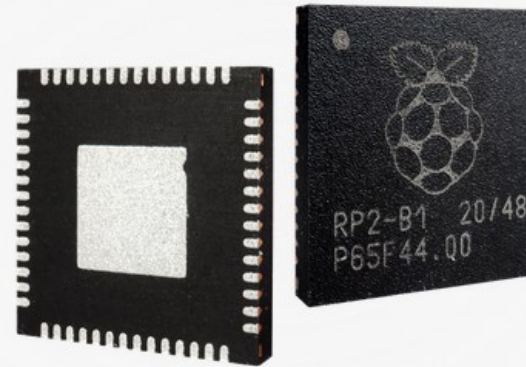
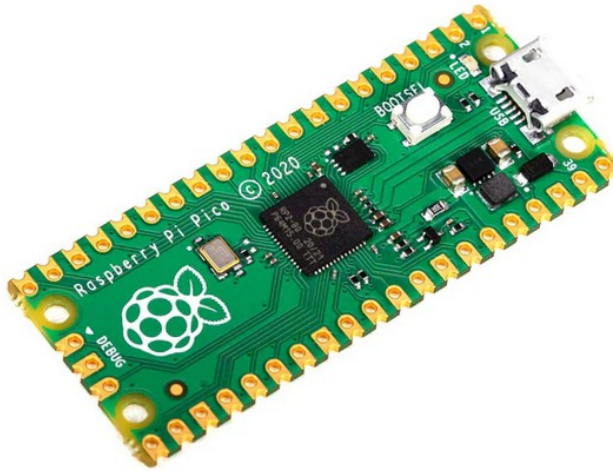


