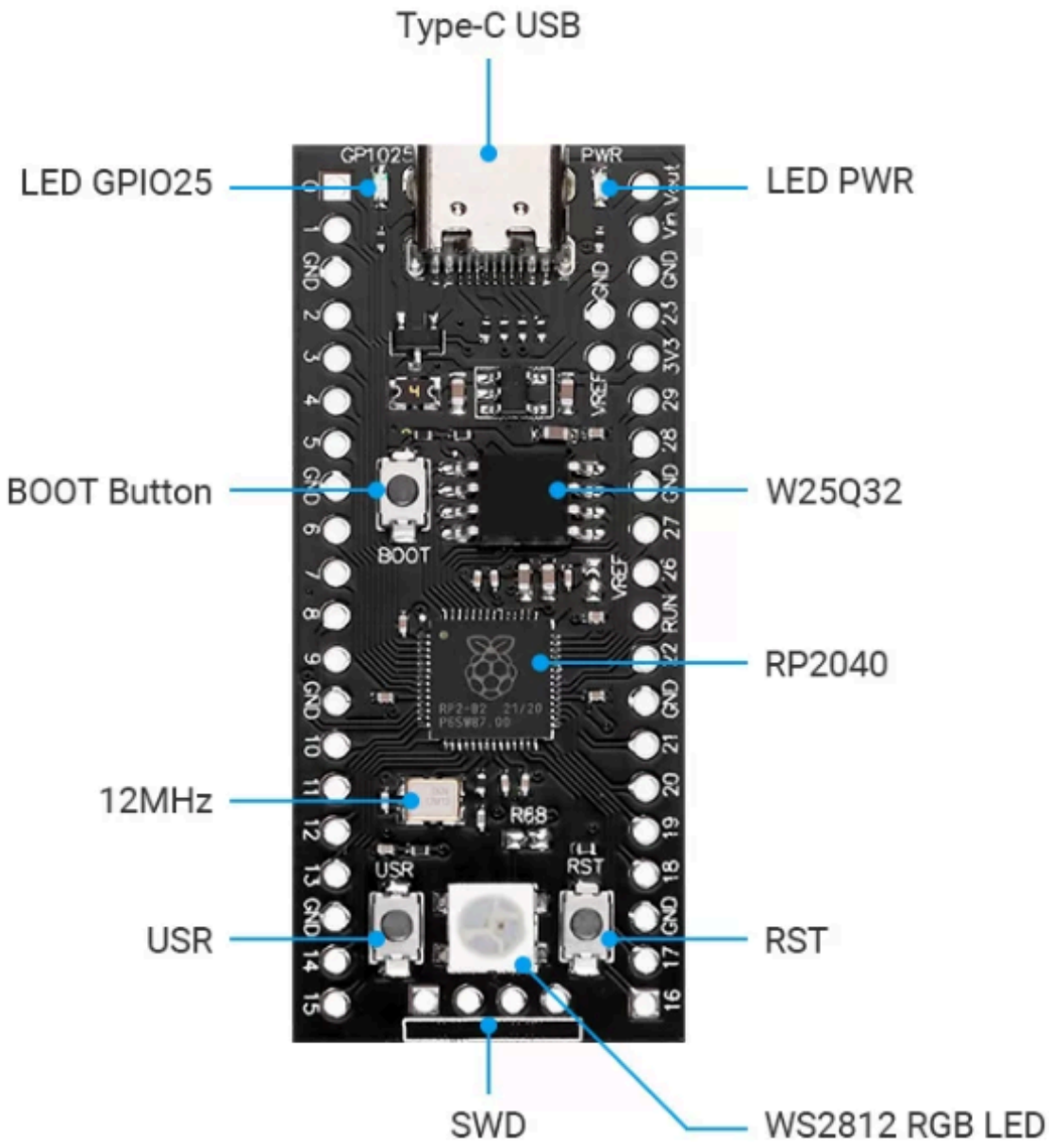


UF2

For Correct Flash Size we use WeAct Studio RP2040 uf2 files, the default is 16MB, the rest are for 8MB, 4MB,2MB.

externa QSPI y configuraciones de E/S altamente flexibles, incluyendo I2C, SPI, PIO (Programming I /O). Esta producción de placa central RP2040 utiliza microprocesamiento (MCU) RP2040 para hacer la placa de circuito de la placa central, que es conveniente para que los usuarios amplíen periféricos, diseñen y construyan circuitos. El RP2040 es compatible con la placa Raspberry Pi Pico en la mayoría de los casos y mejorar las deficiencias de la placa Pico.

The RP2040 microprocessor (MCU) integrates a dual-core ARM architecture Cortex-M0+ core, directly integrates 264kb RAM inside, supports up to 16M QSPI external expansion Flash, and highly flexible I/O settings, including I2C, SPI, PIO (can Programming I/O). This RP2040 core board production uses RP2040 micro-processing (MCU) to make the core board circuit board, which is convenient for users to expand peripherals and design and build circuits. The RP2040 is compatible with the Raspberry Pi Pico core board in most cases and improves the deficiencies of the Pico board.

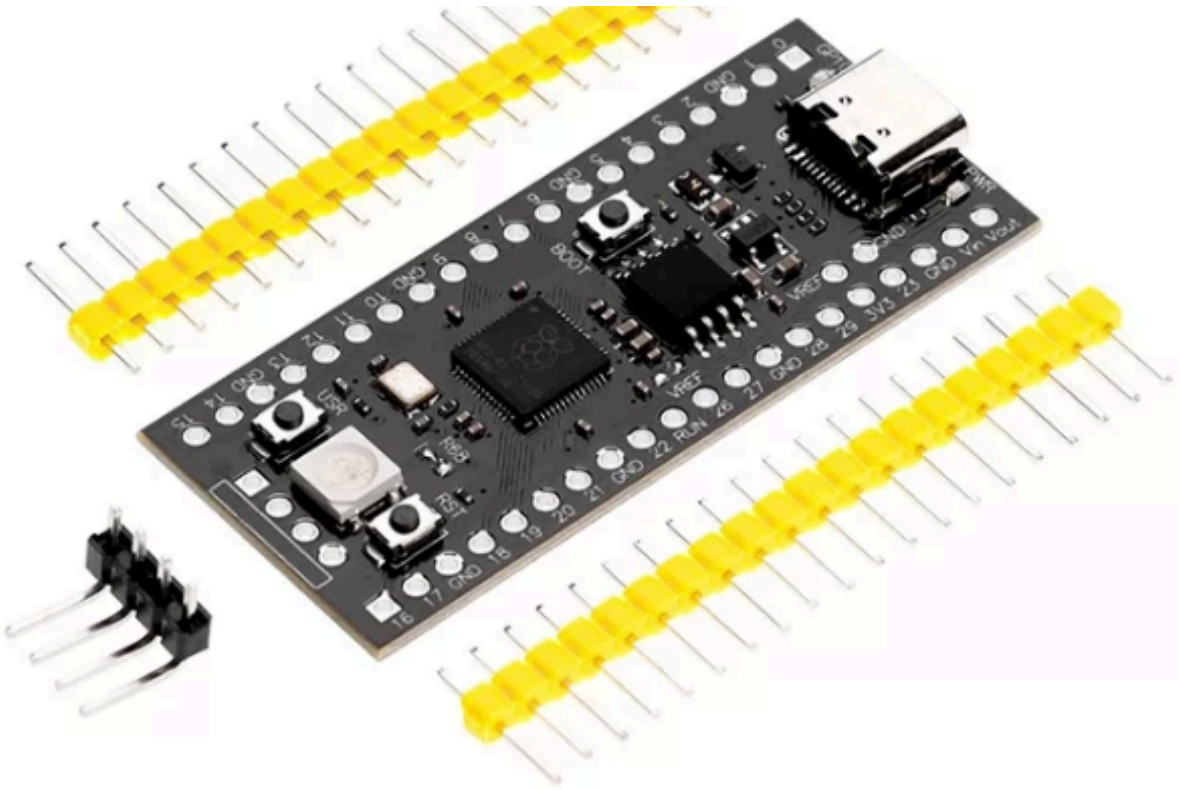
- Dual-core Cotex-M0+
- Low power consumption
- High performance
- WS2812-RGB light group
- Support up to 16M QSPI external expansion Flash, rich peripheral interfaces
- Support Python fast programming
- Suppport C language programming
- Added PWR power indicator to improve the human-computer interaction experience of plugging in USB power supply
- Change MicroUSB interface to Type-C interface
- Add a reset button, which is convenient for users to reset and update the firmware in the power-on state
- Add the USRkey user button, which is convenient for user verification and a simple input function
- The RGB light group is added, which is convenient for users to operate. The RGB light group reflects various information through the on-off and brightness of the red, green, and blue lights.
- CPU Dual-core ARM Cortex M0+ processor up to 133MHz
- FLASH Memory 16MB off-chip
- SRAM 264KB
- Digital I/O Pins: 28
- Analog I/O Pins: 4
- I2C interface: 2

