por ejemplo, entrada digital o modo de salida digital. Como estamos configurando GP5 como salida digital, lo dimos como 'Pin.Out'. Esto se almacena en el objeto 'cs'.

```
cs = Pin(5, Pin.OUT)
```

   Luego crearemos el objeto de matriz 'display' y especificaremos 4 dispositivos.

```
display = max7219.Matrix8x8(spi, cs, 1)
```

3. Estableceremos el brillo de la pantalla utilizando el método brightness() en el objeto de visualización. Esto toma un parámetro de 1 a 15 donde 15 es el nivel más alto de brillo.

```
display.brightness(10)
```

4. Borramos la pantalla usando el método fiill

```
display.fill(False)
```

5. Blink cuatro leds. Se trata de activar y desactivar los LEDs: (0,0), (0,7),(7,) y (7,7)

```
while True:
    #Blink en el LED 0,7
    display.pixel(0,0, True) #Enciende el LED 7,7
    display.pixel(0, 7, True)
    display.pixel(7, 0, True)
    display.pixel(7, 7, True)
    display.show()
    sleep(0.5)
    display.pixel(0,0, False) #Enciende el LED 7,7
    display.pixel(0,7, False)
    display.pixel(7,0, False)
    display.pixel(7,7, False)
    display.show()
    sleep(0.5)
```

El programa completo es

```
#Activacion Blink de cuatro leds de las esquinas
import max7219
from machine import Pin, SPI
from time import sleep

spi = SPI(0,sck=Pin(2),mosi=Pin(3))
cs = Pin(5, Pin.OUT)

display = max7219.Matrix8x8(spi, cs, 1)

display.brightness(10)

display.fill(False)

while True:
    #Blink en el LED 0,7
    display.pixel(0,0, True) #Enciende el LED 7,7
    display.pixel(0, 7, True)
    display.pixel(7, 0, True)
    display.pixel(7, 7, True)
    display.show()
    sleep(0.5)
    display.pixel(0,0, False) #Enciende el LED 7,7
    display.pixel(0,7, False)
```

```
    display.pixel(7,0, False)
    display.pixel(7,7, False)
    display.show()
    sleep(0.5)
```

Ejemplo 2

Se trata de mostrar los caracteres siguientes que los recogeremos de las cadenas

```
display.text('PICO',0,0,1)      Muestra “P”
display.text('1234',0,0,1)      Muestra el “1”
display.text('done',0,0,1)      Muestra la “d”
```

Dependiendo del parámetro “1” de la siguiente instrucción así mostraremos un carácter u otro en cada string

```
display = max7219.Matrix8x8(spi, cs, 1)
```

Así se mostrara el carácter correspondiente

Programa completo

Las partes comunes del programa no volveremos a repetirlas

```
#Muestra varias entidades de texto
from machine import Pin, SPI
import max7219
from time import sleep

spi = SPI(0,sck=Pin(2),mosi=Pin(3))
cs = Pin(5, Pin.OUT)

display = max7219.Matrix8x8(spi, cs, 1)

display.brightness(10)

while True:

    display.fill(0)
    display.text('PICO',0,0,1)
    display.show() #Muestra la letra P
```

```
sleep(1)

display.fill(0)
display.text('1234',0,0,1)#Muestra el numero 1
display.show()
sleep(1)

display.fill(0)
display.text('done',0,0,1) #Muestra la d
display.show()
sleep(1)
```

