Programmering af Mobile Robotter

RB1-PMR – Module 3: Basic IO programming using MicroPython

Jes Hundevadt Jepsen | jegj@mmmi.sdu.dk
SDU Drone Center, MMMI, University of Southern Denmark, Odense, Denmark

Agenda

- Recap of last module
- **Hardware walk-through**
  - RP2040 and Raspberry Pi Pico
  - Monk Makes Electronics Kit 1 for Pico (lite edition)
    - Jump-wires, breadboard, electrical components, etc.
- **Software walk-through**
  - Flash the latest firmware to your Pico
  - Thonny Python IDE
- **Introduction to IO programming** using MicroPython
  - Standard **libraries** and **micro-libraries**
  - Key modules, classes and functions (Machine, Pin, and Timer)

- Introduction to "Portfolio 1: 7-Segment Display controller"

- Assignments

Recap

- **Introduction to programming**
  - High-level, mid-level, low-level
  - Syntax
  - Etc.

- **Basic concepts in programming** (using Python)
  - Operators (Arithmetic, conditional, and Logical)
  - Built-in **data types** and **functions**
  - **Conditional statements** and **loops**
  - **Functions** and error handling
  - Basic **troubleshooting** and **debugging**

Recap

- **Introduction to programming**
  - High-level, mid-level, low-level
  - Syntax
  - Etc.

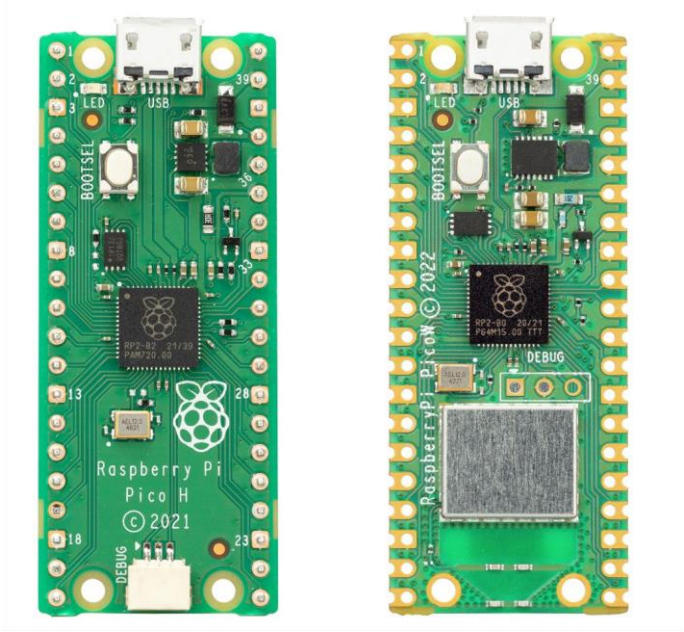
- **Basic concepts in programming** (using Python)
  - Operators (Arithmetic, conditional, and Logical)
  - Built-in **data types** and **functions**
  - **Conditional statements** and **loops**
  - **Functions** and error handling
  - Basic **troubleshooting** and **debugging**

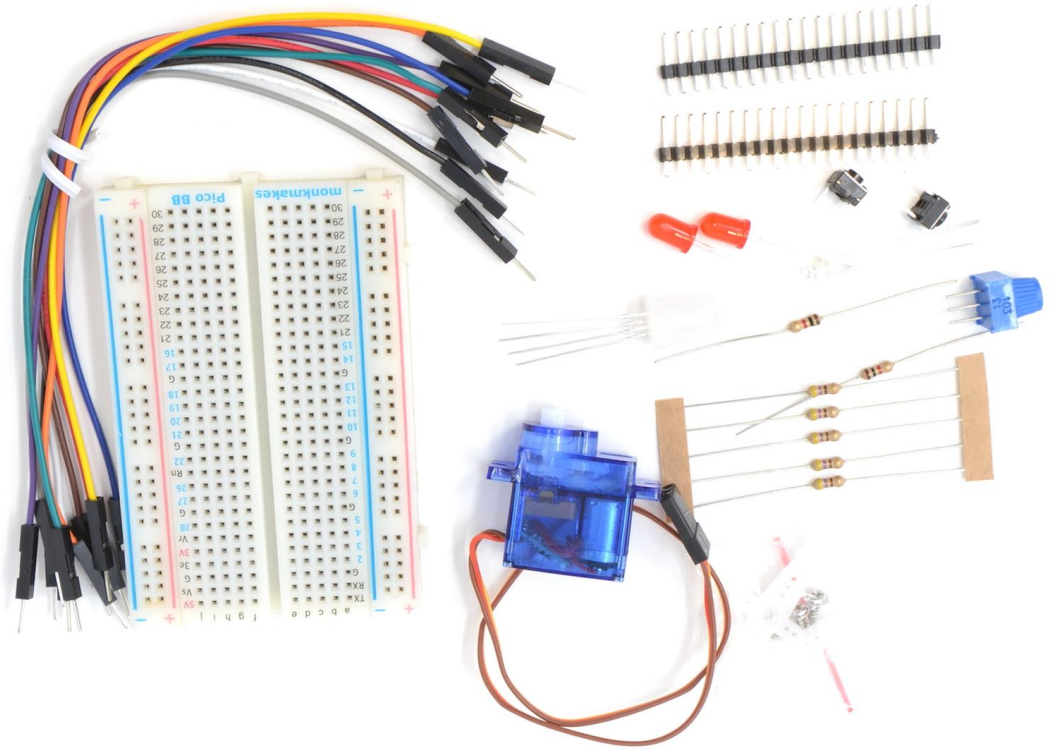
- **Extra Credit Activity 1: Calculator**
  - Remember to*: 1) insert your name and mail, and 2) source code*
  - Any questions?

- **Assignments**
  - …more like a guide / tutorial
  - Any questions?

  - Need more training / exercises? Try:
    - W3 schools: https://www.w3schools.com/python/
    - Youtube?

Hardware walk-through





How have tried it out yet? Or are familiar with the Pico?

Raspberry Pi models

- Raspberry Pi family (single-board computers, Model: 1 - 5, Zero / Zero W / Zero 2 W)
- Raspberry Pi Pico family (microcontroller board)

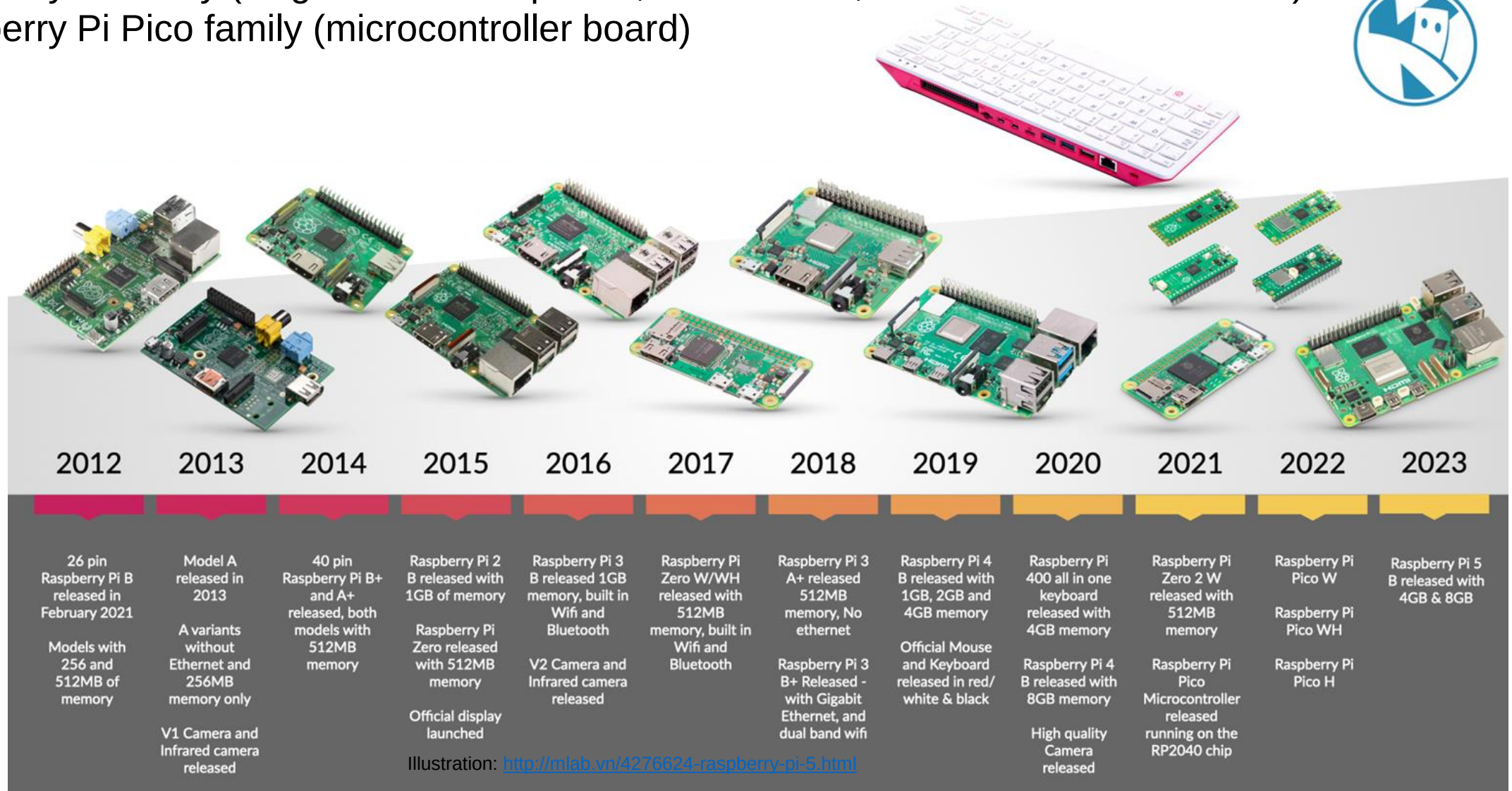


Illustration: http://mlab.vn/4276624-raspberry-pi-5.html

Raspberry Pi models

- Raspberry Pi family (single-board computers, Model: 1 - 5, Zero / Zero W / Zero 2 W)
- Raspberry Pi Pico family (microcontroller board)

Raspberry Pi Pico family

- Currently consists of four boards;
  - Raspberry Pi Pico (far left) / Pico H (middle left)
  - **Raspberry Pi Pico W (middle right) / Pico WH (far right)**

Raspberry Pi models

- Raspberry Pi family (single-board computers, Model: 1 - 5, Zero / Zero W / Zero 2 W)
- Raspberry Pi Pico family (microcontroller board)

Raspberry Pi Pico family

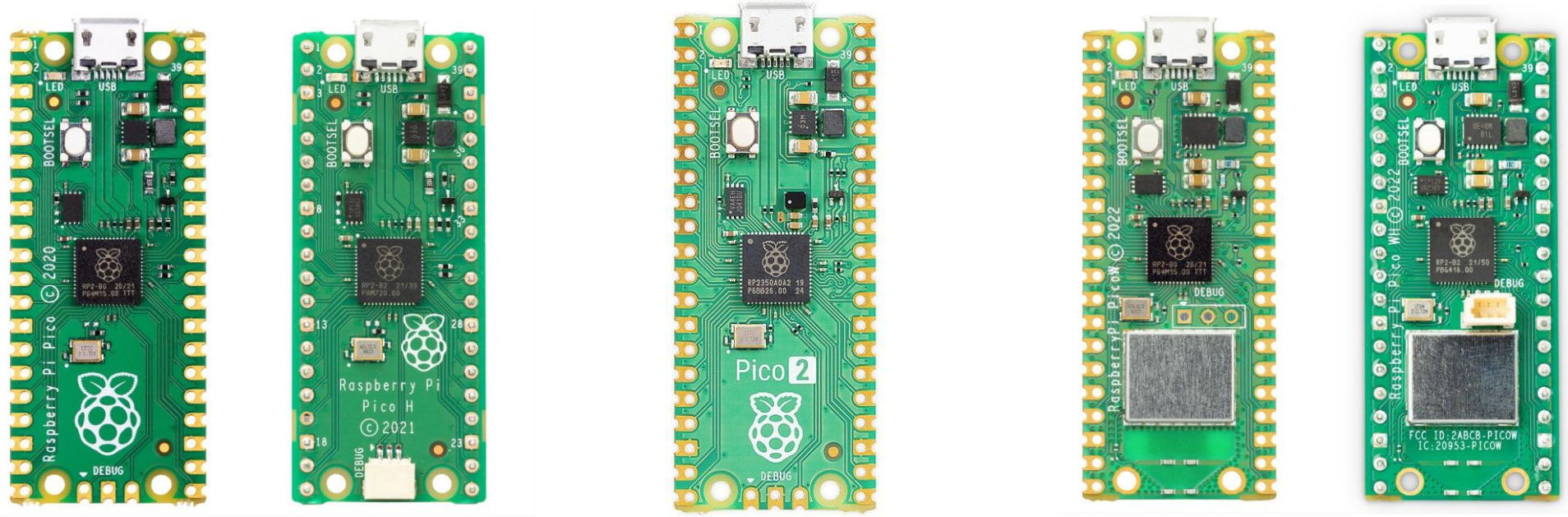
- Currently consists of four boards;
  - Raspberry Pi Pico (far left) / Pico H (middle left)
  - Raspberry Pi Pico W (middle right) / Pico WH (far right) - **W? H? WH? Whats is the difference?**

Raspberry Pi models

- Raspberry Pi family (single-board computers, Model: 1 - 5, Zero / Zero W / Zero 2 W)
- Raspberry Pi Pico family (microcontroller board)

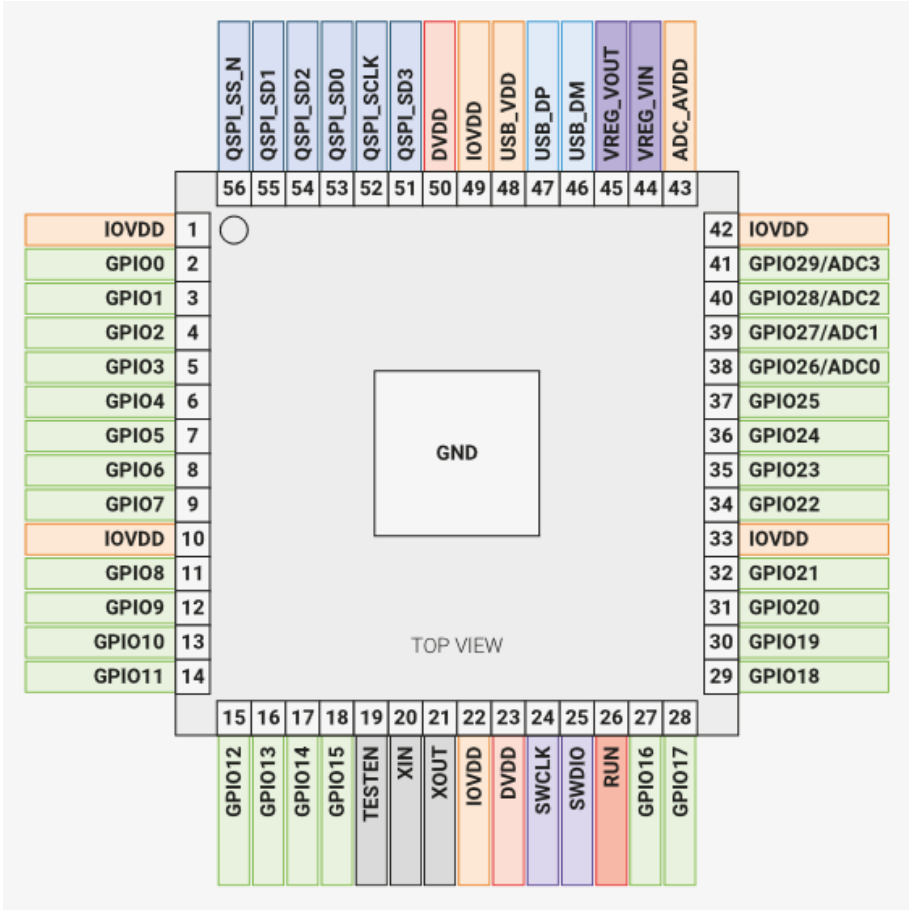
Raspberry Pi Pico family

- Currently consists of four boards;
  - Raspberry Pi Pico (far left) / Pico H (middle left)
  - Raspberry Pi Pico W (middle right) / Pico WH (far right)
  - **Raspberry Pi Pico 2 (middle)** (maybe next year…)