- 3 LEDs for power, user and programming
- User and Reset buttons
- USB Type-C connector
- Power supply and downloading interface USB Type-C
- Circuit Operating Voltage: 3.3V
- Supply Voltage via USB: 5V
- Power 3.3V/5V DC
- Dimensions 53.3 x 22.9mm

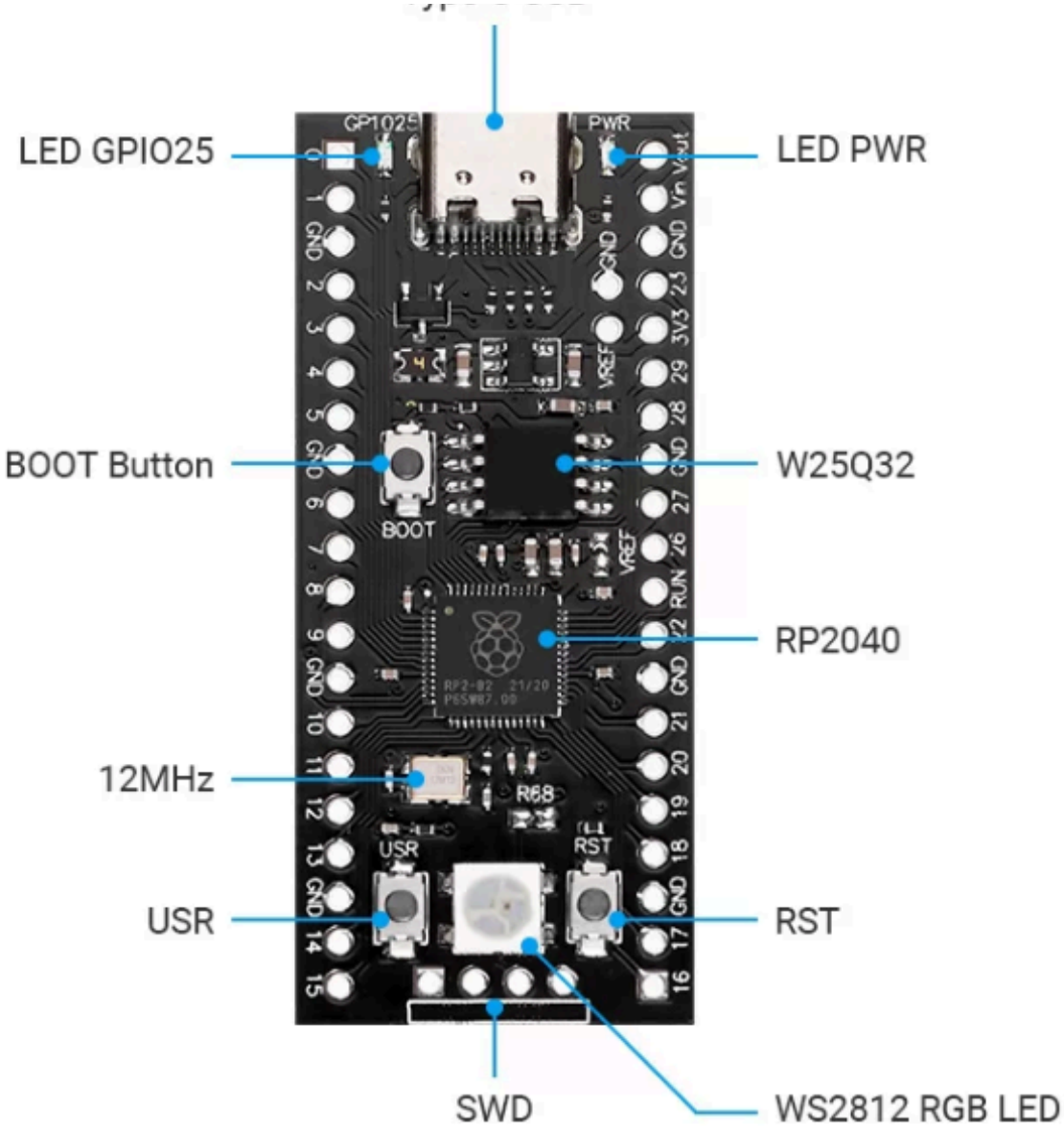
VCC-GND Studios' YD-RP2040 has the following improvements over the Raspberry Pi Pico:

- Added PWR power LED
- The USB interface was changed to type-C USB
- Added reset button to facilitate reset operation and firmware update operation.
- Added the USRkey user button `(GPIO24)`.
- Added RGB lights `(GPIO23)`
- Change PICO's W25Q16 to W25Q32（**4M**）/W25Q64（**8M**）/W25Q128（**16M**）