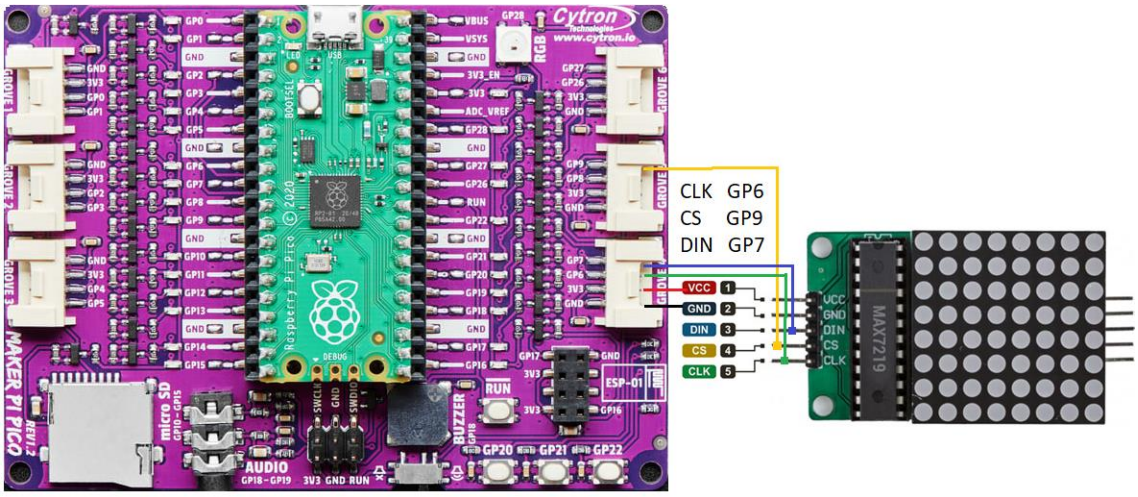
Ejemplo 3

Con este ejemplo terminamos de usar funciones para dibujar puntos, líneas, cuadrados, etc.

```
#Distintos gráficos con MAX7219
from machine import Pin, SPI
import max7219
from time import sleep

spi = SPI(0,sck=Pin(2),mosi=Pin(3))
cs = Pin(5, Pin.OUT)

display = max7219.Matrix8x8(spi, cs, 1)

display.brightness(10)

while True:
#Muestra un LED
    display.fill(False)
    display.pixel(1,2,True)
    display.show()
    sleep(2)
#Muestra una línea horizontal
    display.fill(False)
    display.hline(0,0,6,True)
    display.show()
    sleep(2)
#Muestra una línea vertical
    display.fill(False)
    display.vline(0,0,6,True)
    display.show()
    sleep(2)
#Muestra línea
    display.fill(False)
    display.line(0,0,7,7,True)
```

```
    display.show()
    sleep(2)
#Muestra una rectángulo
    display.fill(False)
    display.rect(0,0,6,6,True)
    display.show()
    sleep(2)
#Muestra un Rectángulo relleno
    display.fill(False)
    display.fill_rect(0,0,4,6,True)
    display.show()
    sleep(2)
#Muestra una Letra
    display.fill(False)
    display.text("A",0,0,True)
    display.show()
    sleep(2)
```

Montaje de la practica



17.6. Escaneo de puertos I2C

Objetivo

Con esta práctica queremos aprender cómo usar los puertos I2C de Raspberry Pi Pico.

Funcionalidad: Funcionamiento de un puerto I2C

¿Qué es I2C? Una breve descripción general

I2C es un protocolo de comunicación en serie inventado por Phillips. I2C se conoce comúnmente como I2C o IIC. Este sistema de comunicación utiliza dos líneas bidireccionales, a saber, SDA (datos seriales) y SCL (reloj serie).

El bus I2C identifica periféricos (nodos) con direcciones de 7 bits. Cada nodo puede actuar como maestro o esclavo mientras se comunica. El dispositivo maestro generalmente genera la señal de reloj en la línea SCL. Varios dispositivos esclavos pueden comunicarse con un único dispositivo maestro.