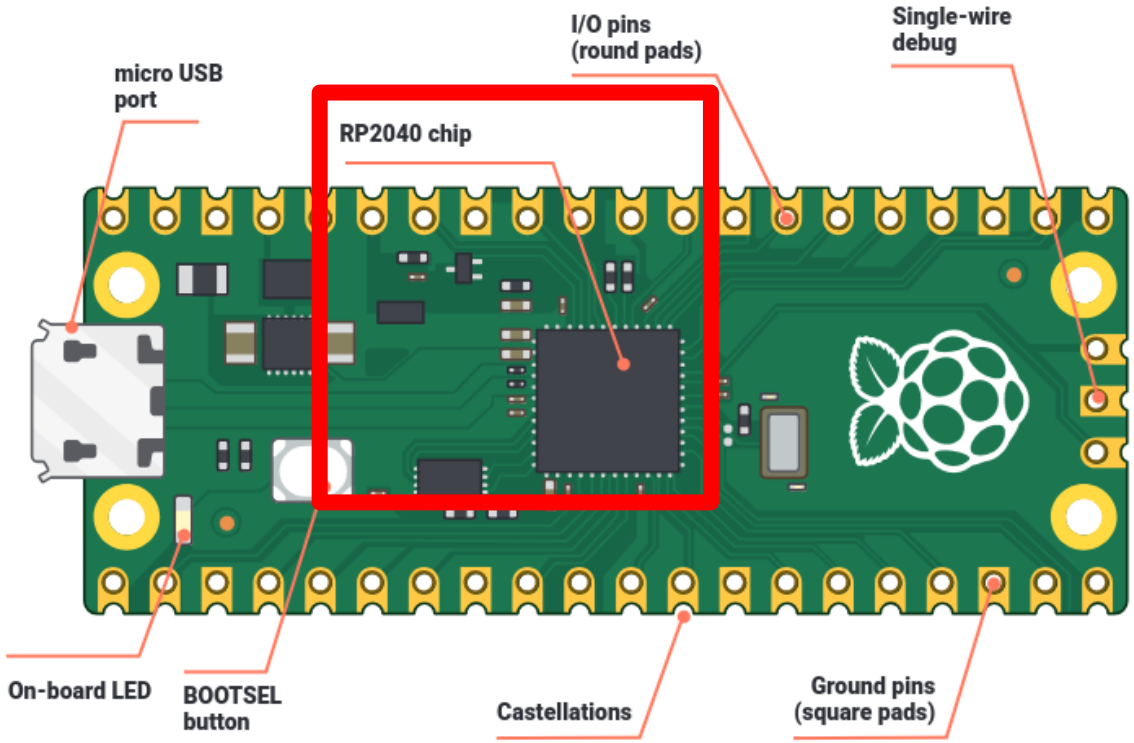
Raspberry Pi Pico and Pico W

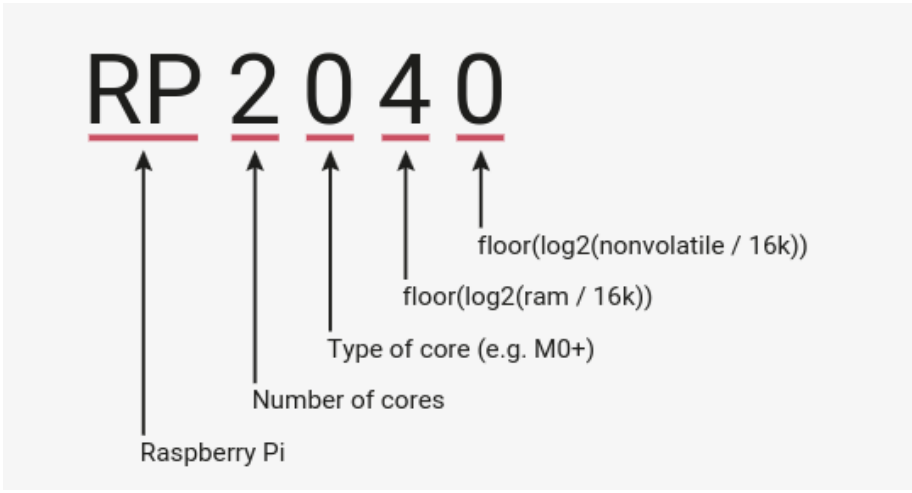
- **Raspberry Pi Pico**'s is **based on an RP2040**
  - More details in [RP2040 Datasheet]

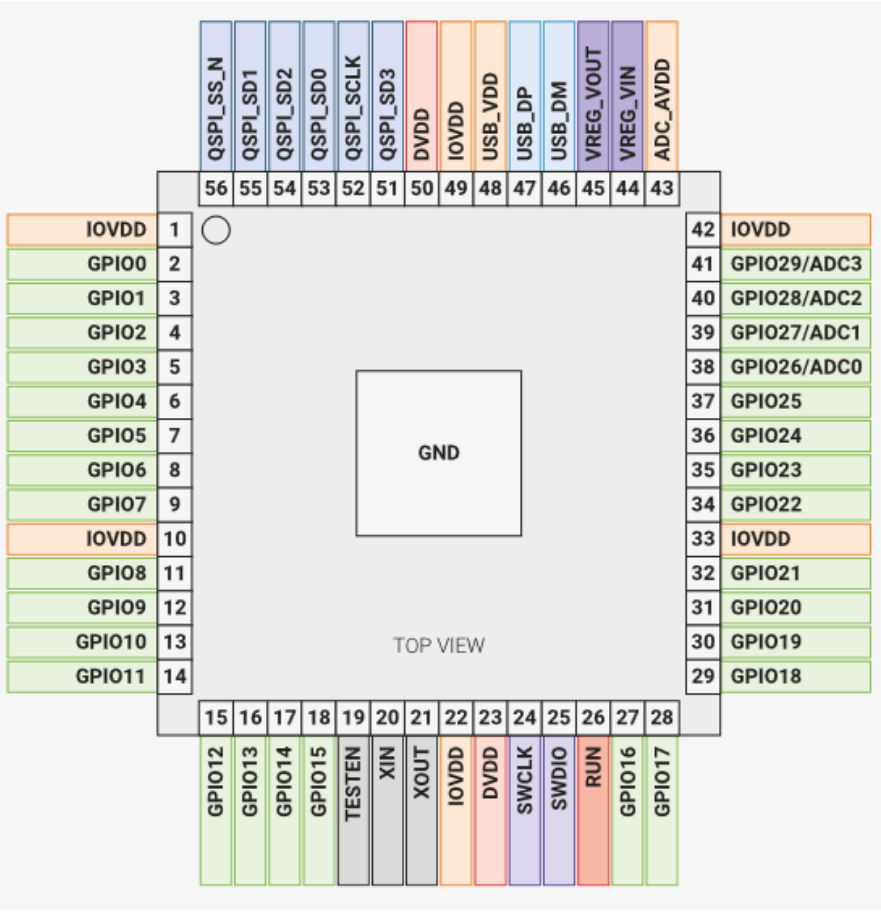


Raspberry Pi Pico and Pico W

- **Raspberry Pi Pico**'s is **based on an RP2040**
  - More details in [RP2040 Datasheet]

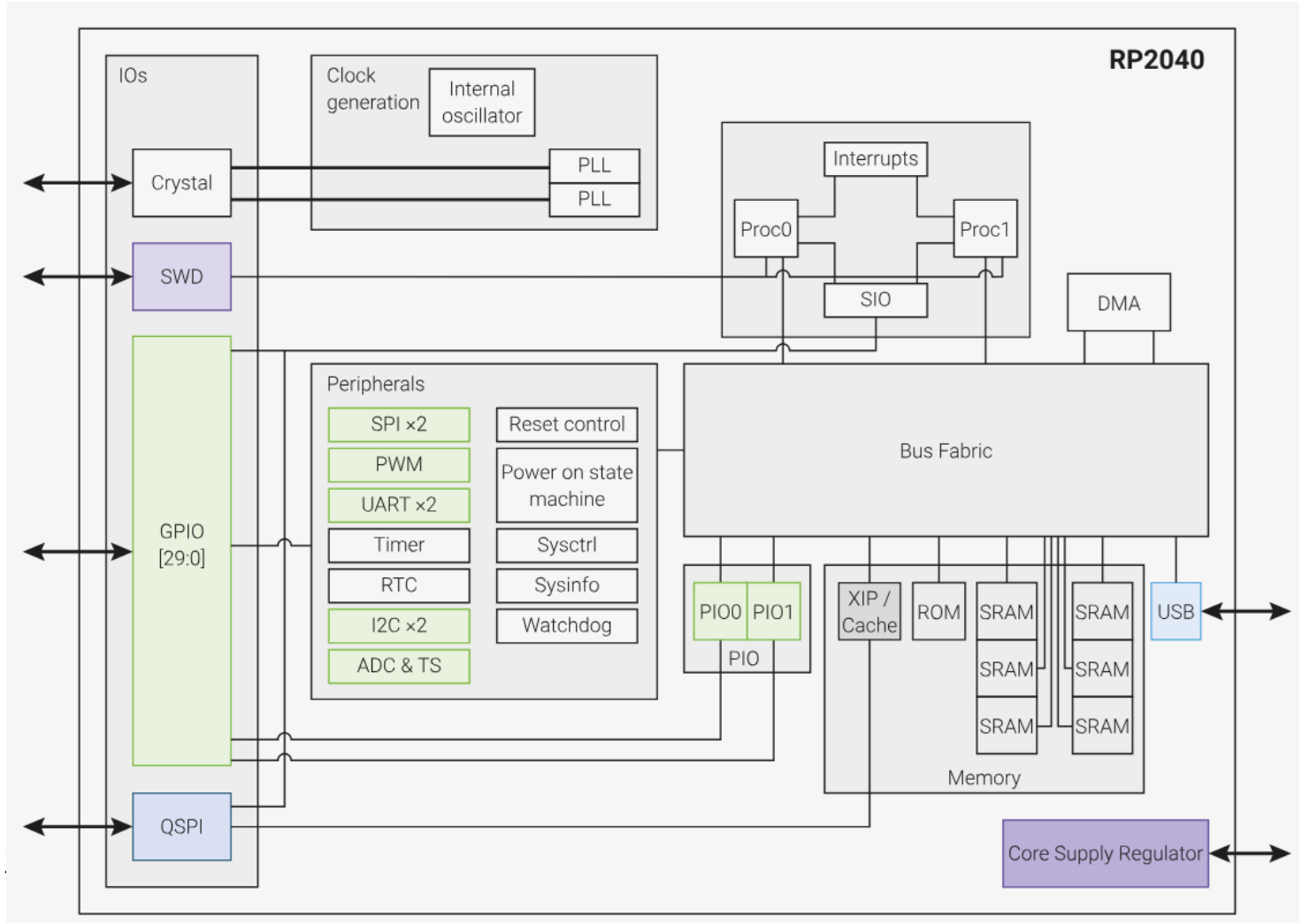


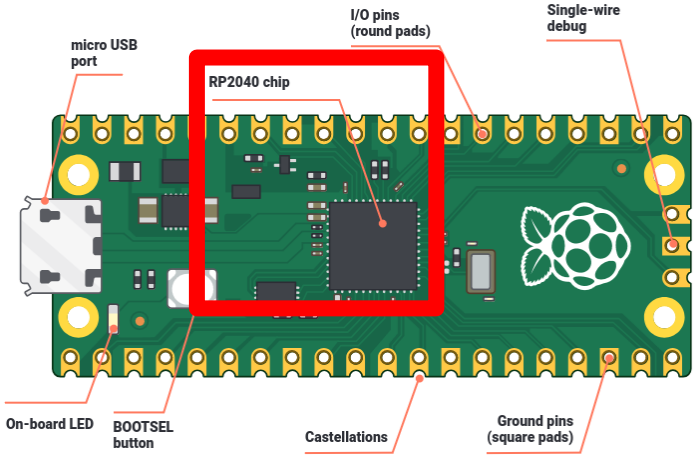


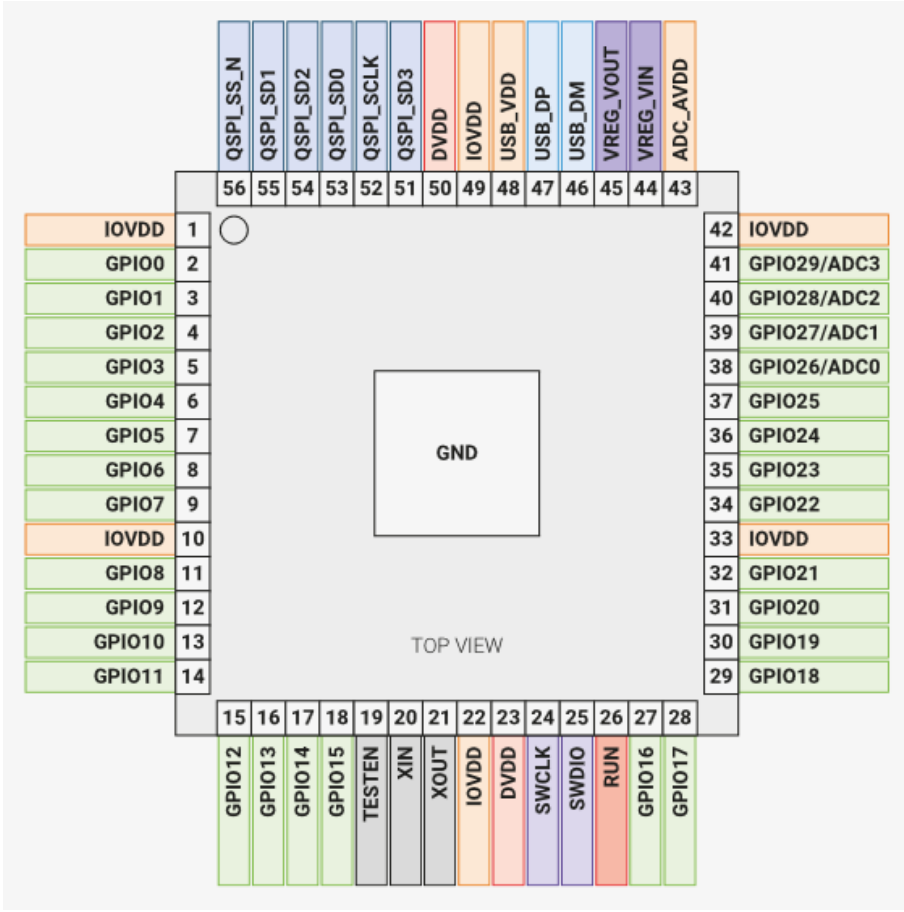


Raspberry Pi Pico and Pico W

- **Raspberry Pi Pico**'s is **based on an RP2040**
  - **More details in [RP2040 Datasheet]**