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 Bios

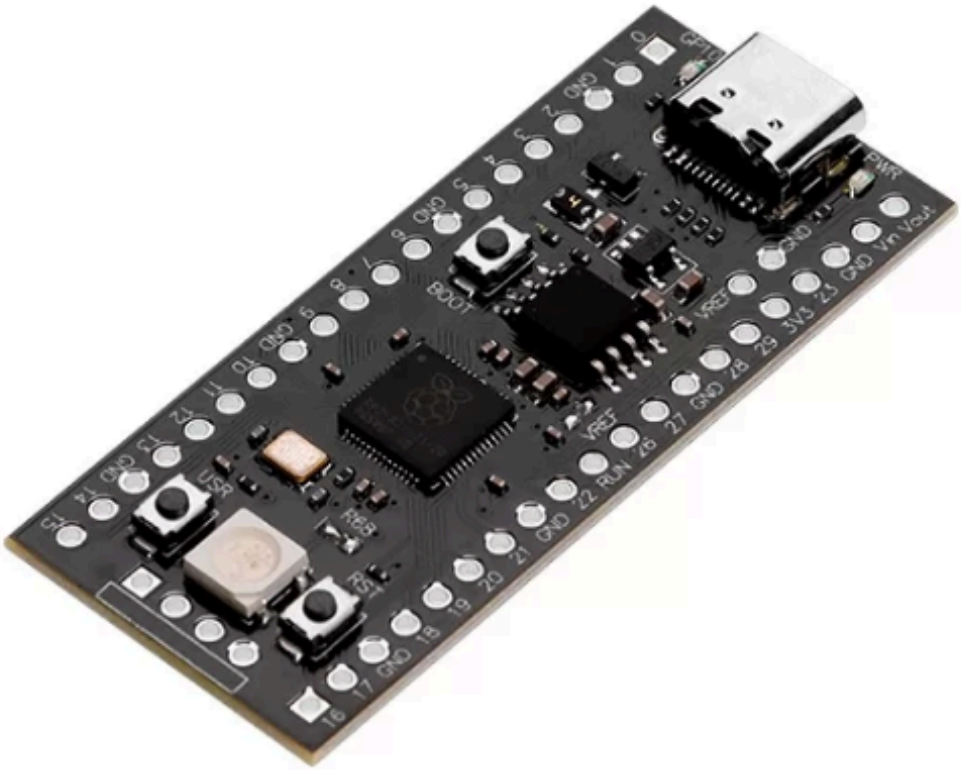


16MB W25Q128JVSG

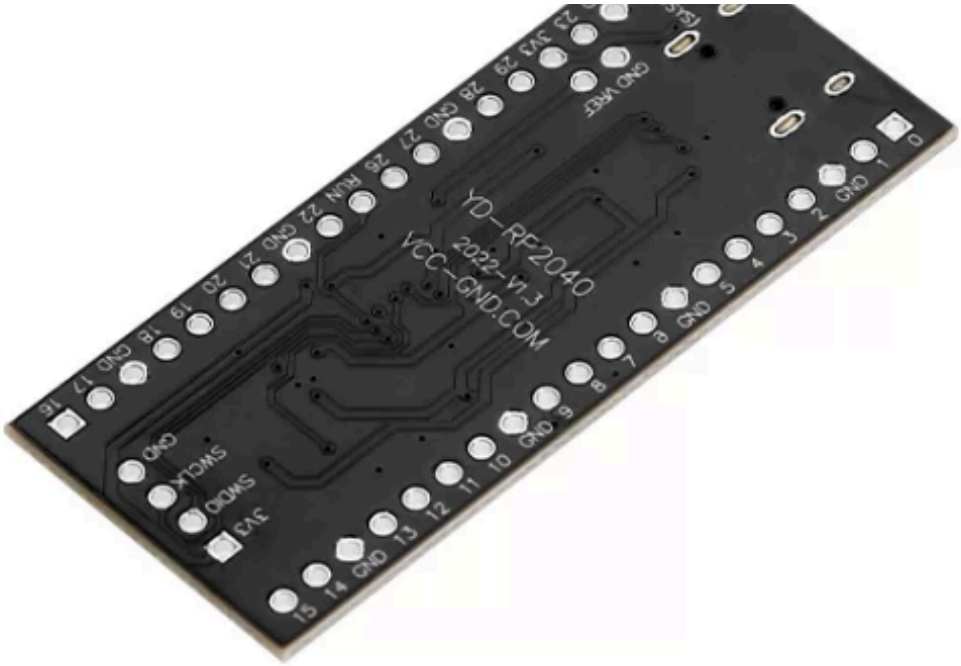

LED GPIO25
LED PWR
BOOT Button
W25Q32
RP2040
12MHz
USR
RST
SWD
WS2812 RGB LED
BOOT
R68
USR
RST

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

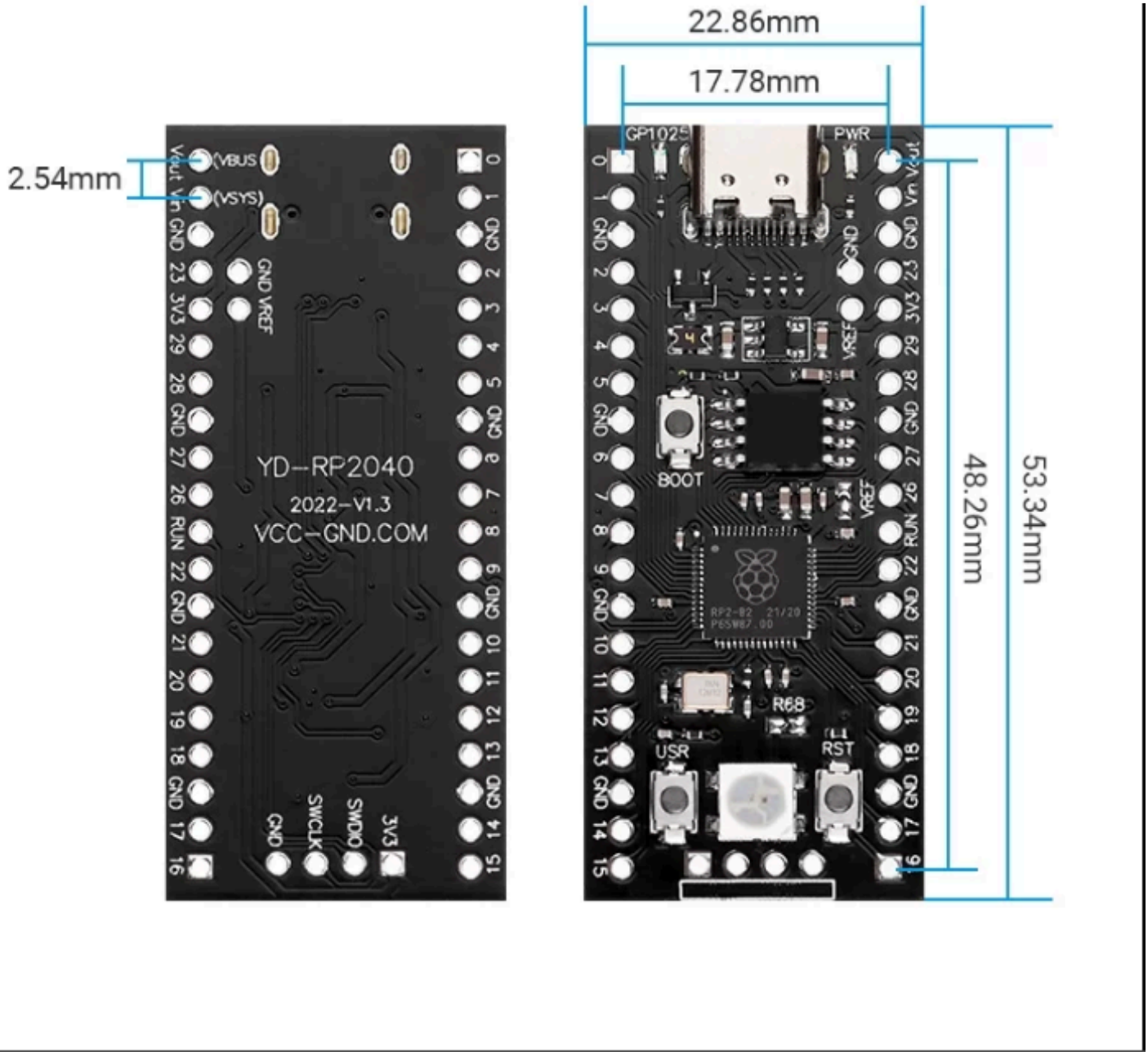
Bios



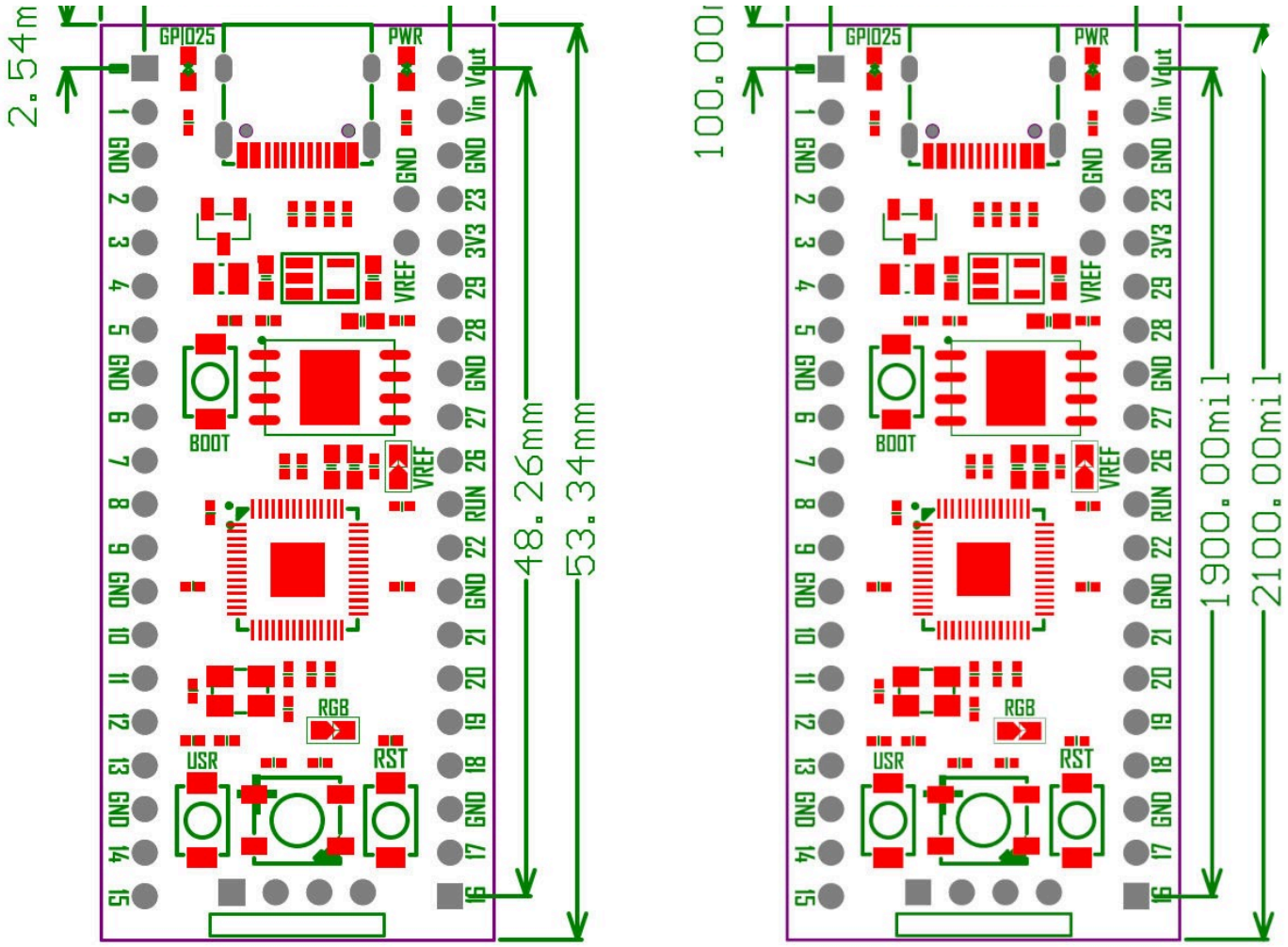
46 Pins



28 GPIO


22.86mm
17.78mm
2.54mm
48.26mm
53.34mm
YD-RP2040
2022-V1.3
VCC-GND.COM
BOOT
USR
RST
R68
PWR
GP1025