## RP2040



A microcontroller chip designed by  
**Raspberry Pi**

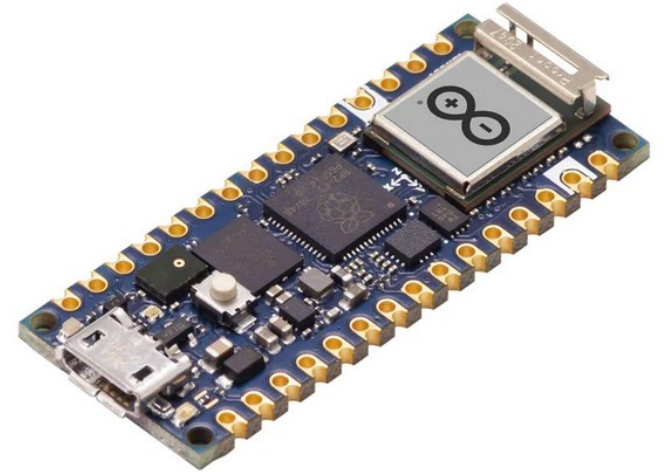
# RP2040 development platforms



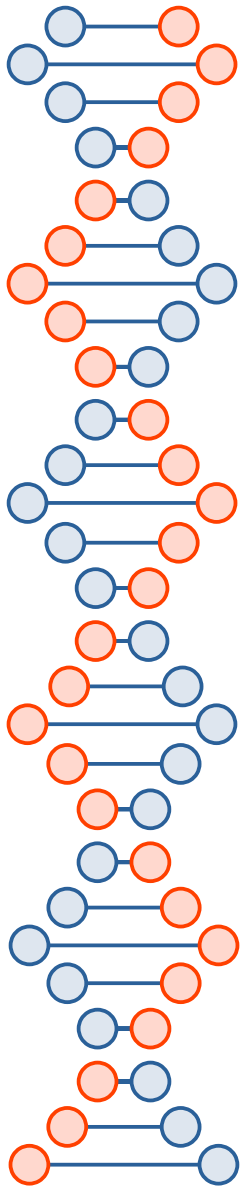
Raspberry Pi Pico



XIAO-RP2040

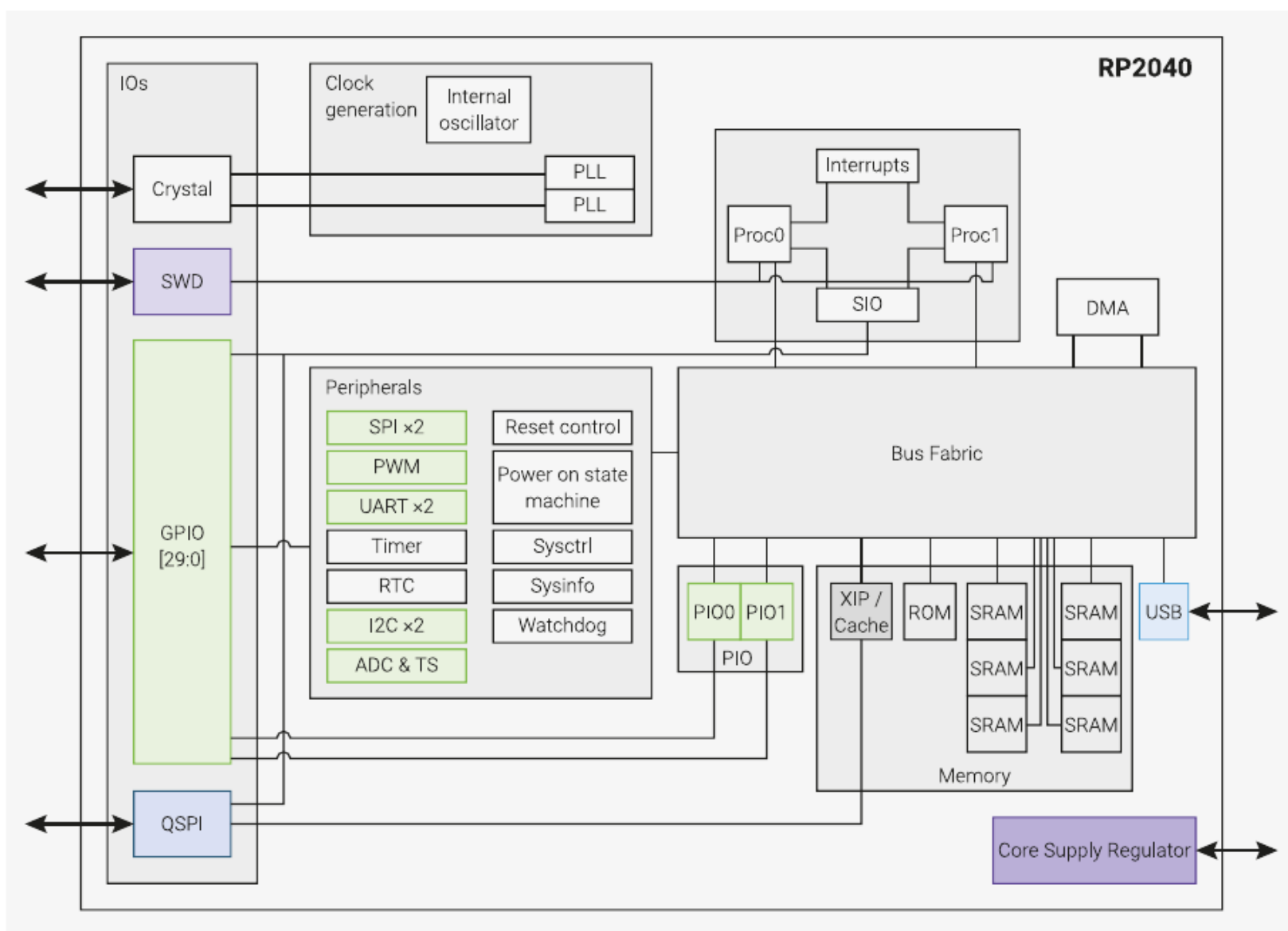
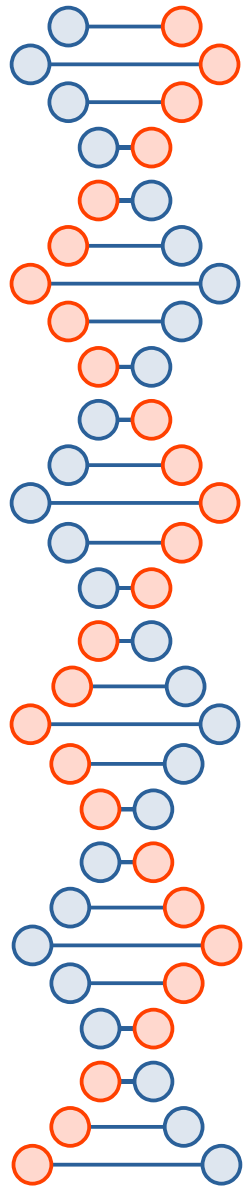