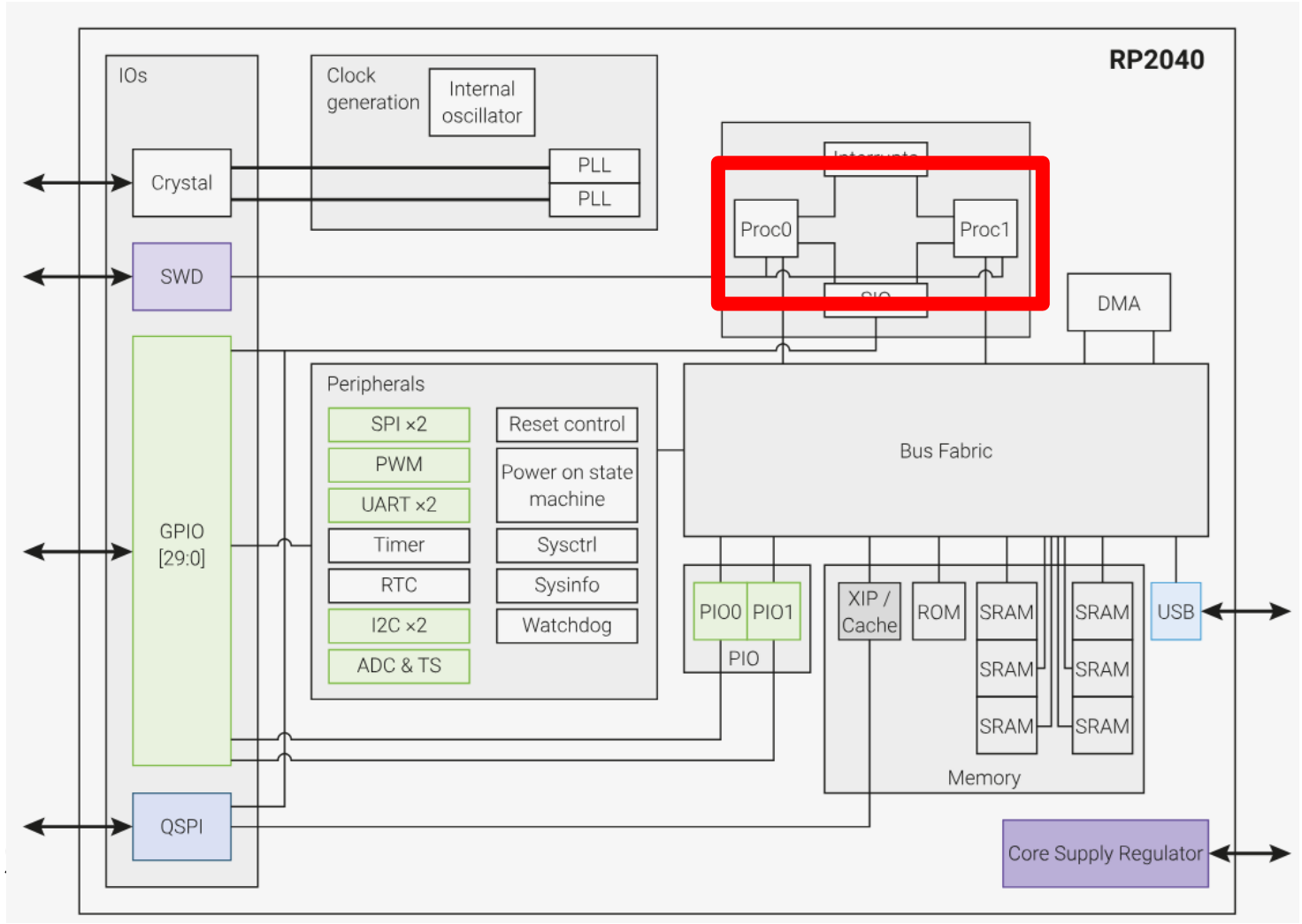


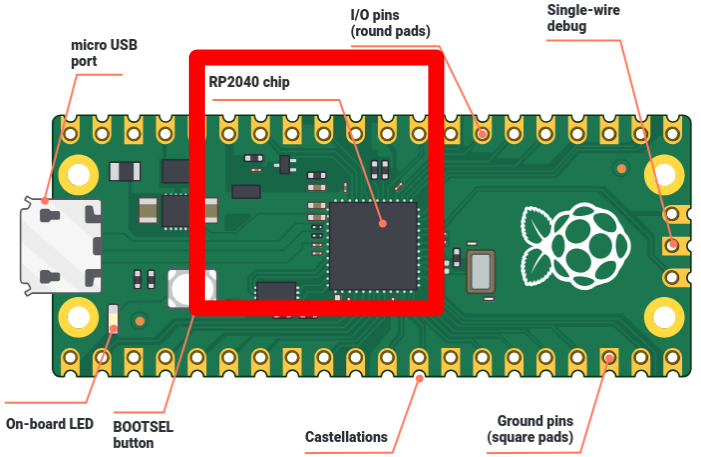


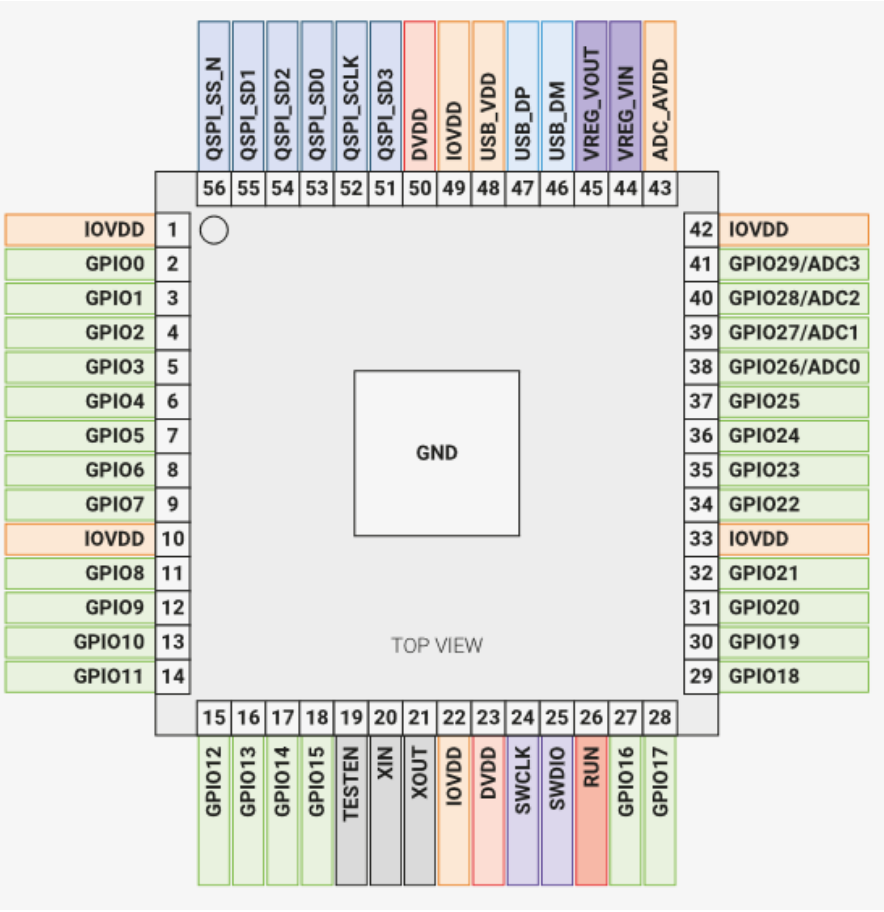


Raspberry Pi Pico and Pico W

- **Raspberry Pi Pico**'s is **based on an RP2040**
  - More details in [RP2040 Datasheet] (Dual cores)

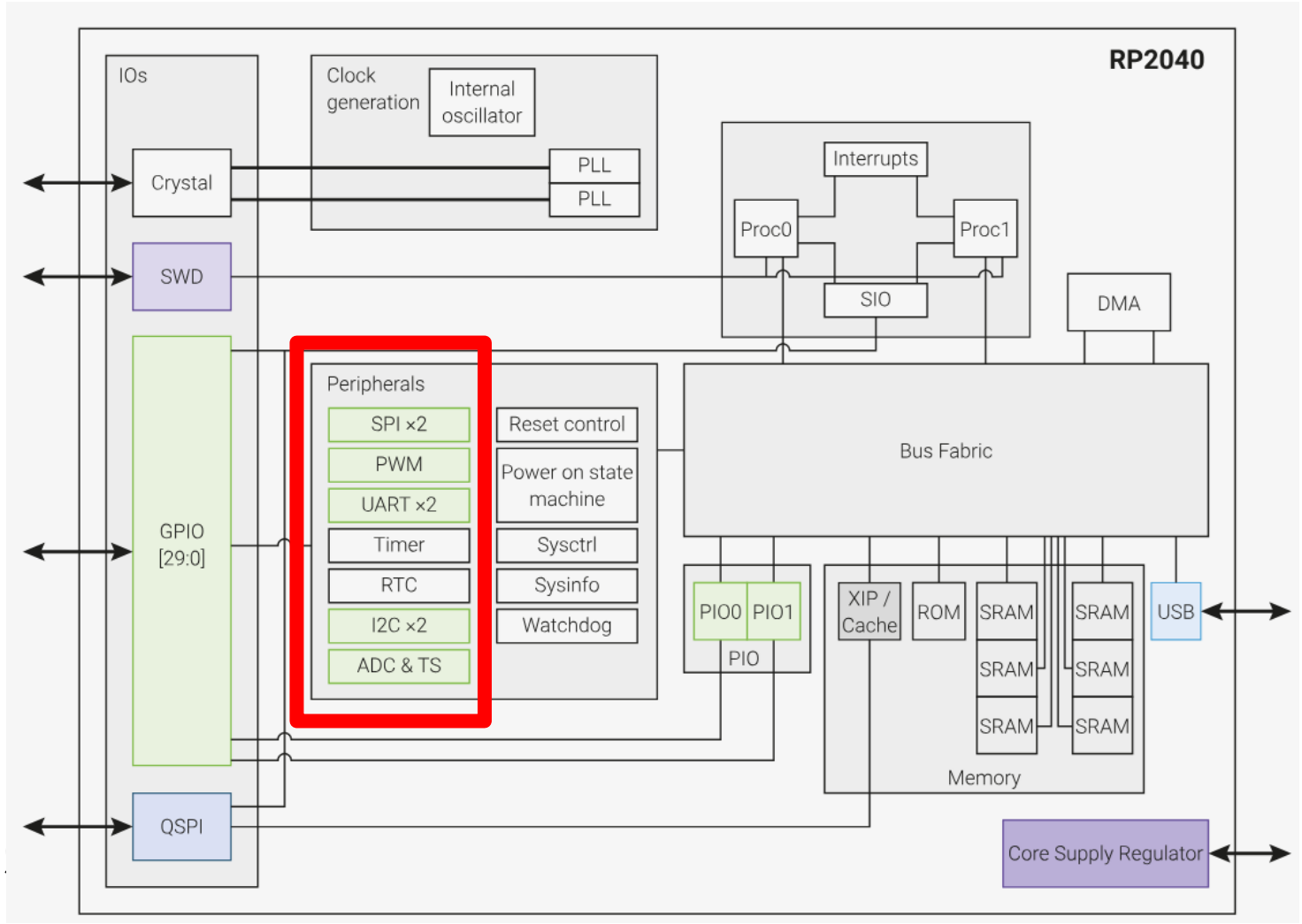


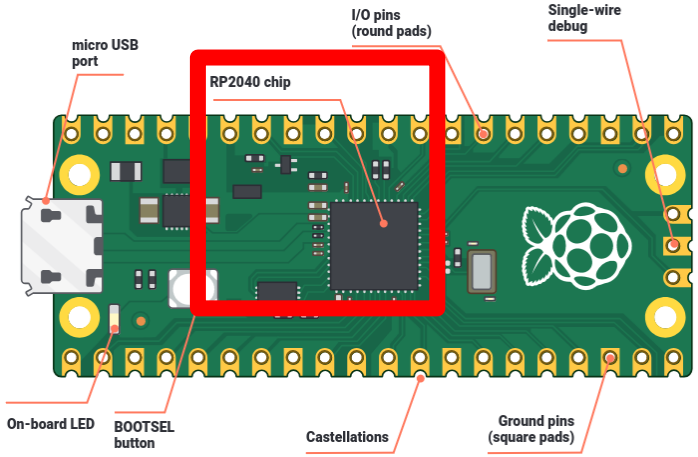


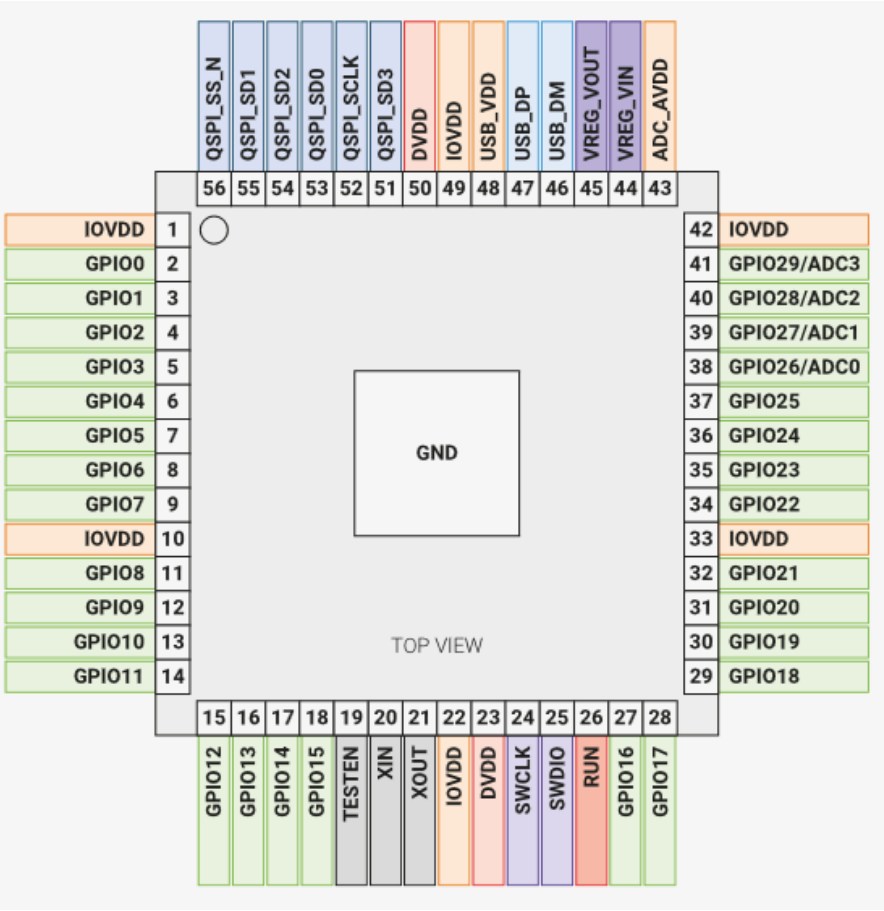


Raspberry Pi Pico and Pico W

- **Raspberry Pi Pico**'s is **based on an RP2040**
  - More details in [RP2040 Datasheet] (Communication peripherals)