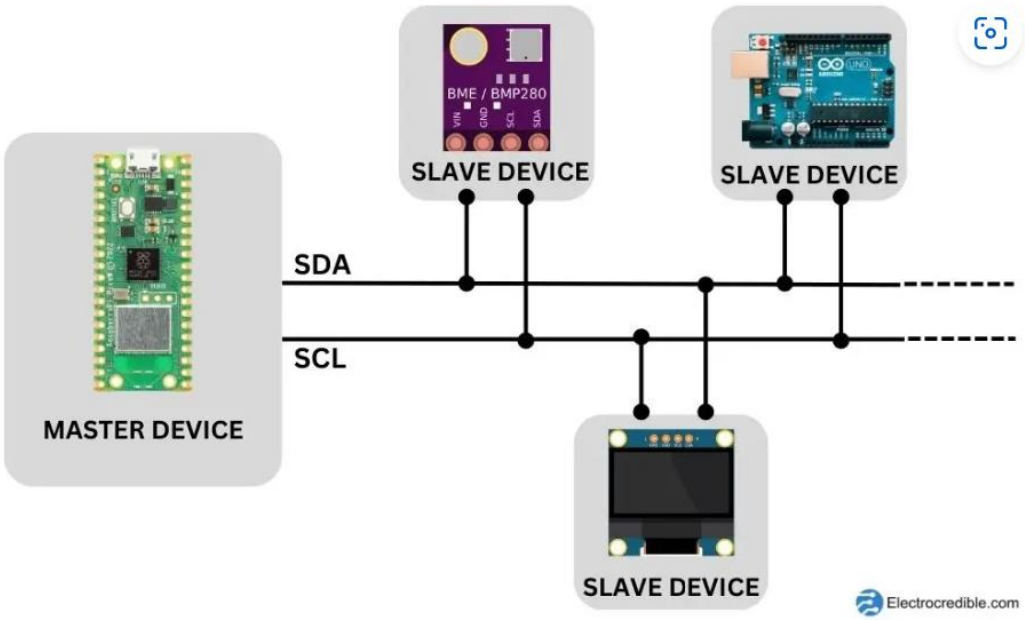


Imagen: Diagrama ilustrativo de comunicación I2C

Raspberry Pi Pico I2C Pinout y especificaciones

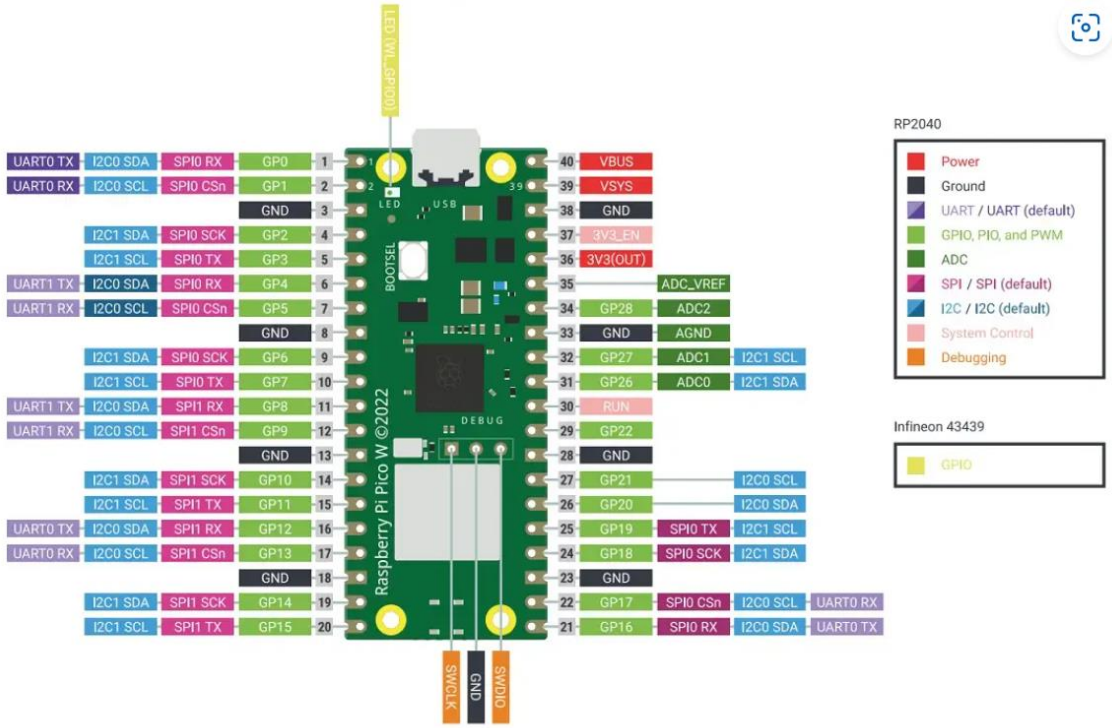
Raspberry Pi Pico se basa en el microcontrolador dual ARM Cortex-M32+ RP0 de 2040 bits. Tiene dos controladores I2C de hardware dedicados (I2C0 e I2C1) que son idénticos y se pueden controlar de forma independiente. El controlador I2C puede actuar como maestro y esclavo, siendo el modo maestro el predeterminado. Sin embargo, no puede

comportarse como amo y esclavo al mismo tiempo. El I2C en Pi Pico puede funcionar en tres modos de transferencia de datos:

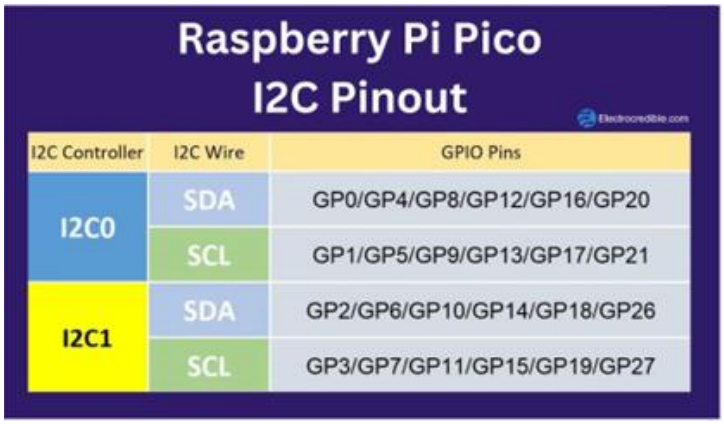
- Modo estándar (con velocidades de datos de 0 a 100 kbps),• modo rápido (con velocidades de datos inferiores o iguales a 400 kbps)
- Modo rápido plus (con velocidades de datos inferiores o iguales a 1000 kbps).

Los dispositivos en modo rápido son compatibles con versiones anteriores. Por ejemplo, con el sistema de bus I0C de 100 a 2 kbps, los dispositivos de modo rápido pueden conectarse con dispositivos de modo estándar. Sin embargo, debido a que no pueden mantenerse al día con la velocidad de transmisión más rápida y experimentarían estados impredecibles, los dispositivos de modo estándar no son compatibles con versiones superiores y no deben incluirse en un sistema de bus I2C de modo rápido.

Vemos en la siguiente imagen el pinout de Raspberry Pi Pico. Los pines marcados como SDA y SCL se pueden utilizar para la comunicación I2C.



Fuente: Hoja de datos Imagen: Rpi Pico I2C Pinout



Fuente de la imagen (*Electrocredible.com*).

Además de los controladores I2C de hardware dedicados, también podemos implementar I2C en software. Además, el PIO (Programmable Input Output) en Raspberry Pi Pico se puede configurar para comportarse como una interfaz I2C.

Funciones útiles de MicroPython para la comunicación I2C en RP2040

Constructores de la clase I2C: un protocolo serie de dos hilos

***.class*máquina. I2C(*id, *, scl, sda, freq=400000*)**

Construya y devuelva un nuevo objeto I2C utilizando los siguientes parámetros:

- *id* identifica un periférico I2C particular. Valores permitidos para dependen del puerto/placa en particular
- *scl* debe ser un objeto de PIN que especifique el pin que se va a utilizar para SCL.
- *sda* debe ser un objeto pin que especifique el pin que se usará para SDA.
- *Freq* debe ser un número entero que establezca la frecuencia máxima para SCL.

Tenga en cuenta que algunos puertos/placas tendrán valores predeterminados de *scl* y *sda* que se pueden cambiar en este constructor. Otros tendrán valores fijos de *SCL* y *SDA* que no se pueden cambiar.

***.class*máquina. SoftI2C(*scl, sda, *, freq=400000, timeout=50000*)**

Construir un nuevo objeto I2C de software. Los parámetros son:

- *scl* debe ser un objeto de PIN que especifique el pin que se va a utilizar para SCL.