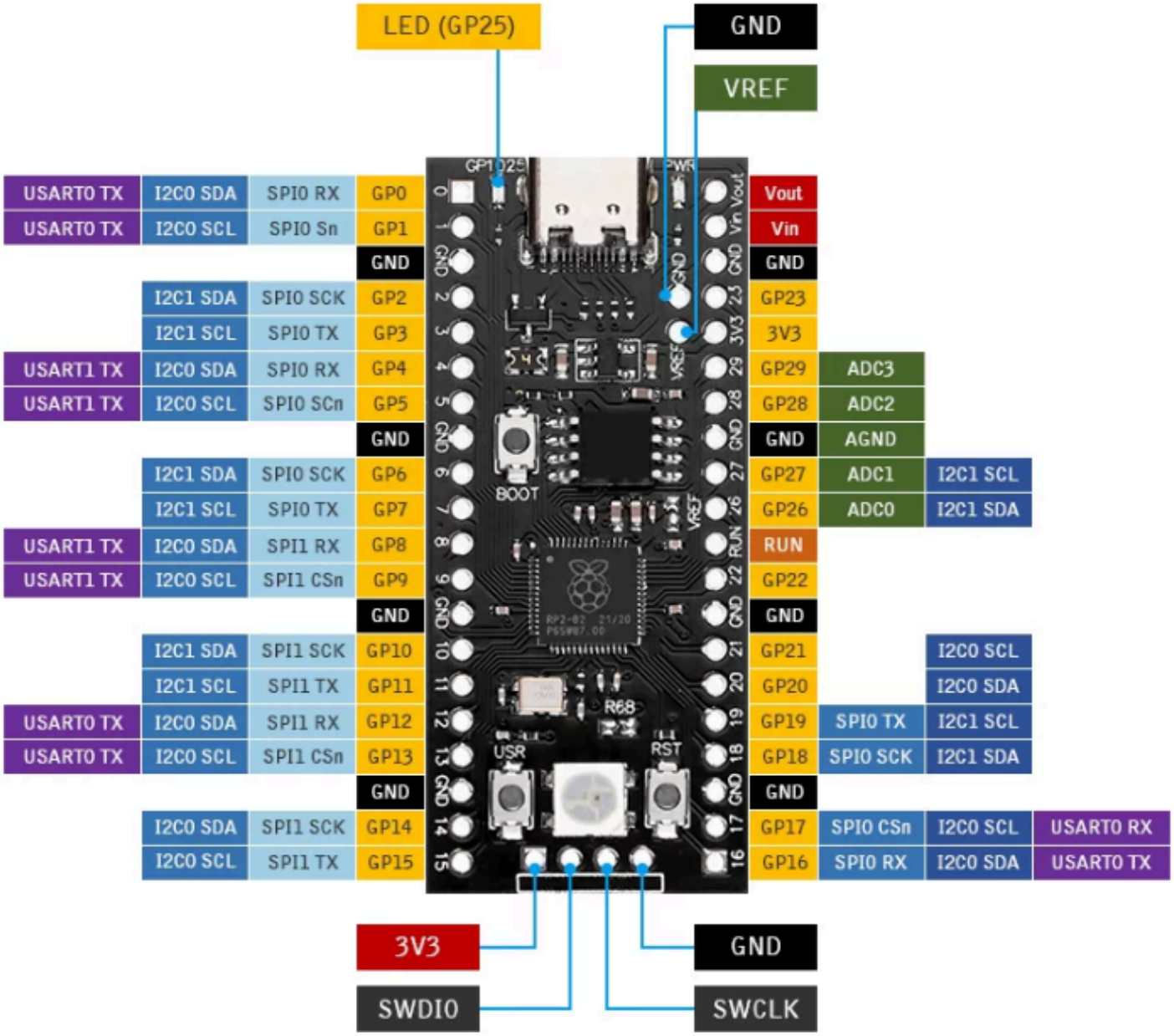
Dimensions

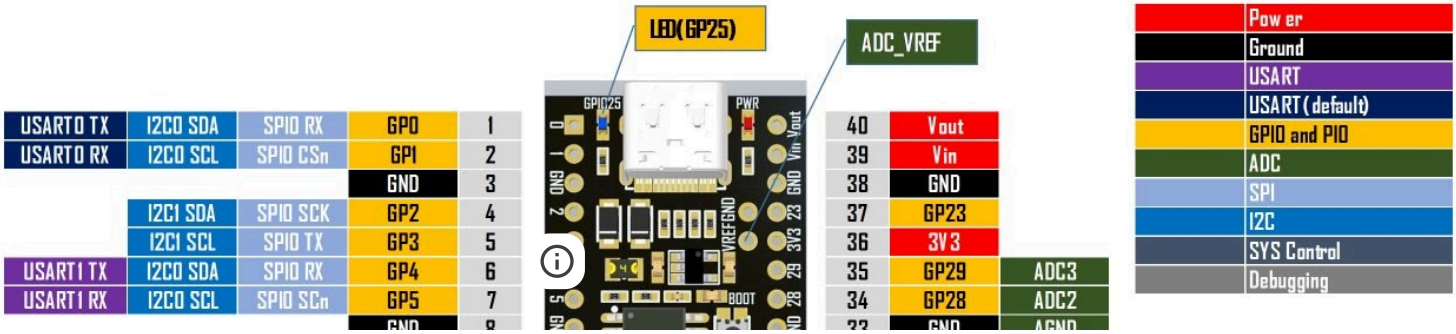

2.54m
GPIO25
PWR
GND
3V3
VREF
BOOT
RUN
RGB
USR
RST
Vin
Vout
48.26mm
53.34mm
100.00
1900.00mil
2100.00mil

Pinout:

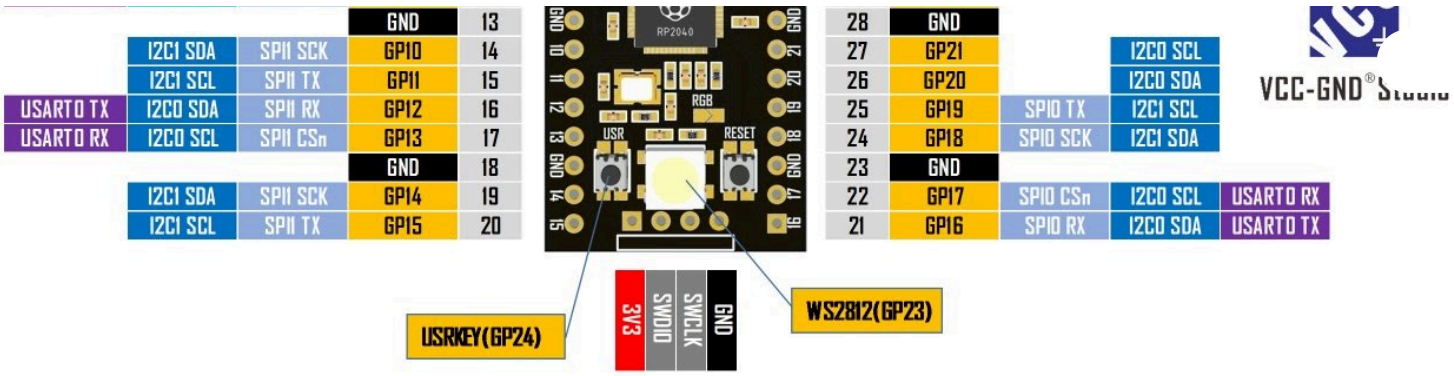
TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