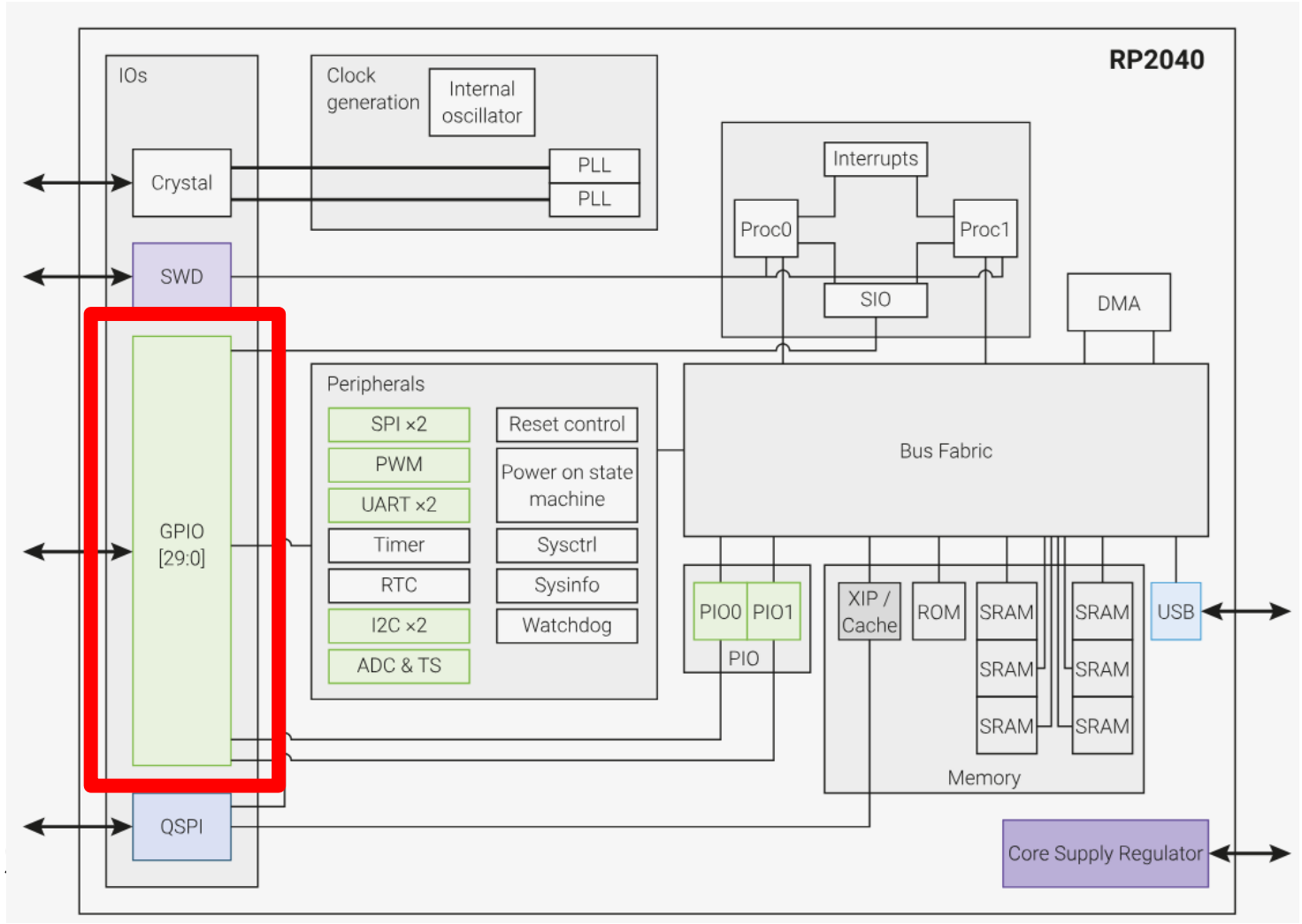


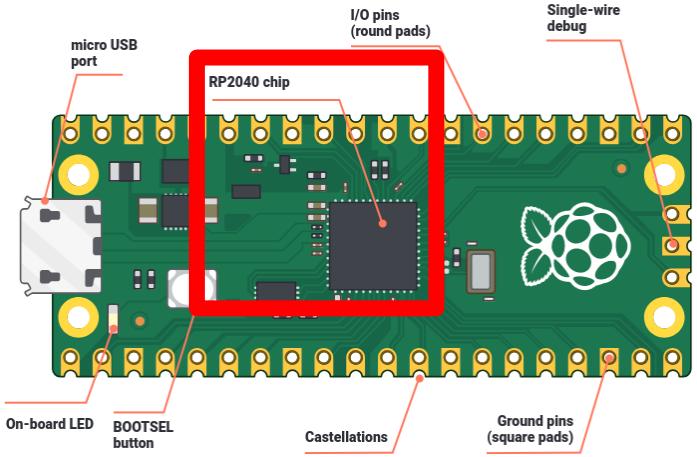


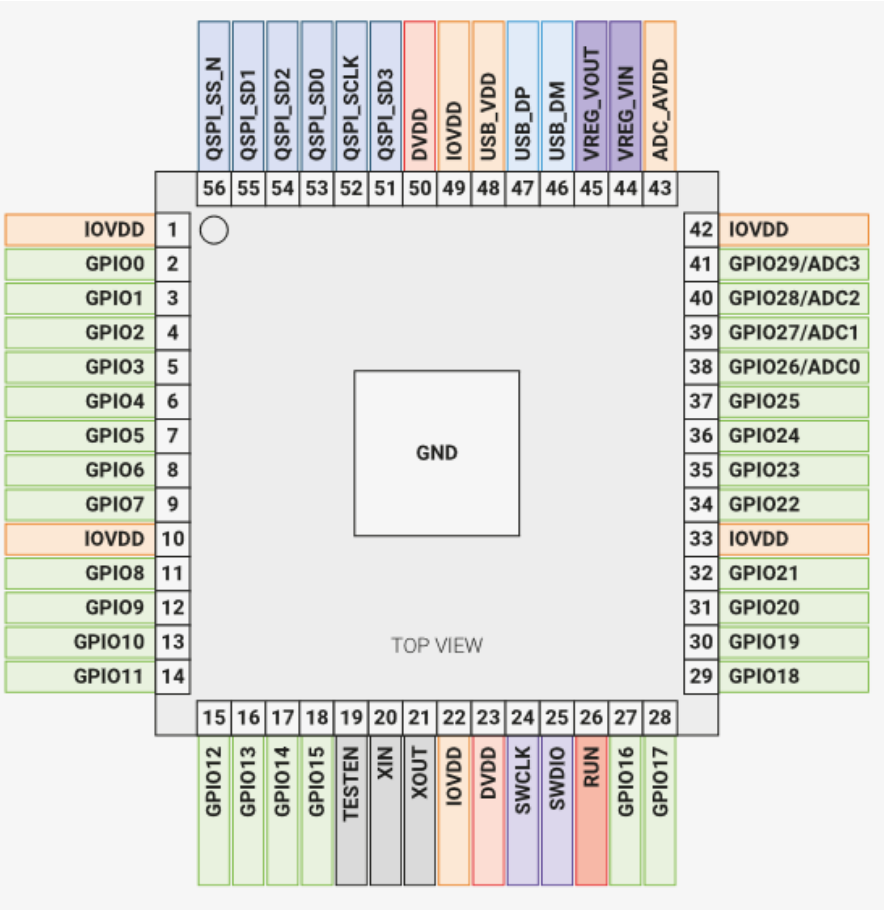


Raspberry Pi Pico and Pico W

- **Raspberry Pi Pico**'s is **based on an RP2040**
  - More details in [RP2040 Datasheet] (General-Purpose Input/Output (GPIO))

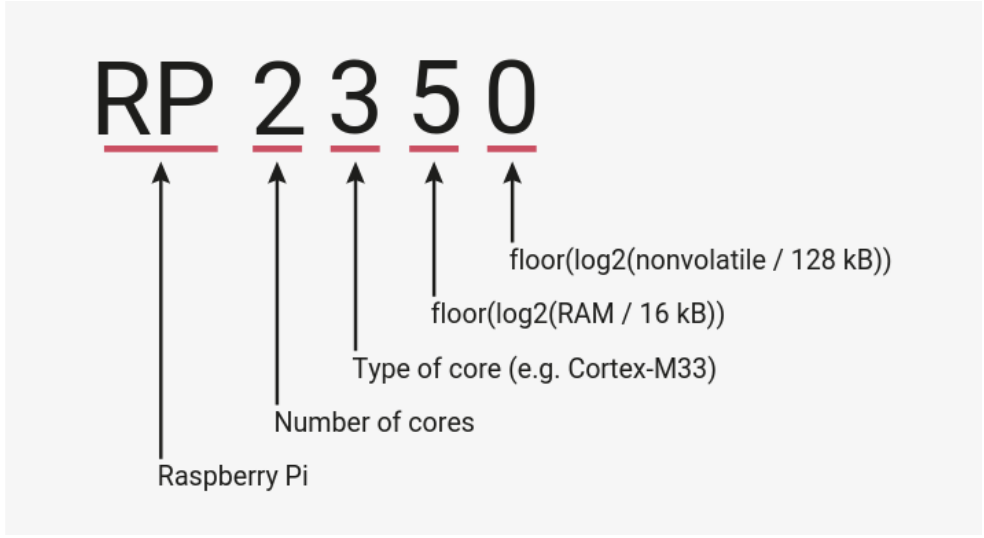


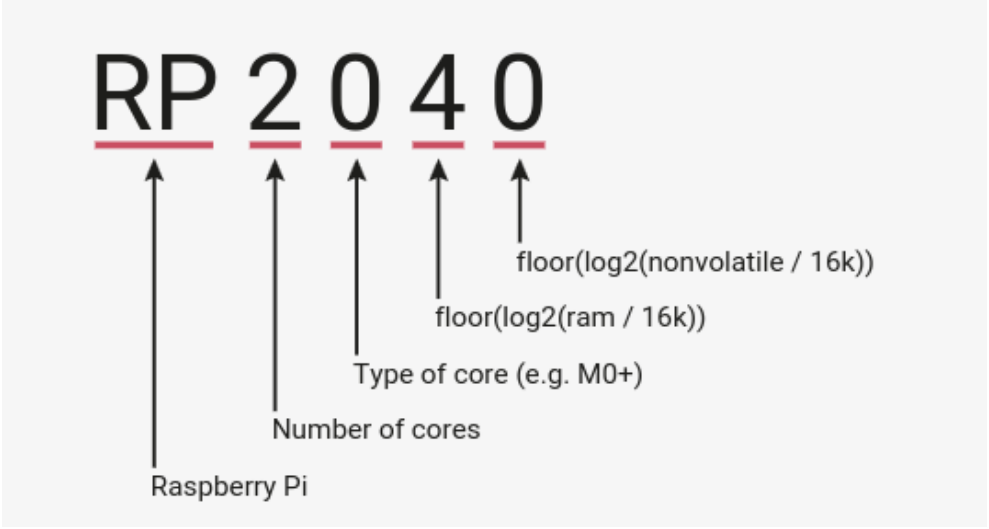




Raspberry Pi Pico and Pico W

- **Raspberry Pi Pico 2**'s is **based on an RP2350**
- **Raspberry Pi Pico**'s is **based on an RP2040**
  - **RP:** means 'Raspberry Pi'
  - **2:** is the number of processor cores the microcontroller has.
  - **0:** is the type of processor core, indicating in this case the RP2040 uses a processor core called the **Cortex-M0+** and the RP2350 uses a processor core called the **Cortex-M33**.
  - **4:** is how much random access memory (RAM) the microcontroller has, based on a special mathematical function: floor(log2(RAM/16)). In this case, '4' means the chip has 264 kilobytes (kB) of RAM.
    - Used to store your programs and the data they need.
  - **0:** is how much non-volatile storage the chip has, and is worked out in the same way as the RAM: floor(log2(NV/16)). In this case, 0 simply means there is no non-volatile storage on-board.