- *sda* debe ser un objeto pin que especifique el pin que se usará para SDA.
- *Freq* debe ser un número entero que establezca la frecuencia máxima para SCL.
- *El tiempo de espera* es el tiempo máximo en microsegundos para esperar el reloj estiramiento (SCL mantenido bajo por otro dispositivo en el bus), después de que se plantea una excepción. `OSError(ETIMEDOUT)`

Métodos generales

I2C. init(*scl, sda, *, freq=400000*)

Inicialice el bus I2C con los argumentos dados:

- *scl* es un objeto pin para la línea SCL
- *sda* es un objeto pin para la línea SDA
- *freq* es la frecuencia de reloj SCL

En el caso del hardware I2C, la frecuencia de reloj real puede ser inferior a la frecuencia solicitada. Esto depende del hardware de la plataforma. El La velocidad se puede determinar imprimiendo el objeto I2C.

I2C. Deinit()

Apague el bus I2C.

Disponibilidad: WiPy.

I2C. escanear()

Escanee todas las direcciones I2C entre 0x08 y 0x77 inclusive y devuelva una lista de los que responden. Un dispositivo responde si tira de la línea SDA baja después de Su dirección (incluyendo un bit de escritura) se envía en el bus.

Operaciones I2C primitivas

Los métodos siguientes implementan las operaciones primitivas del bus del controlador I2C y pueden combinarse para realizar cualquier transacción I2C. Se proporcionan si necesita más control sobre el bus, de lo contrario se pueden utilizar los métodos estándar (ver más abajo).

Estos métodos solo están disponibles en la **máquina. Clase SoftI2C**.

I2C. empezar()

Genere una condición START en el bus (SDA pasa a baja mientras que SCL es alta).

I2C. parar()

Genere una condición STOP en el bus (SDA pasa a alta mientras SCL es alta).

I2C. readinto(*buf, nack=True, /*)

Lee bytes del bus y los almacena en *buf*. El número de bytes Read es la longitud de *buf*. Se enviará un ACK en el autobús después de recibiendo todos menos el último byte. Después de recibir el último byte, si *nack* es verdadero, se enviará un NACK, de lo contrario se enviará un ACK (y en este en caso de que el periférico asuma que se van a leer más bytes en una llamada posterior).

I2C. write(*buf*)

Escriba los bytes de *buf* en el bus. Comprueba que se ha recibido un ACK después de cada byte y deja de transmitir los bytes restantes si un NACK es recibido. La función devuelve el número de ACK recibidos.

Operaciones de bus estándar

Los métodos siguientes implementan el controlador estándar I2C de lectura y escritura operaciones dirigidas a un dispositivo periférico determinado.

I2C. readfrom(*addr, nbytes, stop=True, /*)

Leer *nbytes* desde el periférico especificado por *addr*. Si *stop* es true, se genera una condición STOP al final de la transferencia. Devuelve un objeto **bytes** con los datos leídos.

I2C. readfrom_into(*addr, buf, stop=True, /*)

Leer en *buf* desde el periférico especificado por *addr*. El número de bytes leídos será la longitud de *buf*. Si *stop* es true, se genera una condición STOP al final de la transferencia.

El método devuelve . None

I2C. writeto(*addr, buf, stop=True, /*)

Escriba los bytes de *buf* en el periférico especificado por *addr*. Si un NACK se recibe después de la escritura de un byte de *buf* y luego el Los bytes restantes no se envían. Si *stop* es true, entonces una condición STOP es generado al final de la transferencia, incluso si se recibe un NACK. La función devuelve el número de ACK recibidos.

I2C. writevto(*addr, vector, stop=True, /*)