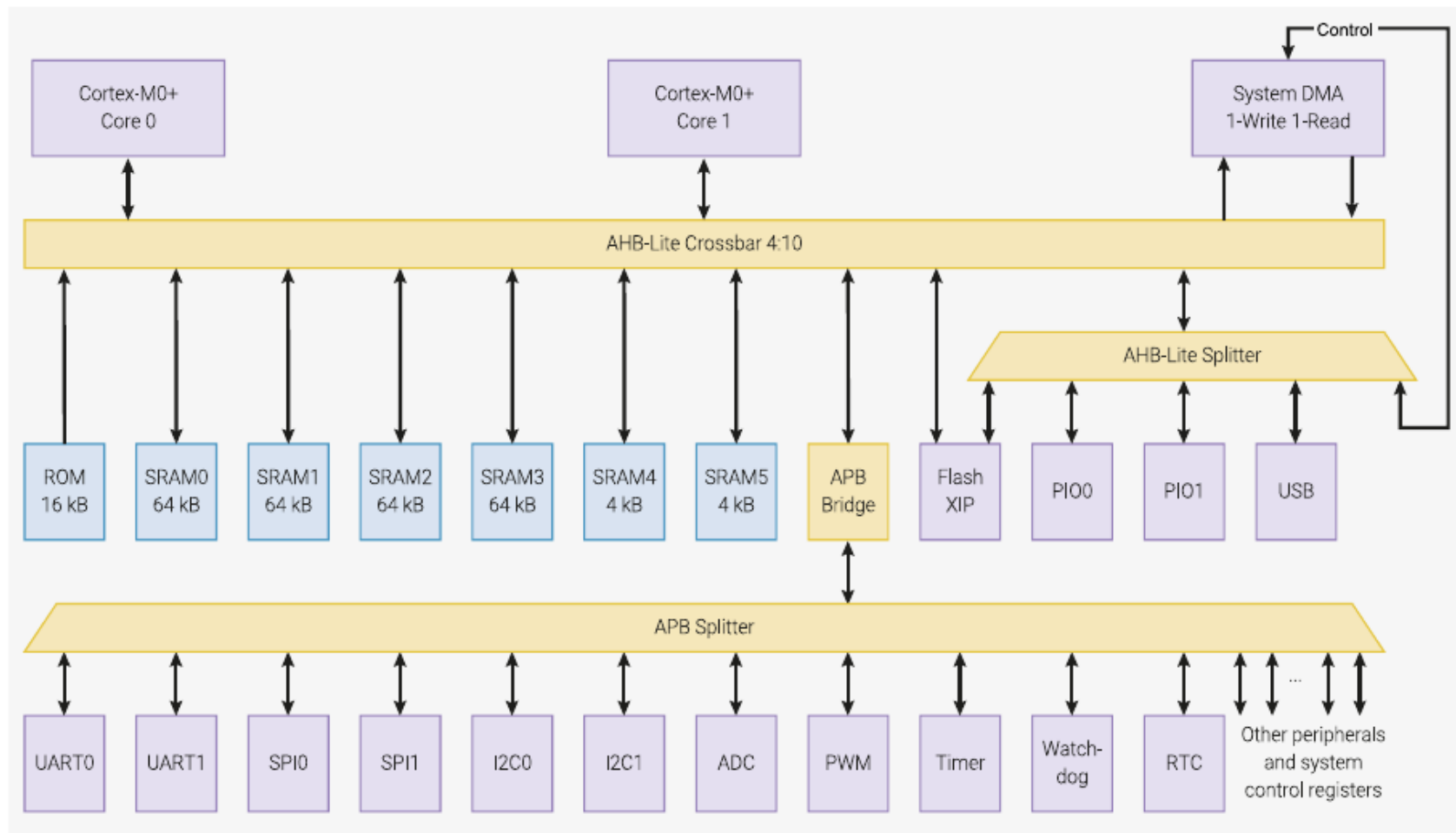
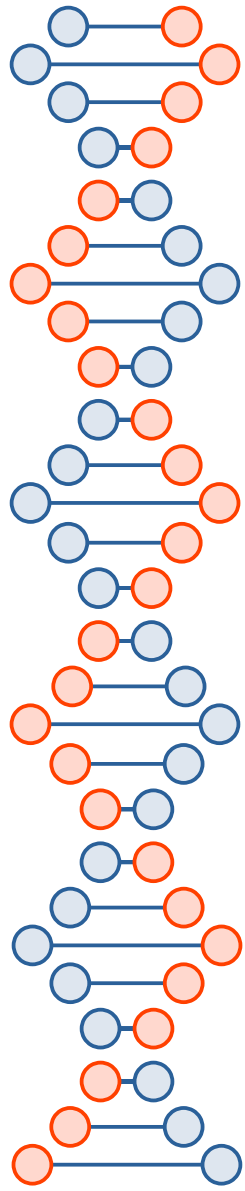
Arduino Nano RP2040