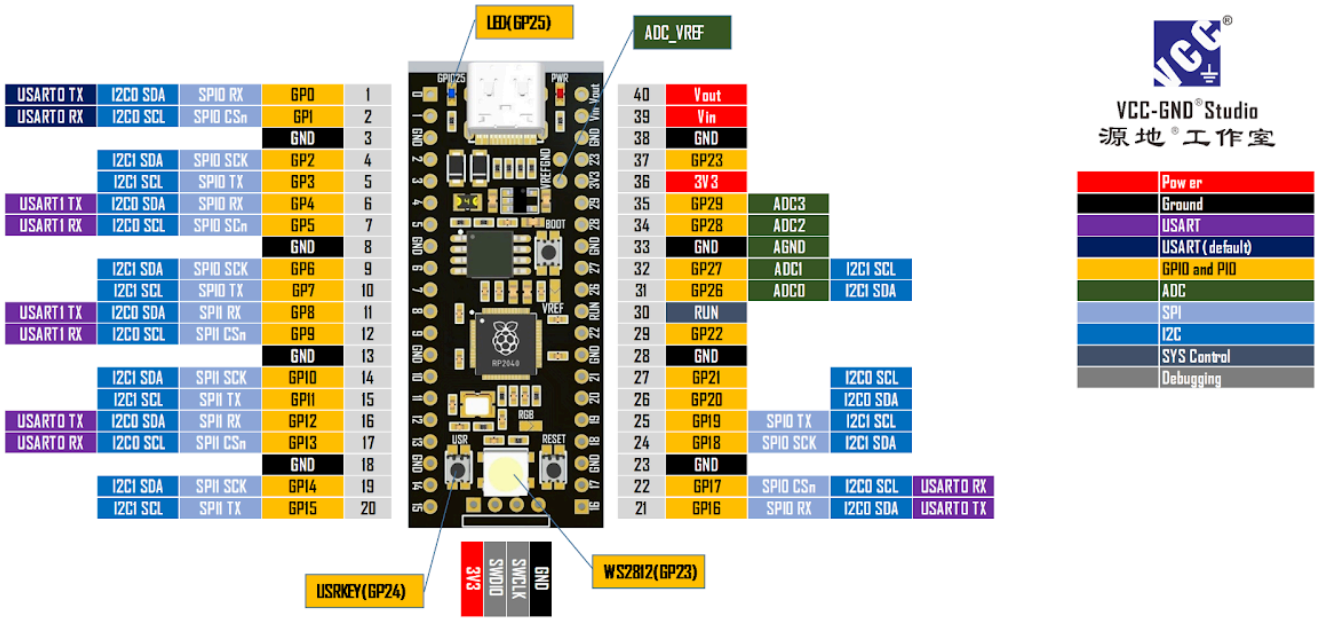
Bios


LED (GP25)
GND
VREF
USART0 TX I2C0 SDA SPI0 RX GP0 0
USART0 TX I2C0 SCL SPI0 Sn GP1 1
GND
I2C1 SDA SPI0 SCK GP2
I2C1 SCL SPI0 TX GP3
USART1 TX I2C0 SDA SPI0 RX GP4
USART1 TX I2C0 SCL SPI0 SCn GP5
GND
I2C1 SDA SPI0 SCK GP6
I2C1 SCL SPI0 TX GP7
USART1 TX I2C0 SDA SPI1 RX GP8
USART1 TX I2C0 SCL SPI1 CSn GP9
GND
I2C1 SDA SPI1 SCK GP10
I2C1 SCL SPI1 TX GP11
USART0 TX I2C0 SDA SPI1 RX GP12
USART0 TX I2C0 SCL SPI1 CSn GP13
GND
I2C0 SDA SPI1 SCK GP14
I2C0 SCL SPI1 TX GP15
Vout
Vin
GND
GP23
3V3
GP29 ADC3
GP28 ADC2
GND AGND
GP27 ADC1 I2C1 SCL
GP26 ADC0 I2C1 SDA
RUN
GP22
GND
GP21 I2C0 SCL
GP20 I2C0 SDA
GP19 SPI0 TX I2C1 SCL
GP18 SPI0 SCK I2C1 SDA
GND
GP17 SPI0 CSn I2C0 SCL USART0 RX
GP16 SPI0 RX I2C0 SDA USART0 TX
BOOT
USR
RST
3V3
GND
SWDIO
SWCLK


LED(GP25)
ADC_VREF
USART0 TX I2C0 SDA SPI0 RX GP0 1
USART0 RX I2C0 SCL SPI0 CSn GP1 2
GND 3
I2C1 SDA SPI0 SCK GP2 4
I2C1 SCL SPI0 TX GP3 5
USART1 TX I2C0 SDA SPI0 RX GP4 6
USART1 RX I2C0 SCL SPI0 SCn GP5 7
40 Vout
39 Vin
38 GND
37 GP23
36 3V3
35 GP29 ADC3
34 GP28 ADC2
Pow er
Ground
USART
USART (default)
GPIO and PIO
ADC
SPI
I2C
SYS Control
Debugging

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 Bios





3D Case:

RP2040 USB Type-C RGB Case

- http://vcc-gnd.com:8080/yd-data/YD-RP2040/YD-2040-2022-V1.1-SCH.pdf

Dimensions:

- http://vcc-gnd.com:8080/yd-data/YD-RP2040/YD-RP2040-Metric-SIZE.pdf

Docs:

- https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html
- https://datasheets.raspberrypi.com/debug/debug-connector-specification.pdf
- https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf
- https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf
- https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf
- https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf
- https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-python-sdk.pdf
- https://github.com/ndabas/pico-setup-windows
- https://99tech.com.au/m/rp2040-yd_ws2812_example.zip
- https://datasheets.raspberrypi.com/rp2040/Minimal-KiCAD.zip
- Archivo UF2 nuke file(Para Reseteo)
- Get Latest UF2 from official MicroPython web.
- WeAct Studio 16MB MicroPython UF2.
- Pimoroni PicoLiPo 16MB MicroPython UF2.
- MicroPython with ulab (numpy-simpy like modules) uf2.[Use Pimoroni Pico LiPo 16MB].
- ulab(numpy scipy like) docs.
- JavaScript runtime Kaluma web.
- JavaScript UF2 firmware
- Circuit Python Web
- Circuit Python UF2 Download Page for YD-RP2040
- Crcuit Python UF2 firmware YD-RP2040
- Circuit Python libraries
- Lua Pico UF2
- Lua Pico Github
- CPICOM is a proof-of-concept CI ⓘ -80 2.2 Emulator UF2 firmware
- CPICOM Github

- TinyGo GitHub
- Rust and GitHub RP2040 Rust Project Template
- Rust Language Book online
- uLisp
- uLisp Github
- FreePascal
- FreePascal code Examples
- Pico-infonesPlus latest version
- Pico-infonesPlus AdafruitDVISD UF2
- Pico-infonesPlus FeatherDVI UF2
- Pico-infonesPlus PimoroniDVI UF2
- Github Pico-infonesPlus
- Picones UF2
- GitHub Pico-infones
- Arduino-Pico programming Pico with the same language of Arduino!
- Pico-56 UF2
- GitHub Pico-56
- Picocomputer 6502 RP6502 UF2(inside zip
- Page Picocomputer 6502 RP6502
- GitHub Picocomputer 6502 RP6502
- GitHub and UF2 Pico-ZxSpectrum.
- GitHub pico-tflmicro TensorFlow Lite Micro.
- PiCalc: Pocket Calculator Gerber-Video-Firmware-Instructables & 3D Parts
- Delta-Pico: Rust Scientific Calculator GitHub
- WearPico UF2 SmartWatch DIY
- WearPico GitHub
- WearPico Wiki
- BreadboardOS firmware GitHub
- PiPicoFX Simple Multieffects Gui' Web
- PiPicoFX Simple Multieffects Guitar GitHub

Firmware and Bootloader from another chinese version:

- https://99tech.com.au/m/rp2040-yd_micropython_demo.zip

WS2812 Intelligent Control Led Datasheet:

- https://cdn-shop.adafruit.com/datasheets/WS2812.pdf

GitHub WeActStudio:

- https://github.com/WeActStudio/WeActStudio.RP2040CoreBoard

Raspberry Pi OS Toolchain one line setup:

- `wget -c https://raw.githubusercontent.com/raspberrypi/pico-setup/master/pico_setup.sh && chmod +x pico_setup.sh && ./pico_setup.sh`

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

Bios

Get started with MicroPython on Raspberry Pi Pico


and Ben Everard
Raspberry Pi
PRES

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 Bios

HackSpace

TECHNOLOGY IN YOUR HANDS

hsmag.cc | February 2021 | Issue #39

INTRODUCING

RASPBERRY PI

PICO

LOW-COST · HIGH-PERFORMANCE · FLEXIBLE I/O

A NEW MICROCONTROLLER BOARD FROM RASPBERRY PI

9 772515 514006 39

Feb 2021 Issue 439 £6

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 | Bios

HackSpace

TECHNOLOGY IN YOUR HANDS

hsmag.cc | March 2021 | Issue #40

Laser cut printing

Screen printing in the digital age

RASPBERRY PI PICO

IOT Security

Add encryption to MQTT

PROJECTS

Pinecil

Is this your next soldering iron?