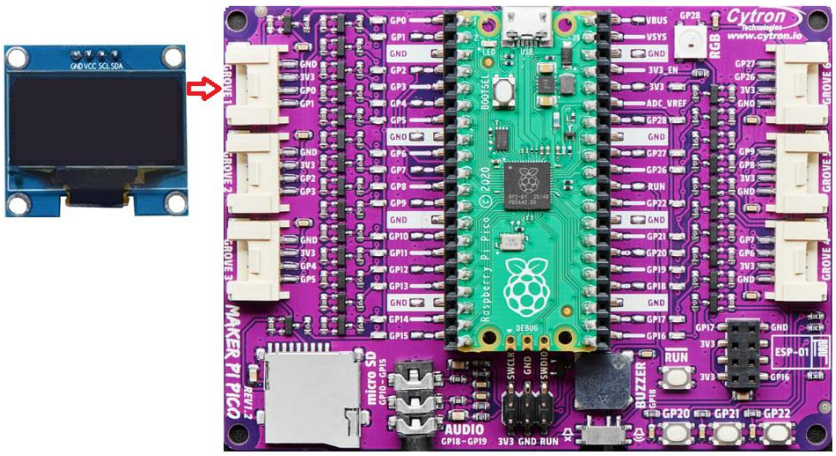
Escriba los bytes contenidos en *vector* en el periférico especificado por *addr*. *vector* debe ser una tupla o lista de objetos con el protocolo buffer. El *sumador* se envía una vez y luego los bytes de cada objeto en *vector* se escriben secuencialmente. Los objetos en *vector* pueden ser cero bytes en longitud, en cuyo caso no contribuyen a la salida.

Si se recibe un NACK después de la escritura de un byte de uno de los objetos en *vector* luego los bytes restantes, y cualquier objeto restante, no se envían. Si *stop* es true, se genera una condición STOP en el final de la transferencia, incluso si se recibe un NACK. La función devuelve el número de ACK recibidos.

Realización de nuestro ejemplo

Se trata de escanear dispositivos conectados I2C

Montaremos el siguiente esquema.

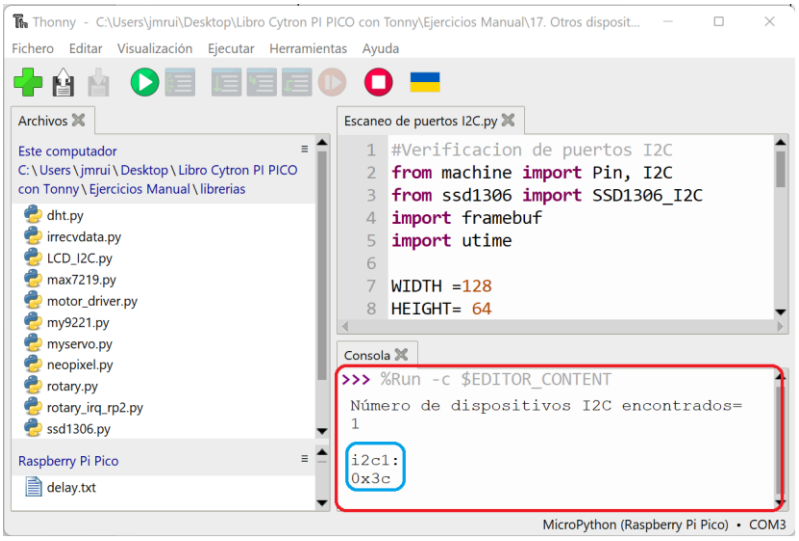


Después de que se hayan realizado las conexiones adecuadas, podemos cargar un script para escanear todos los dispositivos conectados al bus I2C. Copie el código siguiente y péguelo en un nuevo proyecto en **Thonny** IDE.