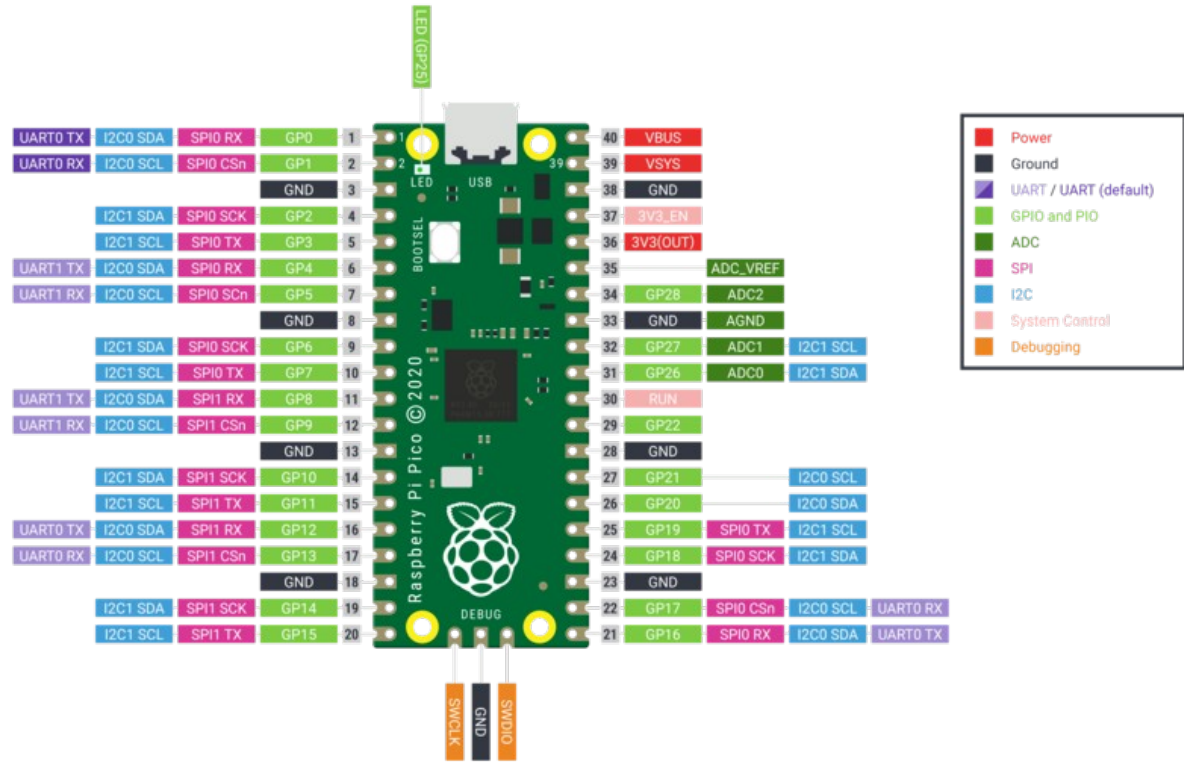
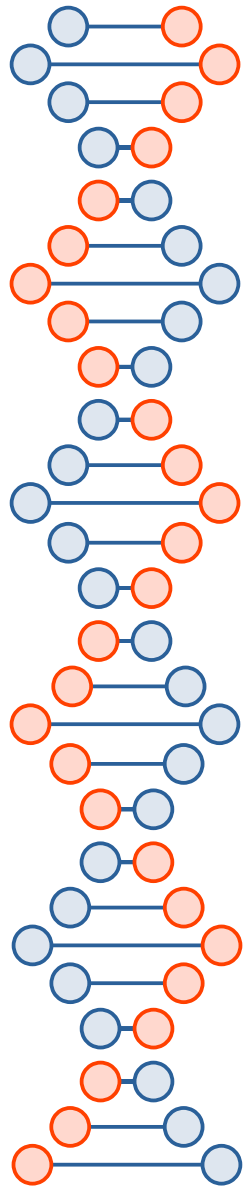
#### Key features:

- Dual ARM Cortex-M0+ @ 133MHz
- 264kB on-chip SRAM in six independent banks
- Support for up to 16MB of off-chip Flash memory via dedicated QSPI bus
- DMA controller
- Fully-connected AHB crossbar
- Interpolator and integer divider peripherals
- On-chip programmable LDO to generate core voltage
- 2 on-chip PLLs to generate USB and core clocks
- 30 GPIO pins, 4 of which can be used as analogue inputs
- Peripherals
  - 2 UARTs
  - 2 SPI controllers
  - 2 I2C controllers
  - 16 PWM channels
  - USB 1.1 controller and PHY, with host and device support
  - 8 PIO state machines



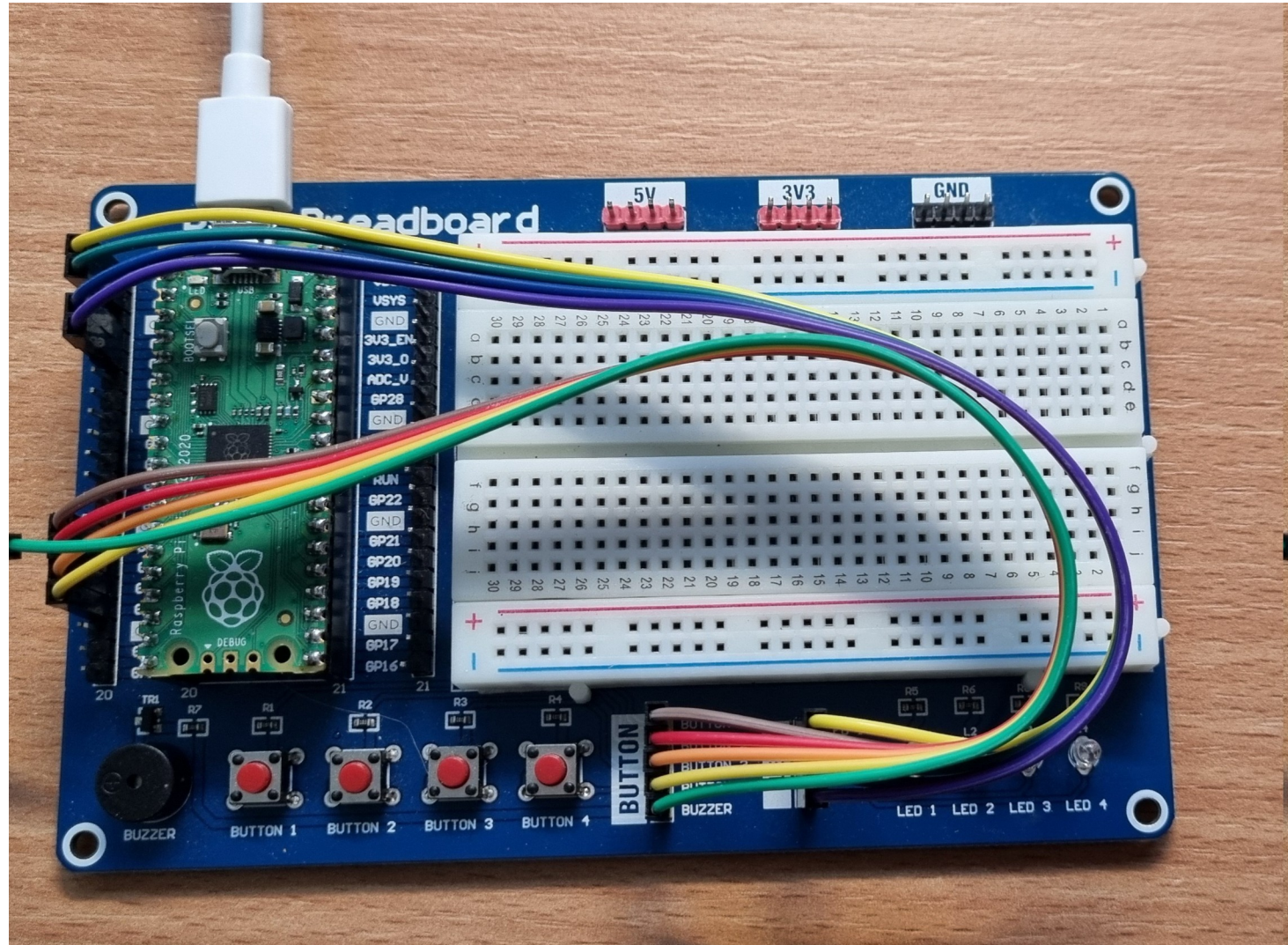
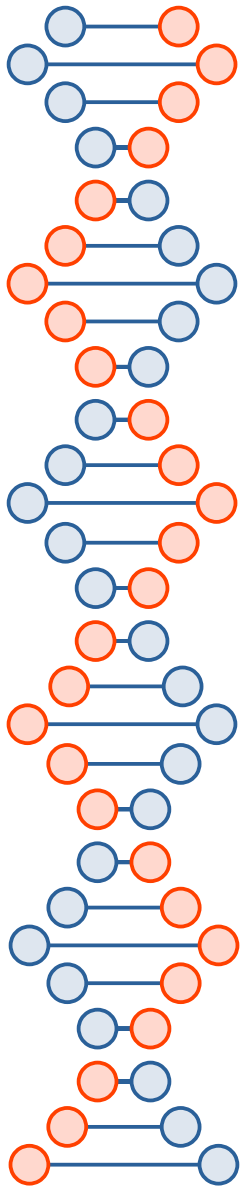


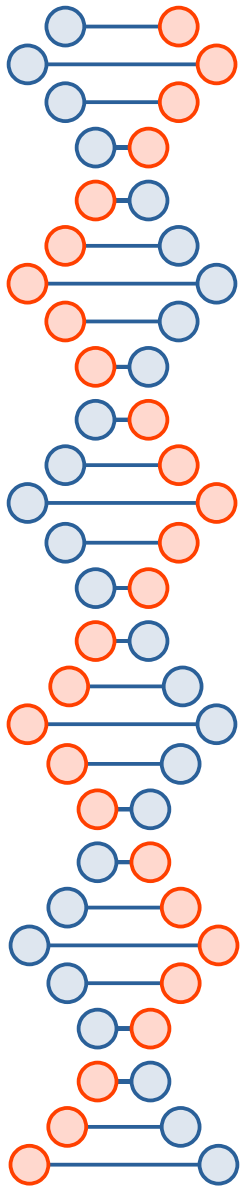
AHB Advanced High Performance Bus (4 x 32 bit transfer per cycle = 2 GB/s @ 125 MHz)  
 APB Advanced Peripheral bus (2 .. 4 cycles)  
 SIO Single cycle IO  
 XIP Execute in place (16 MB window)