More details in [RP2040 Datasheet]

Raspberry Pi Pico and Pico W

- **Raspberry Pi Pico 2**'s is **based on an RP2350**
  - **Harvard** architecture[1]

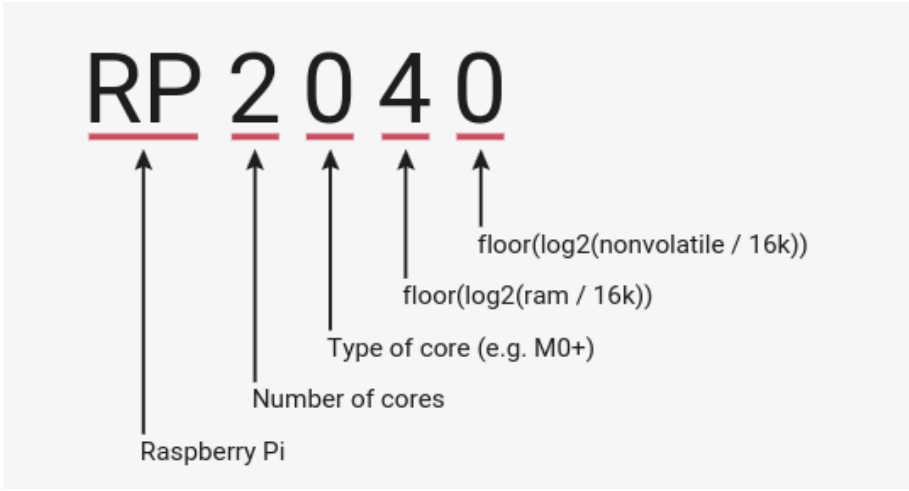
- **Raspberry Pi Pico**'s is **based on an RP2040**
  - **Von Neumann** architecture[1]

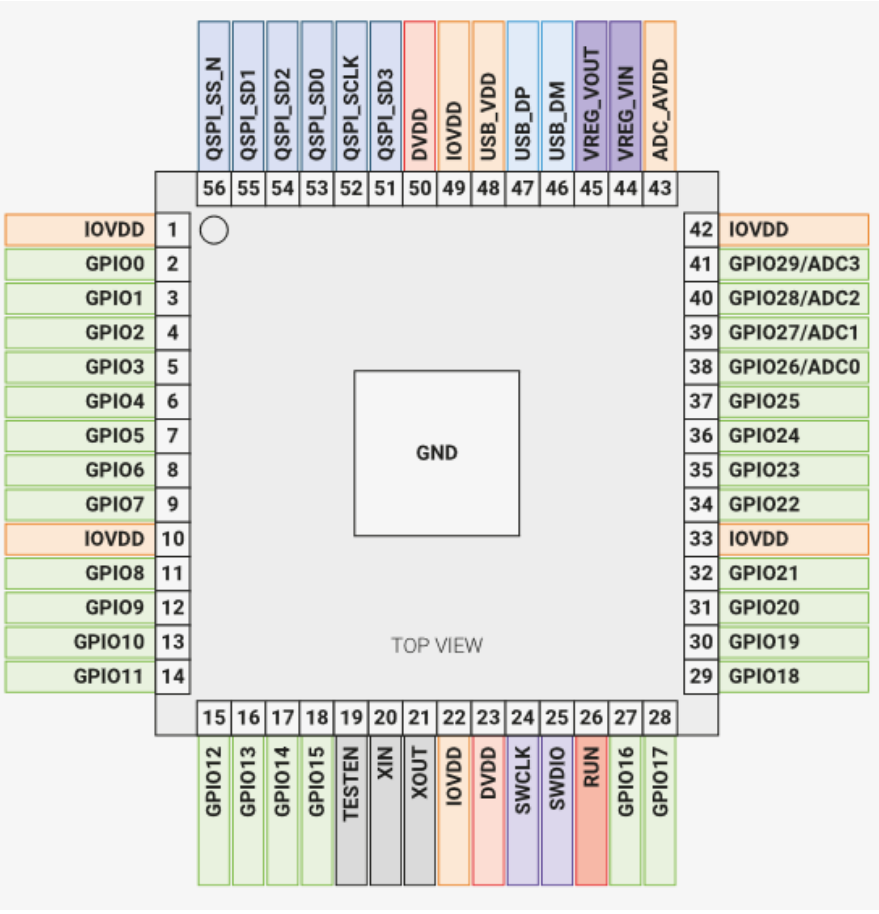
  - **Memory** (embedded ROM and RAM)
    - **ROM** (Read Only Memory)**:** A 16kB ROM is at address 0x00000000, containing: Initial startup routine, Flash boot sequence, etc.

    - **RAM** (Random Access Memory)**:** A 264kB Static RAM is at address 0x10000000
      - External Flash is accessed via the QSPI interface

        (**2 MB external flash** for the Pico / Pico W)

    - More details in RP2040 Datasheet, 2.6. Memory

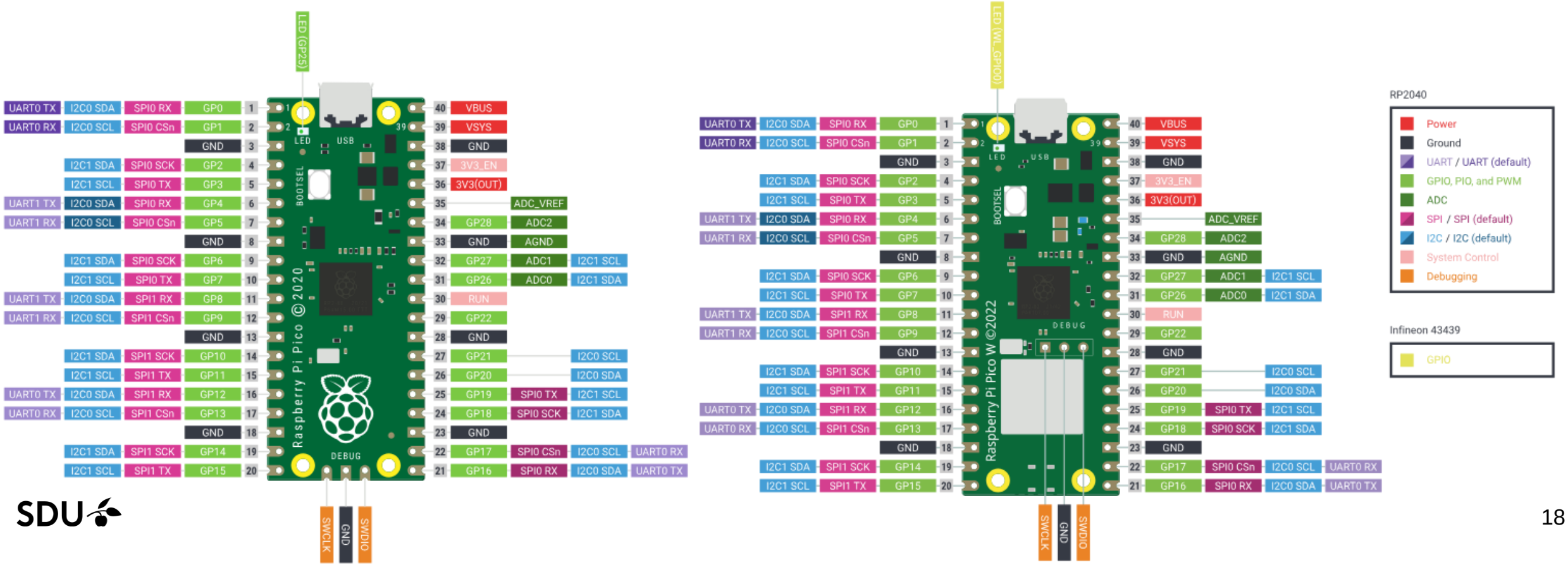




[1] https://en.wikipedia.org/wiki/ARM_Cortex-M

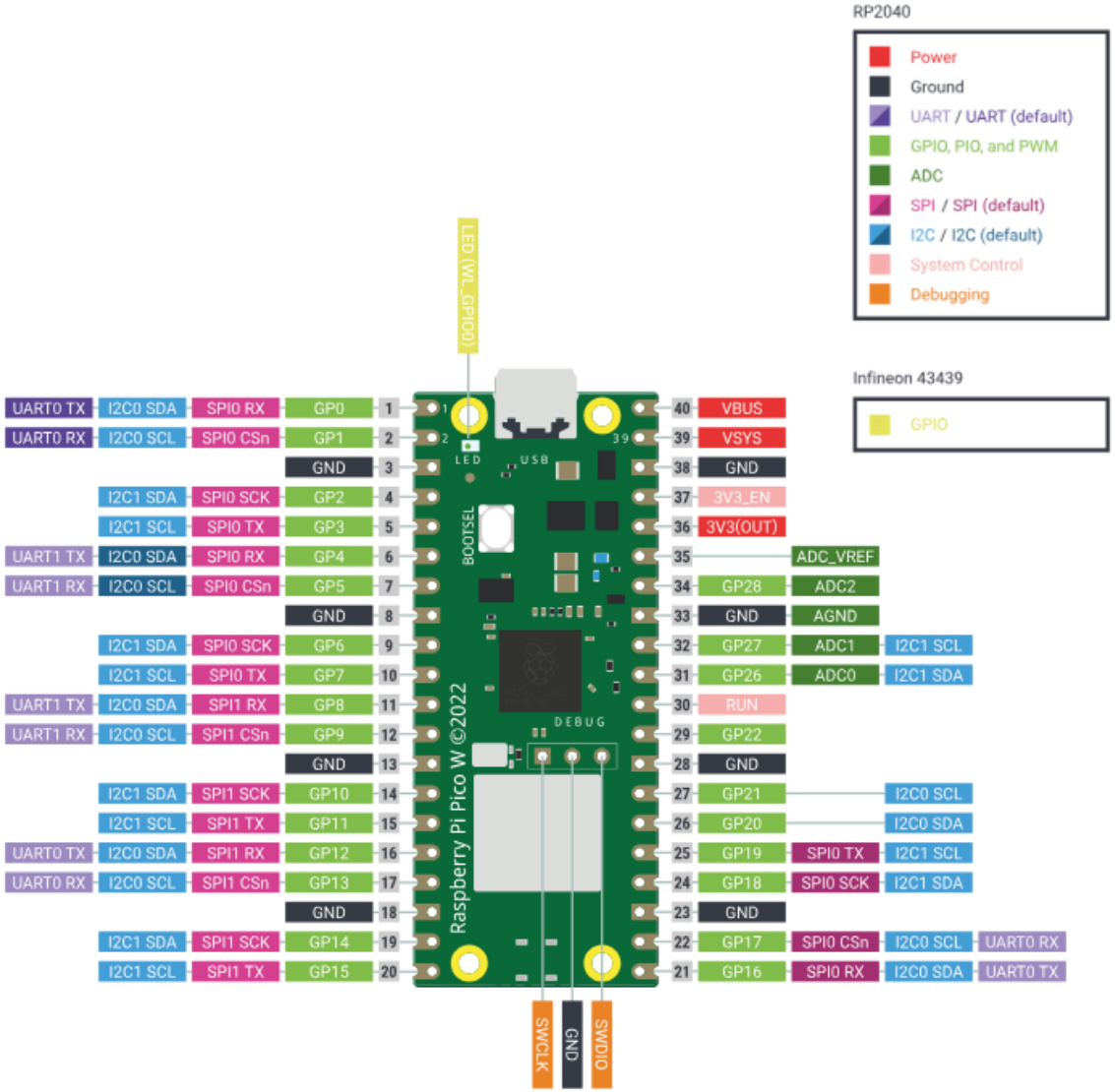
Raspberry Pi Pico and Pico W

- Pinout
  - Pico: https://pico.pinout.xyz/
    - or https://www.raspberrypi.com/documentation/microcontrollers/images/pico-pinout.svg
  - Pico W: https://picow.pinout.xyz/
    - or https://www.raspberrypi.com/documentation/microcontrollers/images/picow-pinout.svg