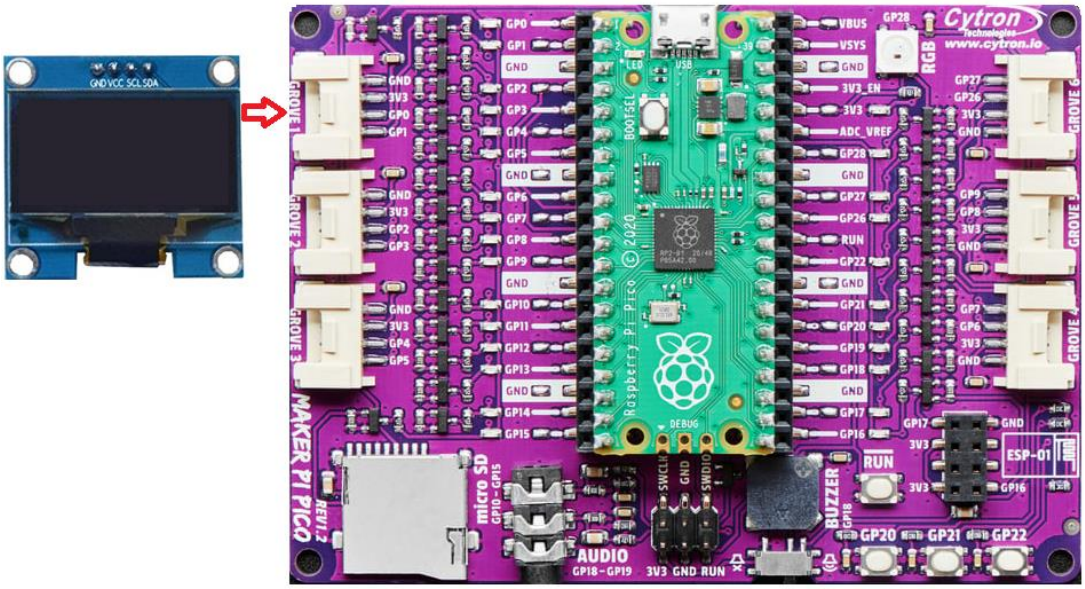
Programa

```
#Verificacion de puertos I2C
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
WIDTH =128
HEIGHT= 64
#Los puertos posibles pueden ser 2
# i2c=I2C(0,scl=Pin(1),sda=Pin(0),freq=200000)
# i2c=I2C(1,scl=Pin(3),sda=Pin(2),freq=200000)
#Vamos a usar el primero de ellos
i2c=I2C(0,scl=Pin(1),sda=Pin(0),freq=200000)
oled = SSD1306_I2C(WIDTH,HEIGHT,i2c)
#Escaneo de puertos I2C e impresión de sus direcciones
devices = i2c.scan()
if len(devices) != 0:
    print('Número de dispositivos I2C encontrados=',len(devices))
    for device in devices:
        printhex(device))
else:
       print("No encuentro dispositivos ")
#Escritura en el dispositivo OLED
while True:
    oled.fill(0)
    oled.text("Probando I2C", 0, 0)
    oled.text("con OLED ", 0, 20)
    oled.show()
```

Al ejecutarse el programa la ventana del Shell mostrará la dirección del dispositivo que luego usaremos en nuestro código. A continuación se muestra una imagen de la salida que se muestra en **Thonny** IDE, con 0x3C devuelto como la dirección I2C de la pantalla OLED.



Montaje de la practica



Análisis y propuesta de actividades

Al ejecutar el programa nos aparecerá en la Consola de **Thonny** el mensaje imprimido del puerto detectado.

Podemos probar a realizar la conexión en el I2C1 y ver el resultado. No olvidemos que se pueden conectar en los pines que se indican en la tabla:

Raspberry Pi Pico I2C Pinout

I2C Controller	I2C Wire	GPIO Pins
I2C0	SDA	GP0/GP4/GP8/GP12/GP16/GP20
	SCL	GP1/GP5/GP9/GP13/GP17/GP21
I2C1	SDA	GP2/GP6/GP10/GP14/GP18/GP26
	SCL	GP3/GP7/GP11/GP15/GP19/GP27

LINKs INTERESANTES

- **Getting started with Raspberry Pi Pico**
- **Maker Pi de Cytron**
- **Maker Pi RP2040 de Cytron**
- **Thonny, Python IDE for beginners**
- **Overview — MicroPython latest documentation**
- **SDK de Python de Raspberry Pi Pico.**
- **Hoja de especificaciones del microcontrolador**
- **Portal 330 ohms**
- **Electrocredible - Learn Electronics, Make Circuits.**
- **Documentación de la firma Raspberry**
- **MicroPython for Kids (coderdojotc.org)**
- **Projects | Computer coding for kids and teens | Raspberry Pi**
- **Pico Archives (peppe8o.com)**
- **Raspberry Pi Book PDF Download from HackSpace Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)**

Profesor: José Manuel Ruiz Gutiérrez

Marzo 2023