GP0	Led 1		GP10	Btn 1		GP16	SPI0 MISO
GP1	Led 2		GP11	Btn 2		GP17	SPI0 CS
GP2	Led 3		GP12	Btn 3		GP18	SPI0 SCK
GP3	Led 4		GP13	Btn 4		GP19	SPI0 MOSI
GP14	Buzzer						







MicroPython

[FORUM](#)

[DOCS](#)

[QUICK-REF](#)

[DOWNLOAD](#)

[STORE](#)

[CONTACT](#)

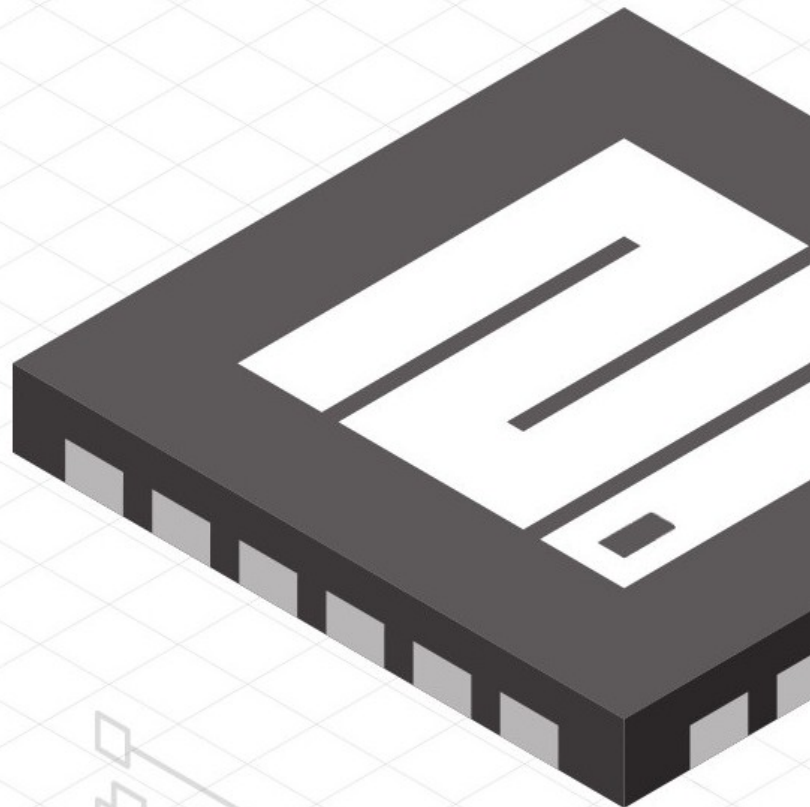
# MicroPython

MicroPython is a lean and efficient implementation of the **Python 3** programming language that includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments.

The MicroPython **pyboard** is a compact electronic circuit board that runs MicroPython on the bare metal, giving you a low-level Python operating system that can be used to control all kinds of electronic projects.

MicroPython is packed full of advanced features such as an interactive prompt, arbitrary precision integers, closures, list comprehension, generators, exception handling and more. Yet it is compact enough to fit and run within just 256k of code space and 16k of RAM.

MicroPython aims to be as compatible with normal Python as possible to allow you to transfer code with ease from the desktop to a microcontroller or embedded system.