Raspberry Pi Pico and Pico W

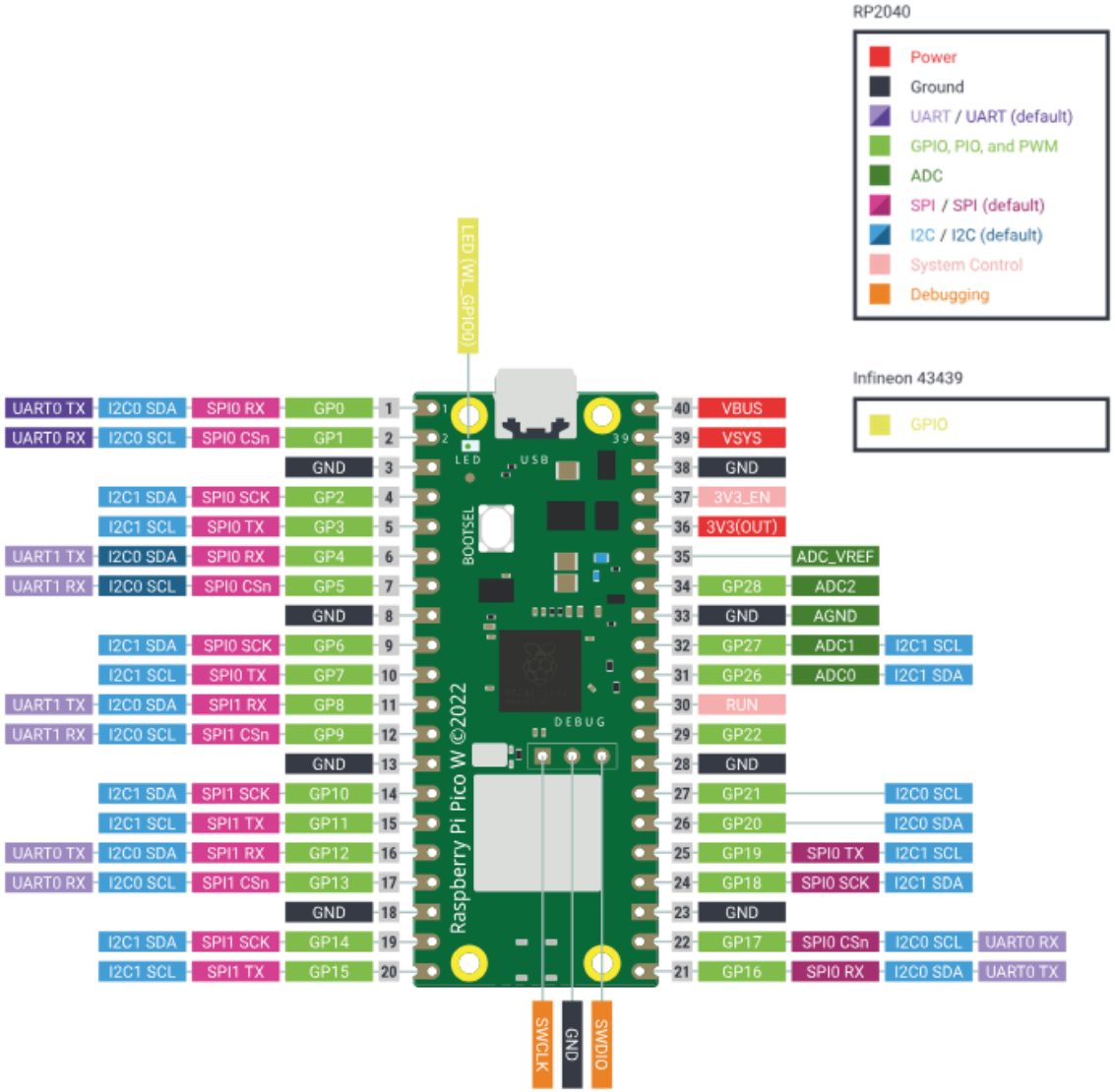
- **Overall specifications**



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

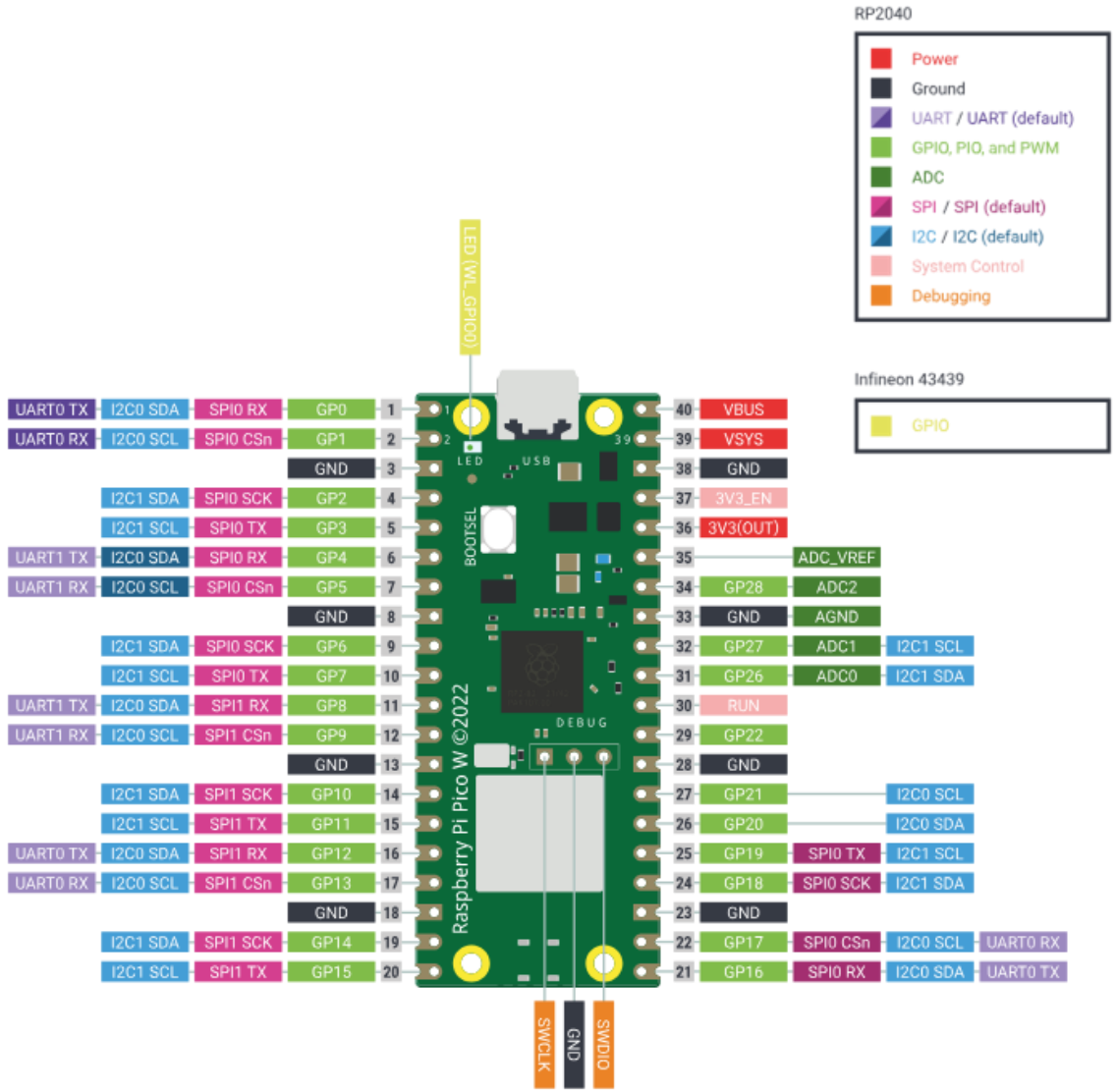
- **Overall specifications**
  - **CPU:** 32-bit ***dual-core*** ARM Cortex-M0+ at 48MHz, configurable up to 133MHz



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

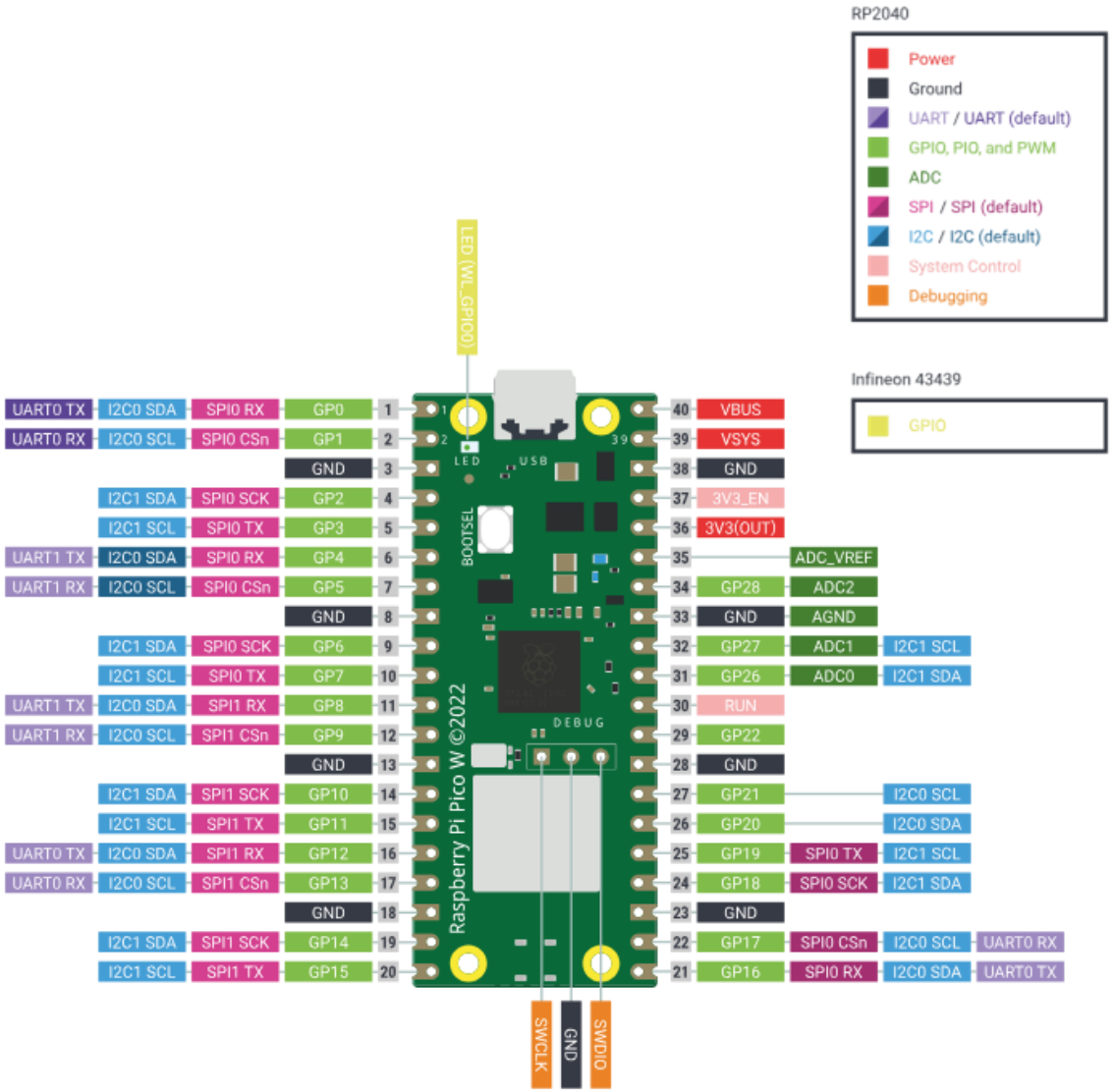
- **Overall specifications**
  - **CPU:** 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
  - **General-Purpose Input/Output (GPIO):** 26 pins



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

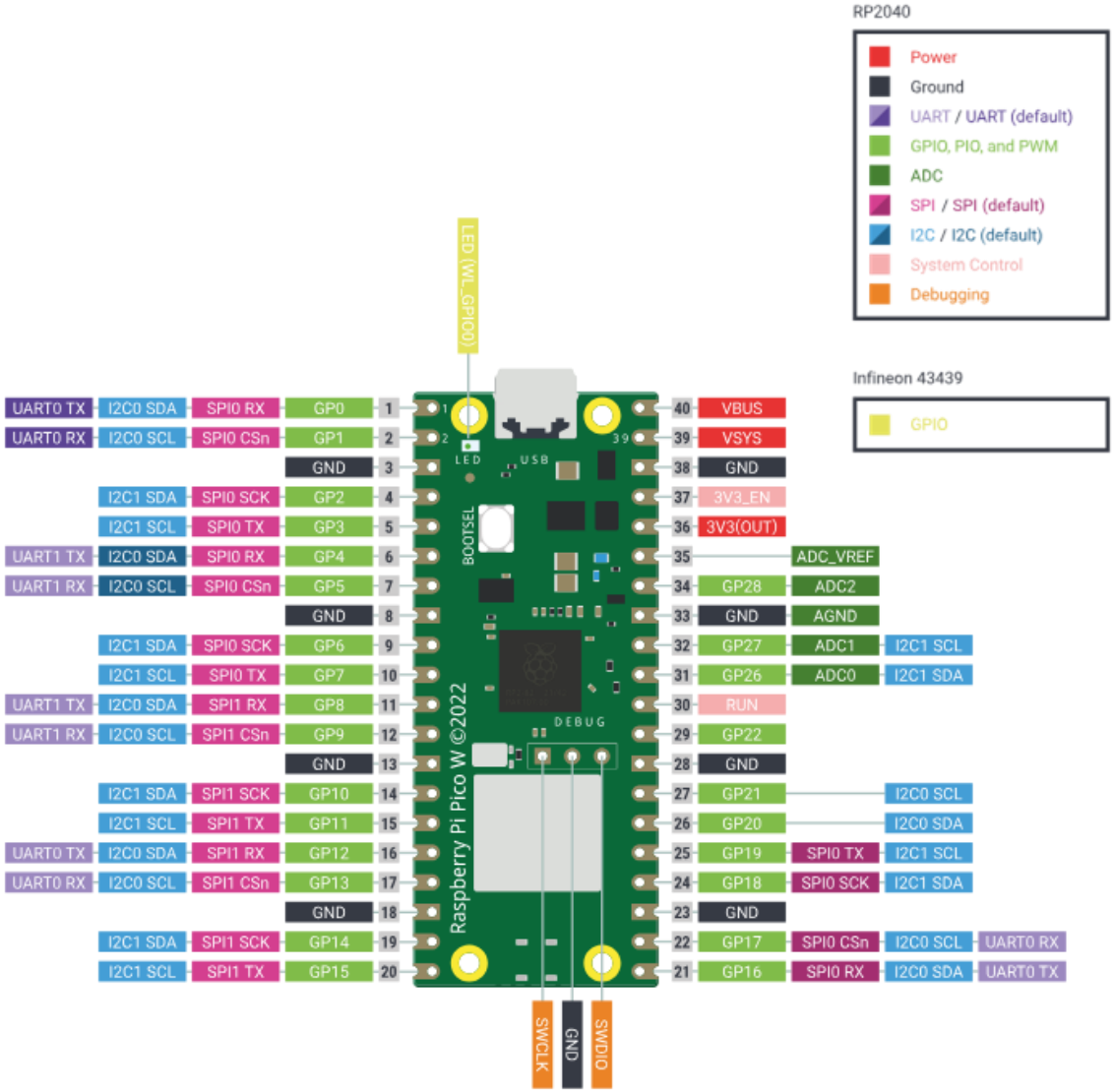
- **GPIO Pins:** 26 multi-functional GPIO pins, labeled GP0 to GP28. These pins can be used for various digital input/output tasks and **support multiple interfaces and protocols**.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

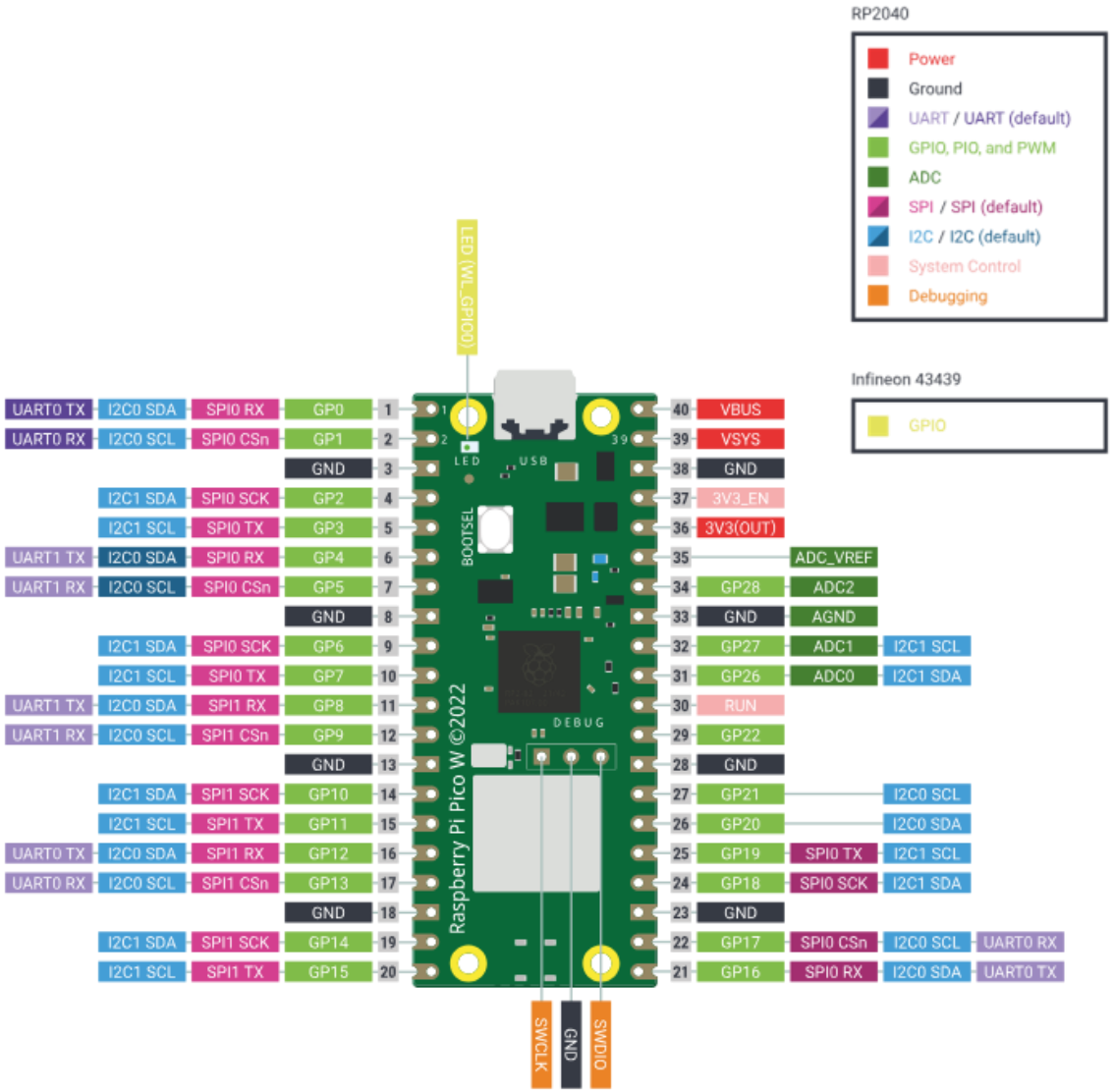
- **GPIO Pins:** 26 multi-functional GPIO pins, labeled GP0 to GP28. These pins can be used for various digital input/output tasks and **support multiple interfaces and protocols**.
    - **Digital I/O:** Each GPIO pin can be configured as either an input or output (High / Low).