Things to make with your $4 microcontroller

BEES

How citizen scientists are looking after these lovely little honey-makers

ALUMINIUM FOIL CNC CROCHET LEDS

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

Bios

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

Bios

HackSpace

TECHNOLOGY IN YOUR HANDS

hsmag.cc | July 2021 | Issue #44

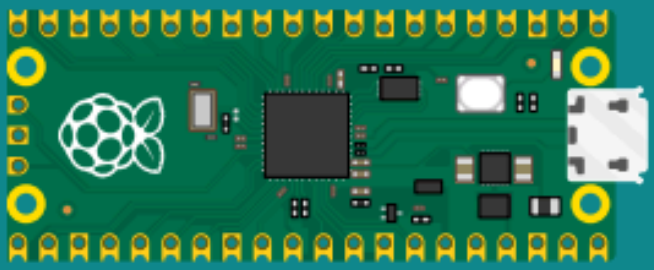
PALLETS

Creative uses for budget wood

PICO AUDIO

Make music on a microcontroller

ARDUINO + PICO



Use your Arduino skills on Raspberry Pi Pico to power EVERYTHING

Simone GIERTZ

All hail the queen of robots

3D PRINTED WIND FARM

Generate your own electricity

July 2021
Issue #44 £8
9 772515 514006 44

CNC MITRE SAWS BLUETOOTH LASERS

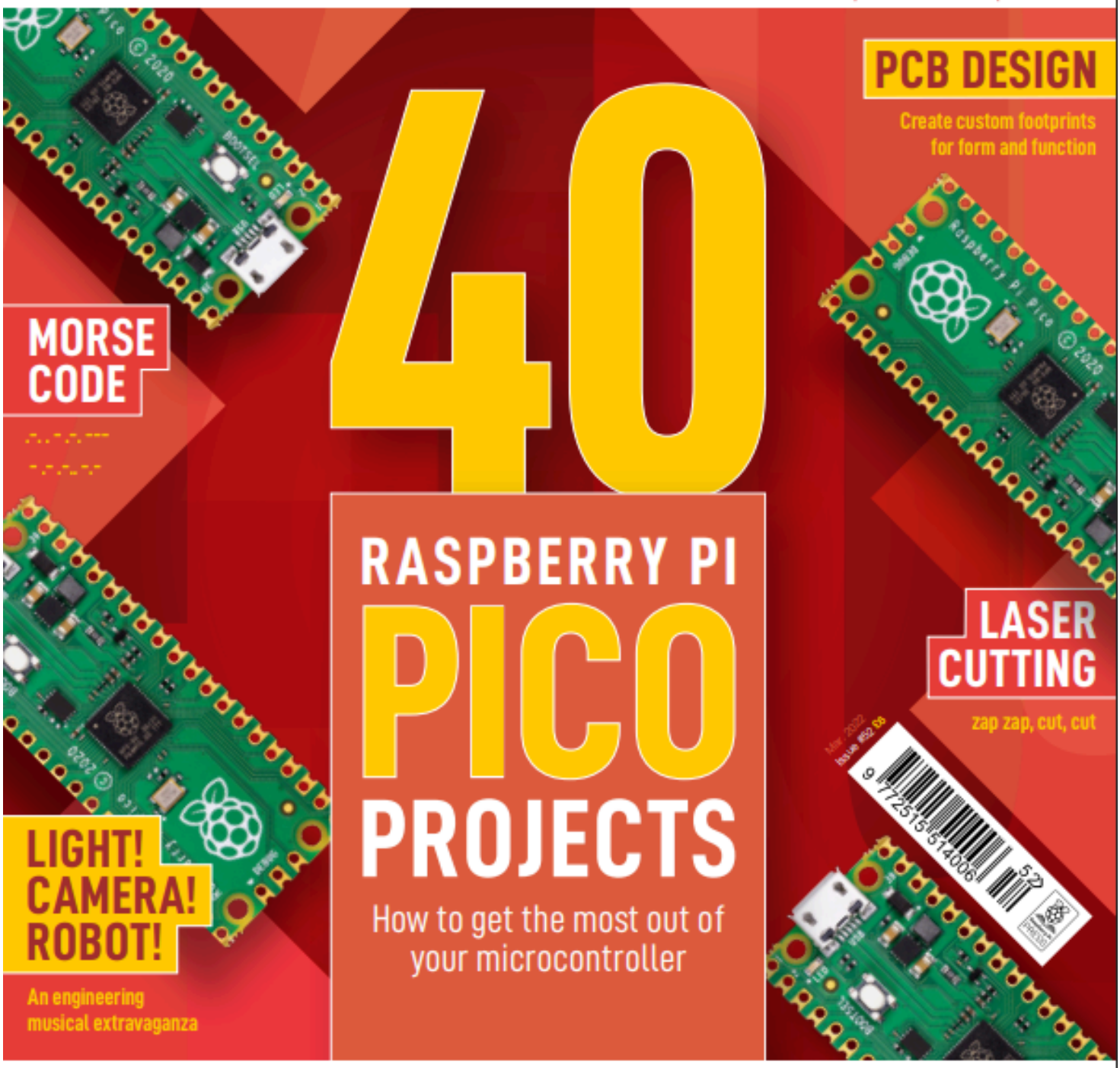
TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