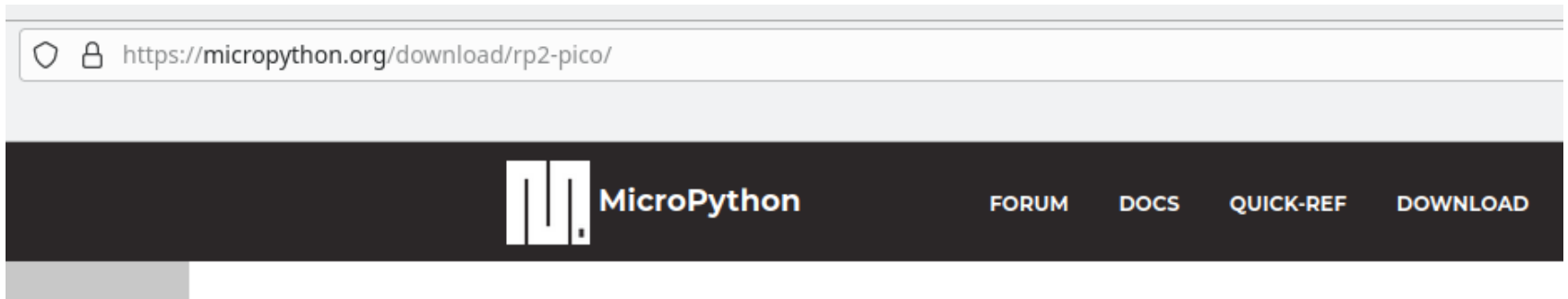
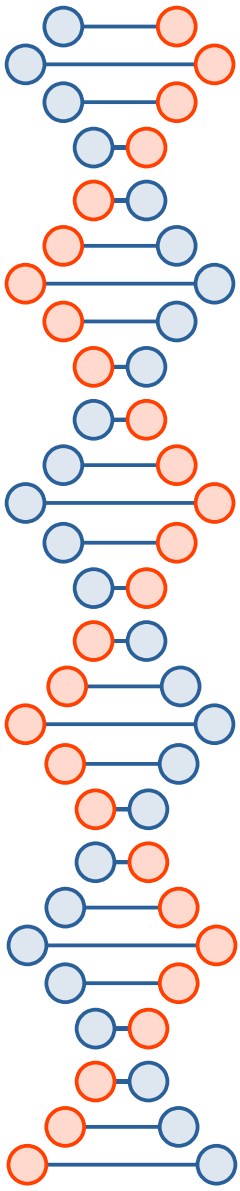


[TEST DRIVE A PYBOARD](#)

[BUY A PYBOARD](#)

[USE MICROPYTHON ONLINE](#)





## Installation instructions

### Flashing via UF2 bootloader

To get the board in bootloader mode ready for the firmware update, execute `machine.bootloader()` at the MicroPython REPL. Alternatively, hold down the BOOTSEL button while plugging the board into USB. The uf2 file below should then be copied to the USB mass storage device that appears. Once programming of the new firmware is complete the device will automatically reset and be ready for use.

## Firmware

### Releases

**v1.24.1 (2024-11-29) .uf2** / [\[Release notes\]](#) (latest)

v1.24.0 (2024-10-25) .uf2 / [\[Release notes\]](#)

v1.23.0 (2024-06-02) .uf2 / [\[Release notes\]](#)

MySQL - Knoppix NPO

Quick reference for the pyboard

Quick reference for the ESP8266

Quick reference for the ESP32

## Quick reference for the RP2

General information about the RP2xxx port

Getting started with MicroPython on the RP2xxx

Installing MicroPython

General board control

Delay and timing

Timers

Pins and GPIO

Programmable IO (PIO)

UART (serial bus)

PWM (pulse width modulation)

ADC (analog to digital conversion)

Software SPI bus

Hardware SPI bus

Software I2C bus

Hardware I2C bus

I2S bus

Real time clock (RTC)

```
time.sleep_us(10)      # sleep for 10 microseconds
start = time.ticks_ms() # get millisecond counter
delta = time.ticks_diff(time.ticks_ms(), start) # compute time difference
```

## Timers

RP2040's system timer peripheral provides a global microsecond timebase and generates interrupts for it. The software timer is available currently, and there are unlimited number of them (memory permitting). There is no need to specify the timer id (id=-1 is supported at the moment) as it will default to this.

Use the `machine.Timer` class:

```
from machine import Timer

tim = Timer(period=5000, mode=Timer.ONE_SHOT, callback=lambda t:print(1))
tim.init(period=2000, mode=Timer.PERIODIC, callback=lambda t:print(2))
```

## Pins and GPIO

Use the `machine.Pin` class:

```
from machine import Pin

p0 = Pin(0, Pin.OUT) # create output pin on GPIO0
p0.on()               # set pin to "on" (high) level
p0.off()              # set pin to "off" (low) level
p0.value(1)           # set pin to on/high
```