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

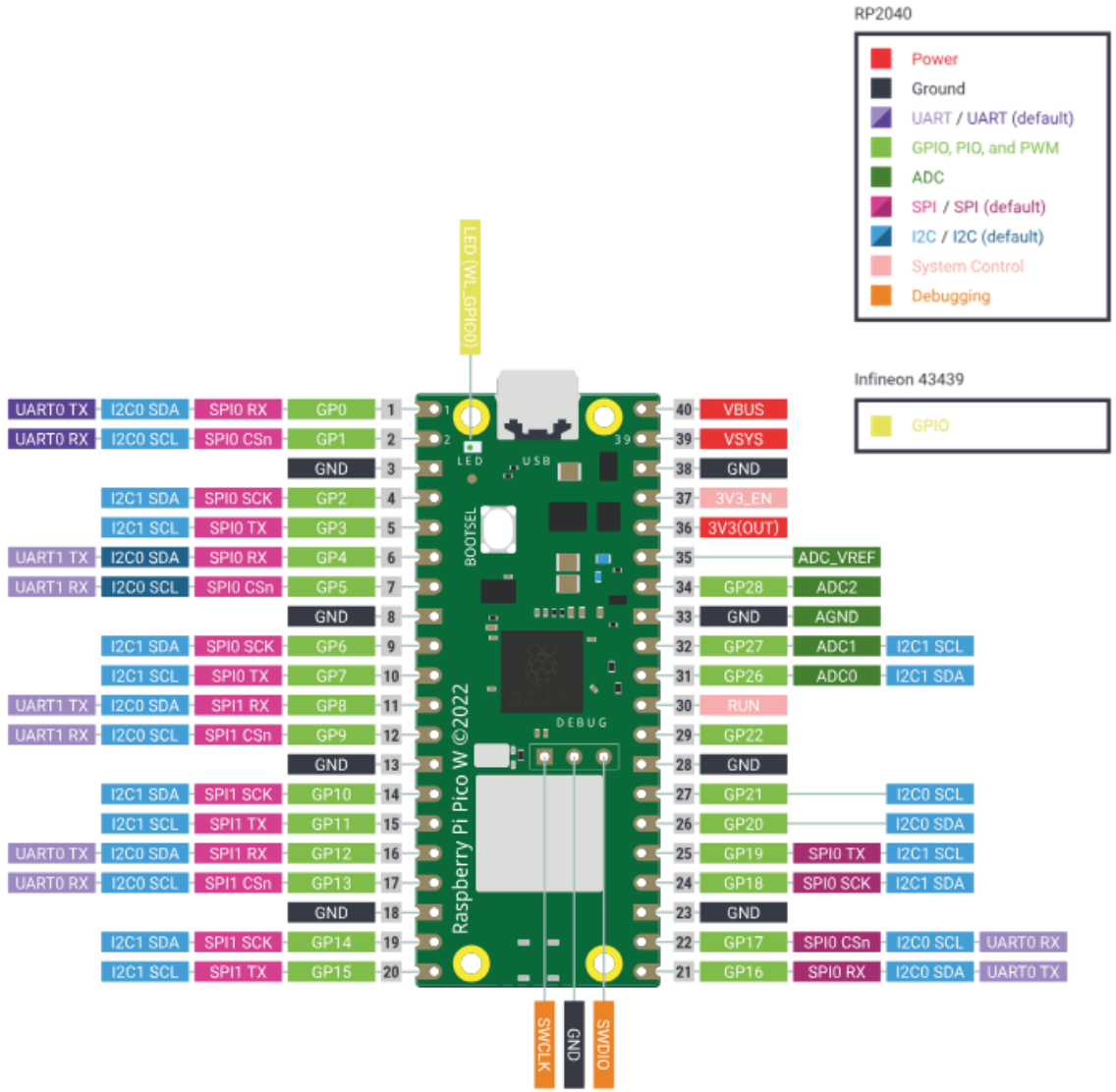
- **GPIO Pins:** 26 multi-functional GPIO pins, labeled GP0 to GP28. These pins can be used for various digital input/output tasks and **support multiple interfaces and protocols**.
  - **Digital I/O:** Each GPIO pin can be configured as either an input or output (High / Low).
  - **Analog Input:** There are three ADC (Analog-to-Digital Converter) channels available on GP26, GP27, and GP28, which can read analog voltage levels, eg. from sensors.
    - **…more about ADC in module 6!**



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

- **GPIO Pins:** 26 multi-functional GPIO pins, labeled GP0 to GP28. These pins can be used for various digital input/output tasks and **support multiple interfaces and protocols**.
  - **Digital I/O:** Each GPIO pin can be configured as either an input or output (High / Low).
  - **Analog Input:** There are three ADC (Analog-to-Digital Converter) channels available on GP26, GP27, and GP28, which can read analog voltage levels, eg. from sensors.
  - **Pulse-width modulation (PWM):** PWM can be generated on most GPIO pins, useful for controlling motors, LEDs, and other devices requiring varying signal strength.
    - **…more about PWM in module 4!**



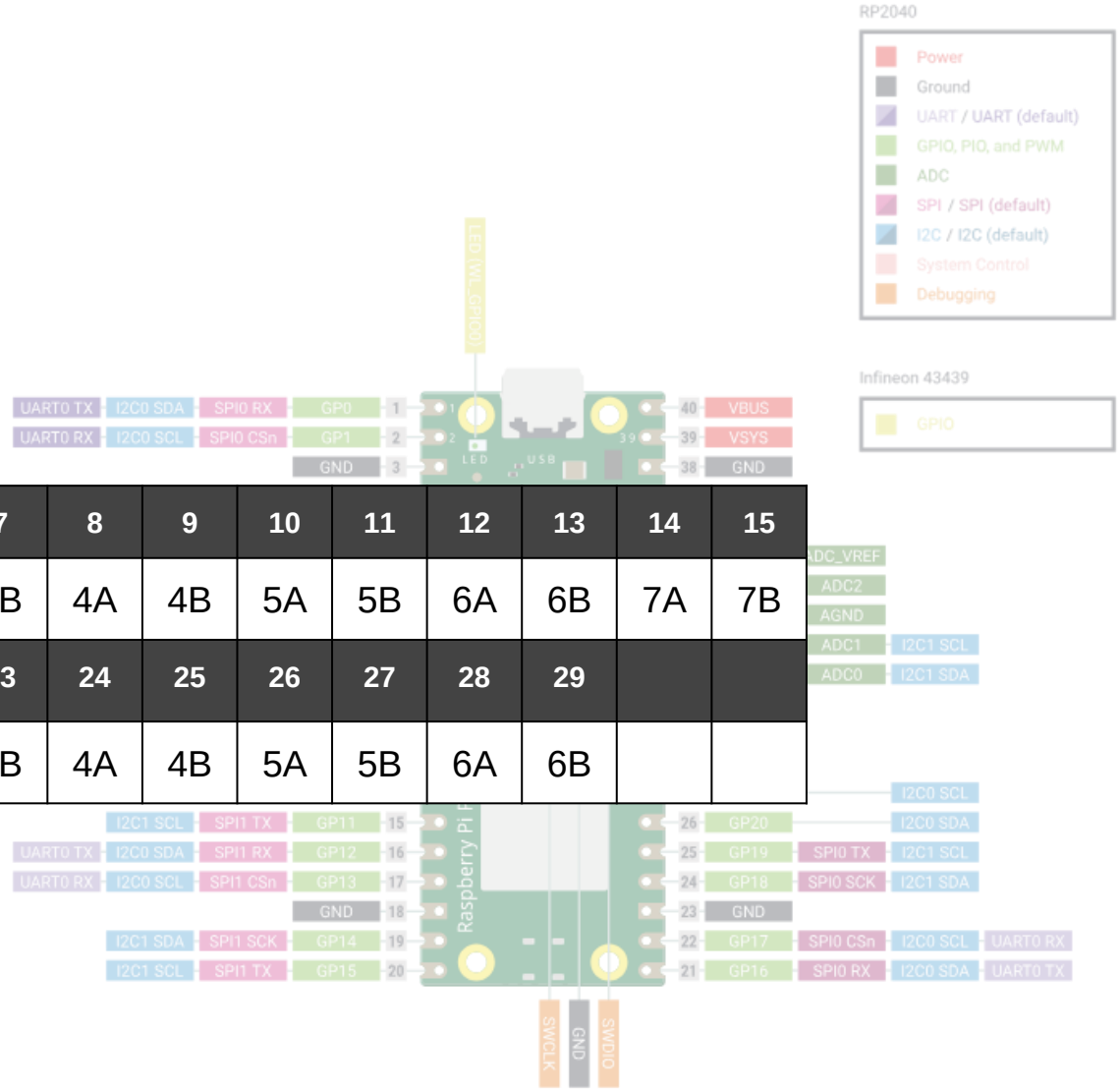
Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

- **GPIO Pins:** 26 multi-functional GPIO pins, labeled GP0 to GP28. These pins can be used for various digital input/output tasks and support multiple interfaces and protocols.
  - **Digital I/O:** Each GPIO pin can be configured as either an input or output (High / Low).
  - **An**
    Co
    GP
    se

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B		

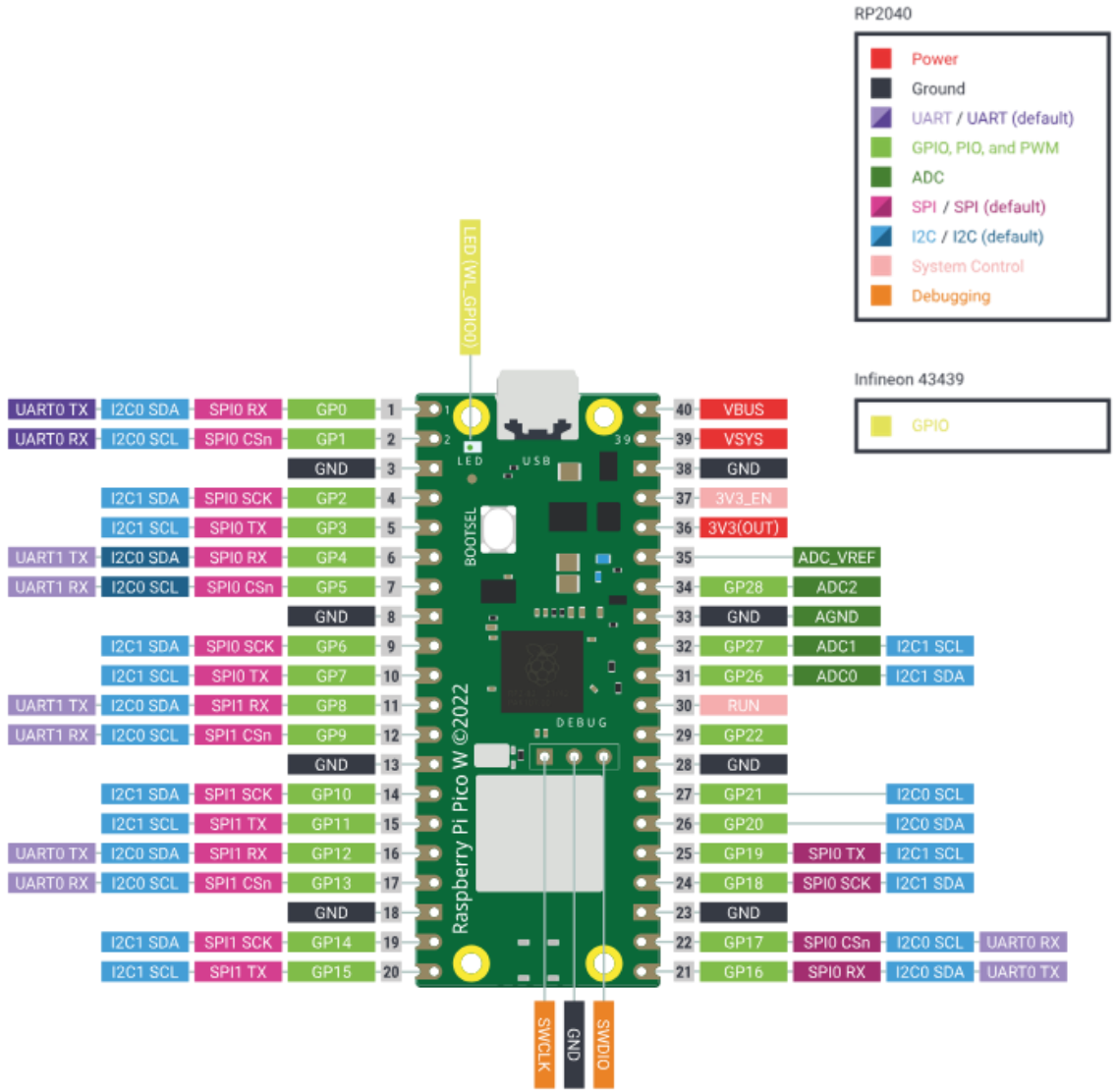
  - **Pulse-width modulation (PWM):** PWM can be generated on most GPIO pins, useful for controlling motors, LEDs, and other devices requiring varying signal strength.
    - **8 independent PWM generators** called slices. Each slice has two channels (A and B), which makes a total of 16 PWM channels.
      - More details in RP2040 Datasheet