Bios

HackSpace

TECHNOLOGY IN YOUR HANDS

hsmag.cc | March 2022 | Issue #52



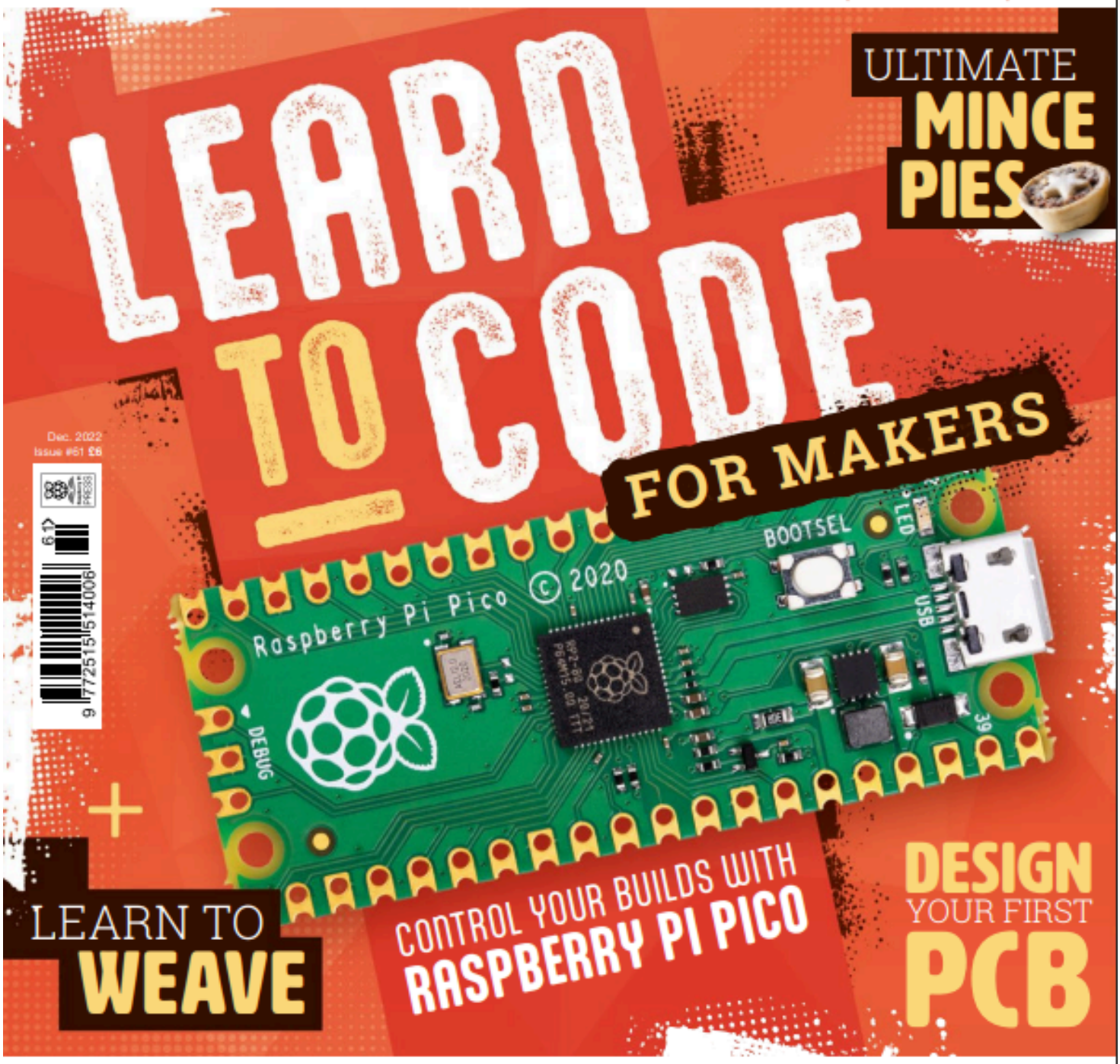
LEDS PGA2040 SENSORS FILAMENT

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 Bios

HackSpace

TECHNOLOGY IN YOUR HANDS

hsmag.CC | December 2022 | Issue #61



SPANNERS MUSIC 3D PRINTING DRAGONS

The MagPi

Issue 109 | September 2021 | magpi.cc

The official Raspberry Pi magazine

Raspberry Pi 4
SID Synth

Upcycling a
Game Boy

JOYSTICK

MIDI FIGHTER

PICO ROBOT

RASPBERRY PI
20 PICO PROJECTS

Make incredible objects with microcontroller boards!

Raspberry Pi
Student Guide

42 PAGES OF PROJECTS & TUTORIALS

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

Bios

RP2040 Datasheet

A microcontroller

by Raspberry Pi

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

Bios

Getting started with Raspberry Pi Pico

C/C++ development with Raspberry Pi Pico and other RP2040-based microcontroller boards

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 Bios

Raspberry Pi Pico C/C++ SDK

Libraries and tools for
C/C++ development on
RP2040 microcontrollers

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5

Bios

Raspberry Pi Pico Python SDK

A MicroPython environment for RP2040 microcontrollers

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 Bios

TalOS | DuetPi | Ultramarine Linux | 458 OS | 293 OS Pi 4B | 43 OS Pi5 Bios

Hardware design with RP2040

Using RP2040 microcontrollers to build boards and products

Alpakka Controller DIY:

- https://inputlabs.io/alpakka/manual/diy_pcb

Subscribe @InputLabs--Web

```
import machine
import utime

import array, time
from machine import Pin
import rp2
from rp2 import PIO, StateMachine, asm_pio
# LED引脚
led_onboard = machine.Pin(25, machine.Pin.OUT)


# Configure the number of WS2812 LEDs.
NUM_LEDS = 10

@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT, autopull=True, pull_thresh=24)
def ws2812():
    T1 = 2
    T2 = 5
    T3 = 3
    label("bitloop")
    out(x, 1)               .side(0)    [T3 - 1]
    jmp(not_x, "do_zero")   .side(1)    [T1 - 1]
    jmp("bitloop")          .side(1)    [T2 - 1]
    label("do_zero")
    nop()                   .side(0)    [T2 - 1]

# Create the StateMachine with the ws2812 program, outputting on Pin(23).ws2812引脚
sm = StateMachine(0, ws2812, freq=8000000, sideset_base=Pin(23))

# Start the StateMachine, it will wait for data on its FIFO.
sm.active(1)

# Display a pattern on the LEDs via an array of LED RGB values.
ar = array.array("I", [0 for _ in range(NUM_LEDS)])

time.sleep_ms(200)

print("red")

for j in range(0, 20):
    for i in range(NUM_LEDS):
        ar[i] = j<<8
    sm.put(ar,8)
    time.sleep_ms(50)

print("green")
for j in range(0, 20):
    for i in range(NUM_LEDS):
        ar[i] = j<<16
    sm.put(ar,8)
    time.sleep_ms(50)

print("blue")
```

Programa precargado de--Another example from another manufacter(99tech):

```
# Example using PIO to drive a set of WS2812 LEDs.

import array, time
from machine import Pin
import rp2
```

```
@rp2.asm_pio(
    sideset_init=rp2.PIO.OUT_LOW,
    out_shiftdir=rp2.PIO.SHIFT_LEFT,
    autopull=True,
    pull_thresh=24,
)
def ws2812():
    # fmt: off
    T1 = 2
    T2 = 5
    T3 = 3
    wrap_target()
    label("bitloop")
    out(x, 1)               .side(0)    [T3 - 1]
    jmp(not_x, "do_zero")   .side(1)    [T1 - 1]
    jmp("bitloop")          .side(1)    [T2 - 1]
    label("do_zero")
    nop()                   .side(0)    [T2 - 1]
    wrap()
    # fmt: on


# Create the StateMachine with the ws2812 program, outputting on Pin(22).
sm = rp2.StateMachine(0, ws2812, freq=8_000_000, sideset_base=Pin(23))

# Start the StateMachine, it will wait for data on its FIFO.
```

Descargamos el archivo neopixel.py o lo abrimos y copiamos todo su contenido, creamos un arc llamado `neopixel.py` y lo guardamos en la Pico:

Download neopixel.py or open it and copy all his content and create a file called `neopixel.py` a save it to the Pico:

```
import array, time
from machine import Pin
import rp2


# PIO state machine for RGB. Pulls 24 bits (rgb -> 3 * 8bit) automatically
@rp2.asm_pio(sideset_init=rp2.PIO.OUT_LOW, out_shiftdir=rp2.PIO.SHIFT_LEFT, autopull=True, pull_thr
def ws2812():
    T1 = 2
    T2 = 5
    T3 = 3
    wrap_target()
    label("bitloop")
    out(x, 1)               .side(0)    [T3 - 1]
    jmp(not_x, "do_zero")   .side(1)    [T1 - 1]
    jmp("bitloop")          .side(1)    [T2 - 1]
    label("do_zero")
    nop()                   .side(0)    [T2 - 1]
    wrap()


# PIO state machine for RGBW. Pulls 32 bits (rgbw -> 4 * 8bit) automatically
@rp2.asm_pio(sideset_init=rp2.PIO.OUT_LOW, out_shiftdir=rp2.PIO.SHIFT_LEFT, autopull=True, pull_thr
def sk6812():
    T1 = 2
    T2 = 5
    T3 = 3
    wrap_target()
    label("bitloop")
    out(x, 1)               .side(0)    [T3 - 1]
    jmp(not_x, "do_zero")   .side(1)    [T1 - 1]
    jmp("bitloop")          .side(1)    [T2 - 1]
    label("do_zero")
    nop()                   .side(0)    [T2 - 1]
    wrap()


# we need this because Micropython can't construct slice objects directly, only by
# way of supporting slice notation.
# So, e.g. slice_maker[1::4] gives a slice(1,None,4) object.
class slice_maker_class:
    def __getitem__(self, slc):
        return slc


slice_maker = slice_maker_class()


# Delay here is the reset time. You r  a pause to reset the LED strip back to the initial LED
# however, if you have quite a bit o  ocessing to do before the next time you update the strip
# you could put in delay=0 (or a lower delay)
#
```

```
# When dealing with just 'RGB' (3 letter string), this means same but reduced by 1 after XOR!.
# Example: in 'GRBW' we want final form of 0bGGRRBBWW, meaning G with index 0 needs to be shifte
# 'G' on index 0: 0b00 ^ 0b11 -> 0b11 (3), just as we wanted.
# Same hold for every other index (and - 1 at the end for 3 letter strings).

class Neopixel:
    # Micropython doesn't implement __slots__, but it's good to have a place
    # to describe the data members...
    # __slots__ = [
    #    'num_leds',   # number of LEDs
    #    'pixels',     # array.array('I') of raw data for LEDs
    #    'mode',       # mode 'RGB' etc
    #    'W_in_mode',  # bool: is 'W' in mode
    #    'sm',         # state machine
    #    'shift',      # shift amount for each component, in a tuple for (R,B,G,W)
    #    'delay',      # delay amount
    #    'brightnessvalue', # brightness scale factor 1..255
    # ]

    def __init__(self, num_leds, state_machine, pin, mode="RGB", delay=0.0001):
        """
        Constructor for library class

        :param num_leds:  number of leds on your led-strip
        :param state_machine: id of PIO state machine used
        :param pin: pin on which data line to led-strip is connected
        :param mode: [default: "RGB"] mode and order of bits representing the color value.
        This can be any order of RGB or RGBW (neopixels are usually GRB)
        :param delay: [default: 0.0001] delay used for latching of leds when sending data
        """
        self.pixels = array.array("I", [0] * num_leds)
        self.mode = mode
        self.W_in_mode = 'W' in mode
        if self.W_in_mode:
            # RGBW uses different PIO state machine configuration
            self.sm = rp2.StateMachine(state_machine, sk6812, freq=8000000, sideset_base=Pin(pin))
            # tuple of values required to shift bit into position (check class desc.)
            self.shift = ((mode.index('R') ^ 3) * 8, (mode.index('G') ^ 3) * 8,
                          (mode.index('B') ^ 3) * 8, (mode.index('W') ^ 3) * 8)
        else:
            self.sm = rp2.StateMachine(state_machine, ws2812, freq=8000000, sideset_base=Pin(pin))
            self.shift = (((mode.index('R') ^ 3) - 1) * 8, ((mode.index('G') ^ 3) - 1) * 8,
                          ((mode.index('B') ^ 3) - 1) * 8, 0)
        self.sm.active(1)
        self.num_leds = num_leds
        self.delay = delay
        self.brightnessvalue = 255

    def brightness(self, brightness=None):
        """
        Set the overall value to adjust brightness when updating leds
        or return class brightnessvalue if brightness is None

        :param brightness: [default: None] Value of brightness on interval 1..255
        :return: class brightnessvalue member or None
        """
        if brightness is None:
            return self.brightnessvalue
        else:
            if brightness < 1:
                brightness = 1
```

```
def set_pixel_line_gradient(self, pixel1, pixel2, left_rgb_w, right_rgb_w, how_bright=None):
    """
    Create a gradient with two RGB colors between "pixel1" and "pixel2" (inclusive)

    :param pixel1: Index of starting pixel (inclusive)
    :param pixel2: Index of ending pixel (inclusive)
    :param left_rgb_w: Tuple of form (r, g, b) or (r, g, b, w) representing starting color
    :param right_rgb_w: Tuple of form (r, g, b) or (r, g, b, w) representing ending color
    :param how_bright: [default: None] Brightness of current interval. If None, use global brig
    :return: None
    """
    if pixel2 - pixel1 == 0:
        return
    right_pixel = max(pixel1, pixel2)
    left_pixel = min(pixel1, pixel2)

    with_W = len(left_rgb_w) == 4 and self.W_in_mode
    r_diff = right_rgb_w[0] - left_rgb_w[0]
    g_diff = right_rgb_w[1] - left_rgb_w[1]
    b_diff = right_rgb_w[2] - left_rgb_w[2]
    if with_W:
        w_diff = (right_rgb_w[3] - left_rgb_w[3])

    for i in range(right_pixel - left_pixel + 1):
        fraction = i / (right_pixel - left_pixel)
        red = round(r_diff * fraction + left_rgb_w[0])
        green = round(g_diff * fraction + left_rgb_w[1])
        blue = round(b_diff * fraction + left_rgb_w[2])
        # if it's (r, g, b, w)
        if with_W:
            white = round(w_diff * fraction + left_rgb_w[3])
            self.set_pixel(left_pixel + i, (red, green, blue, white), how_bright)
        else:
            self.set_pixel(left_pixel + i, (red, green, blue), how_bright)

def set_pixel_line(self, pixel1, pixel2, rgb_w, how_bright=None):
    """
    Set an array of pixels starting from "pixel1" to "pixel2" (inclusive) to the desired color.

    :param pixel1: Index of starting pixel (inclusive)
    :param pixel2: Index of ending pixel (inclusive)
    :param rgb_w: Tuple of form (r, g, b) or (r, g, b, w) representing color to be used
    :param how_bright: [default: None] Brightness of current interval. If None, use global brig
    :return: None
    """
    if pixel2 >= pixel1:
        self.set_pixel(slice_maker[pixel1:pixel2 + 1], rgb_w, how_bright)

def set_pixel(self, pixel_num, rgb_w, how_bright=None):
    """
    Set red, green and blue (+ white) value of pixel on position <pixel_num>
    pixel_num may be a 'slice' object, and then the operation is applied
    to all pixels implied by the slice (most useful when called via __setitem__)

    :param pixel_num: Index of pixel to be set or slice object representing multiple leds
    :param rgb_w: Tuple of form (r, g, b) or (r, g, b, w) representing color to be used
    :param how_bright: [default: None] Brightness of current interval. If None, use global brig
    :return: None
    """
    if how_bright is None:
        how_bright = self.brightness()
    sh_R, sh_G, sh_B, sh_W = self.shift
```

```
    blue = round(rgb_w[2] * bratio)
    white = 0
    # if it's (r, g, b, w)
    if len(rgb_w) == 4 and self.W_in_mode:
        white = round(rgb_w[3] * bratio)

    pix_value = white << sh_W | blue << sh_B | red << sh_R | green << sh_G
    # set some subset, if pixel_num is a slice:
    if type(pixel_num) is slice:
        for i in range(*pixel_num.indices(self.num_leds)):
            self.pixels[i] = pix_value
    else:
        self.pixels[pixel_num] = pix_value

def get_pixel(self, pixel_num):
    """
    Get red, green, blue and white (if applicable) values of pixel on position <pixel_num>

    :param pixel_num: Index of pixel to be set
    :return rgb_w: Tuple of form (r, g, b) or (r, g, b, w) representing color to be used
    """
    balance = self.pixels[pixel_num]
    sh_R, sh_G, sh_B, sh_W = self.shift
    if self.W_in_mode:
        w = (balance >> sh_W) & 255
    b = (balance >> sh_B) & 255
    r = (balance >> sh_R) & 255
    g = (balance >> sh_G) & 255
    red = int(r * 255 / self.brightness() )
    green = int(g * 255 / self.brightness() )
    blue = int(b * 255 / self.brightness() )
    if self.W_in_mode:
        white = int(w * 255 / self.brightness() )
        return (red,green,blue,white)
    else:
        return (red,green,blue)

def __setitem__(self, idx, rgb_w):
    """
    if npix is a Neopixel object,
    npix[10] = (0,255,0)        # <- sets #10 to green
    npix[15:21] = (255,0,0)     # <- sets 16,17 .. 20 to red
    npix[21:29:2] = (0,0,255)   # <- sets 21,23,25,27 to blue
    npix[1::2] = (0,0,0)        # <- sets all odd pixels to 'off'
    (the 'slice' cases pass idx as a 'slice' object, and
    set_pixel processes the slice)

    :param idx: Index can either be indexing number or slice
    :param rgb_w: Tuple of form (r, g, b) or (r, g, b, w) representing color to be used
    :return:
    """
    self.set_pixel(idx, rgb_w)

def colorHSV(self, hue, sat, val):
    """
    Converts HSV color to rgb tuple and returns it.
    The logic is almost the same as in Adafruit NeoPixel library:
    https://github.com/adafruit/Adafruit_NeoPixel so all the credits for that
    go directly to them (license: https://github.com/adafruit/Adafruit_NeoPixel/blob/master/COP

    :param hue: Hue component. Should be on interval 0..65535
    :param sat: Saturation component. Should be on interval 0..255
```

```
        hue %= 65536

    hue = (hue * 1530 + 32768) // 65536
    if hue < 510:
```

Creamos un nuevo archivo `main.py` en la Pico--Make a `main.py` file on the Pico:

En el copiamos el contenido del ejemplo--Copy the content of the file colorwave.py:

Debemos tener en cuenta--Take care of this:

- **numpix ->> N° pixel**
- **Neopixel(numpix, State Machine ID, GPIO, "GRB")**

En esta placa--In this board:

- **Neopixel(numpix, 0, 23,"GRB")**

```
# Example showing how functions, that accept tuples of rgb values,
# simplify working with gradients

import time
from neopixel import Neopixel

numpix = 60
strip = Neopixel(numpix, 0, 23, "GRB")
# strip = Neopixel(numpix, 0, 0, "GRBW")

red = (255, 0, 0)
orange = (255, 50, 0)
yellow = (255, 100, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
indigo = (100, 0, 90)
violet = (200, 0, 100)
colors_rgb = [red, orange, yellow, green, blue, indigo, violet]

# same colors as normaln rgb, just 0 added at the end
colors_rgbw = [color+tuple([0]) for color in colors_rgb]
colors_rgbw.append((0, 0, 0, 255))

# uncomment colors_rgbw if you have RGBW strip
colors = colors_rgb
# colors = colors_rgbw


step = round(numpix / len(colors))
current_pixel = 0
strip.brightness(50)
```

Salvamos y ejecutamos el programa--Save and execute

- neopixel.py + rainbow.py
- neopixel.py + set_range.py
- neopixel.py + smoothRainbow.py

Tutoriales Sensores

Anterior

Home

Siguiente