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

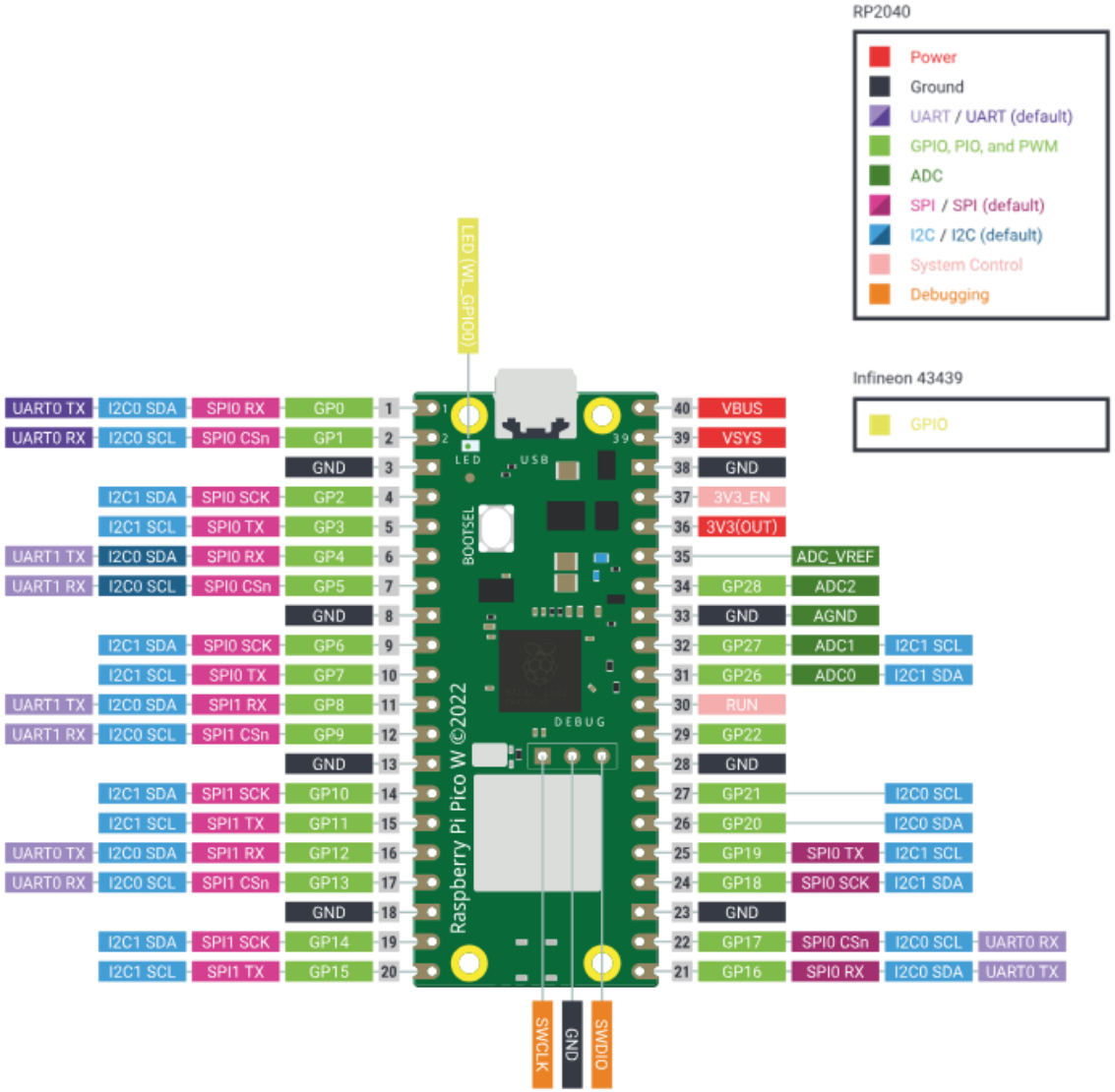
- **GPIO Pins:** 26 multi-functional GPIO pins, labeled GP0 to GP28. These pins can be used for various digital input/output tasks and support multiple interfaces and protocols.
  - **Digital I/O:** Each GPIO pin can be configured as either an input or output (High / Low).
  - **Analog Input:** There are three ADC (Analog-to-Digital Converter) channels available on GP26, GP27, and GP28, which can read analog voltage levels, eg. from sensors.
  - **Pulse-width modulation (PWM):** PWM can be generated on most GPIO pins, useful for controlling motors, LEDs, and other devices requiring varying signal strength.
  - **Communication Protocols (Pins):** Several GPIO pins can function as interfaces for I2C, SPI, and UART, enabling communication with other devices and sensors.
    - **…more about that in Module 6 and Module 9)**



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

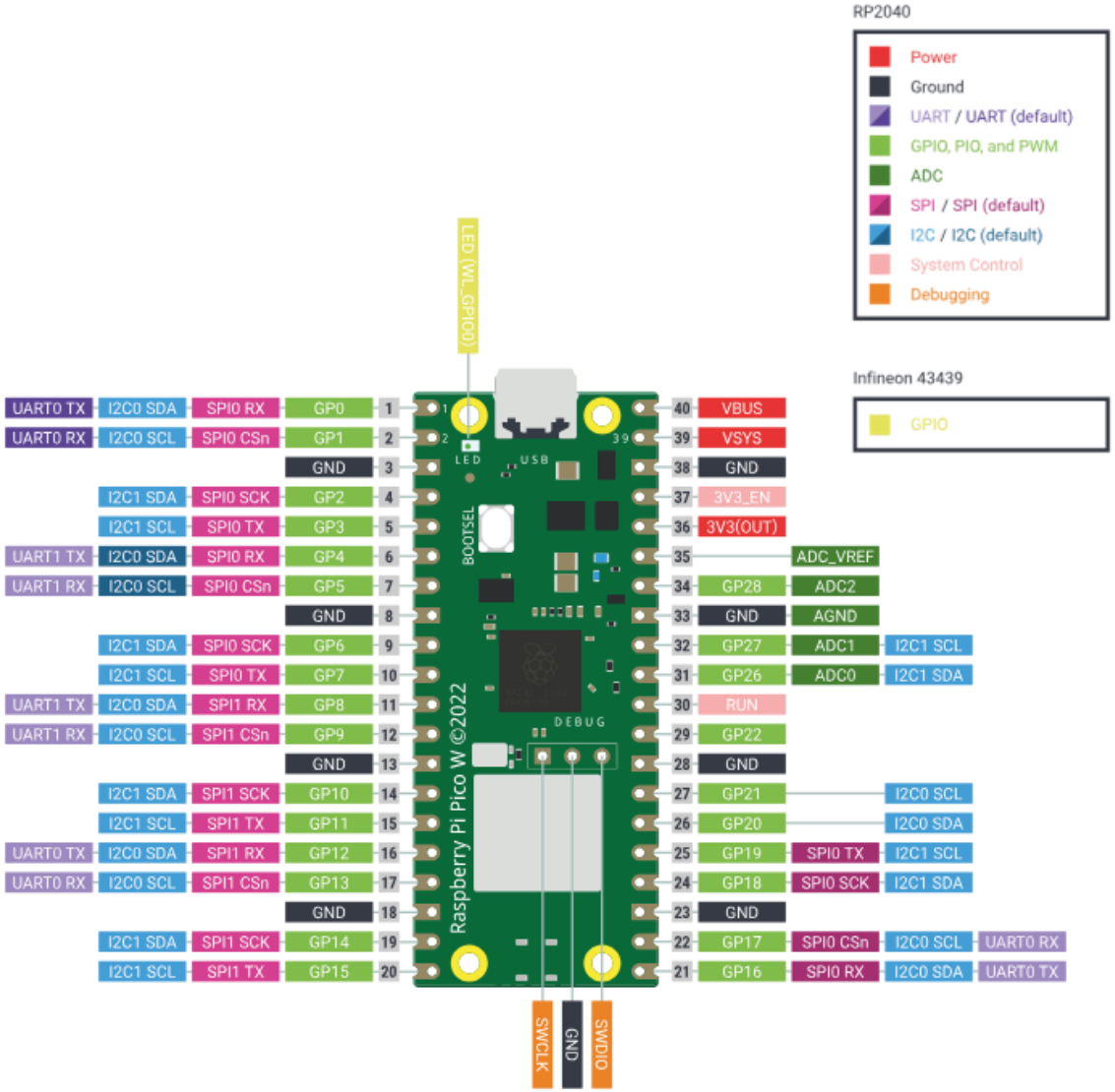
- **Communication Protocols (Pins)**
  - **I2C Pins** (Two I2C peripherals: I2C1 and I2C0)
    - **SDA (Data Line)** and **SCL (Clock Line)**: These pins are typically used for I2C communication. They are multiplexed on various GPIO pins.
  - **SPI Pins** (Two SPI peripherals: SPI0 and SPI1)
    - **MOSI/TX, MISO/RX, SCK,** and **SS/CS**: These pins are used for SPI communication and are multiplexed on various GPIO pins.
  - **UART Pins** (Two UART peripherals: UART0 and UART1)
    - **TX (Transmit)** and **RX (Receive)**: Used for UART communication.
  - …more about that in **Module 6** and **Module 9**.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

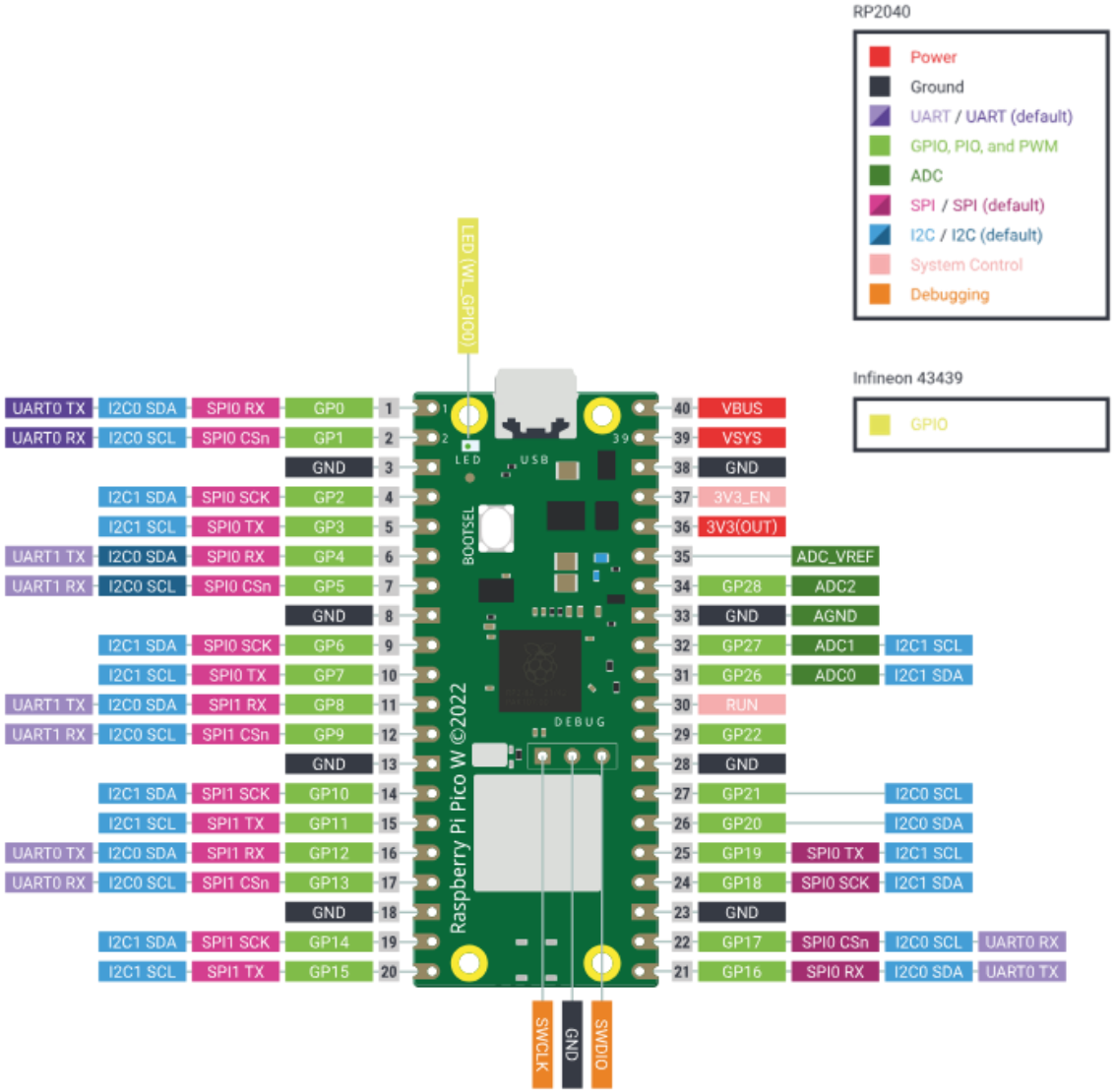
- **Overall specifications**
  - **CPU:** 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
  - **GPIO:** 26 pins
    - **ADC:** 3 × 12-bit ADC pins
    - **PWM:** 8x slices, two outputs per slice for 16 total
    - **Communication Protocols:** 2 × UART, 2 × SPI, 2 × I2C



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

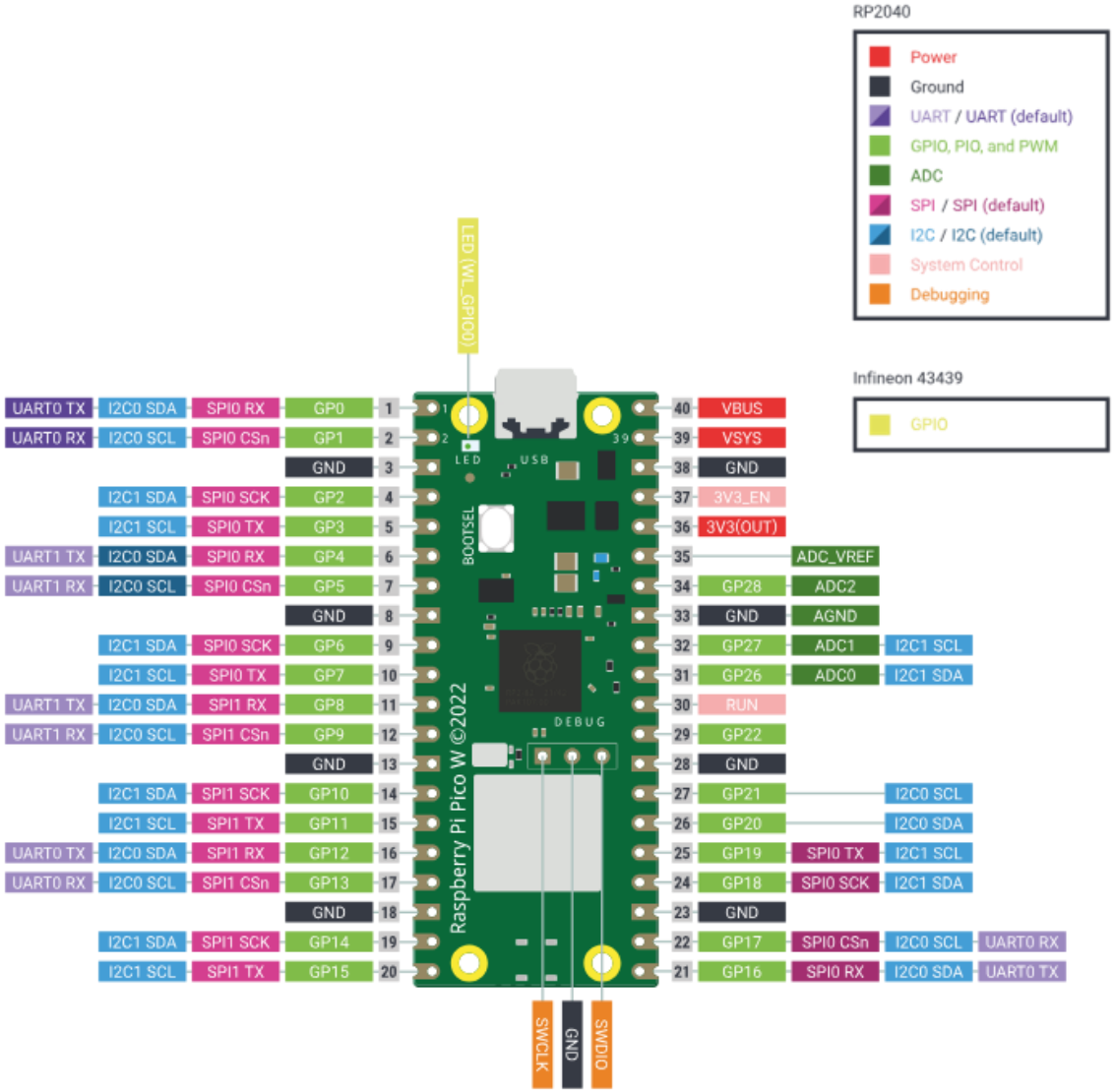
- **Overall specifications**
  - **CPU:** 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
  - **GPIO:** 26 pins
    - **ADC:** 3 × 12-bit ADC pins
    - **PWM:** 8x slices, two outputs per slice for 16 total
    - **Communication Protocols:** 2 × UART, 2 × SPI, 2 × I2C
  - **Clock:** on-chip clock and timer



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

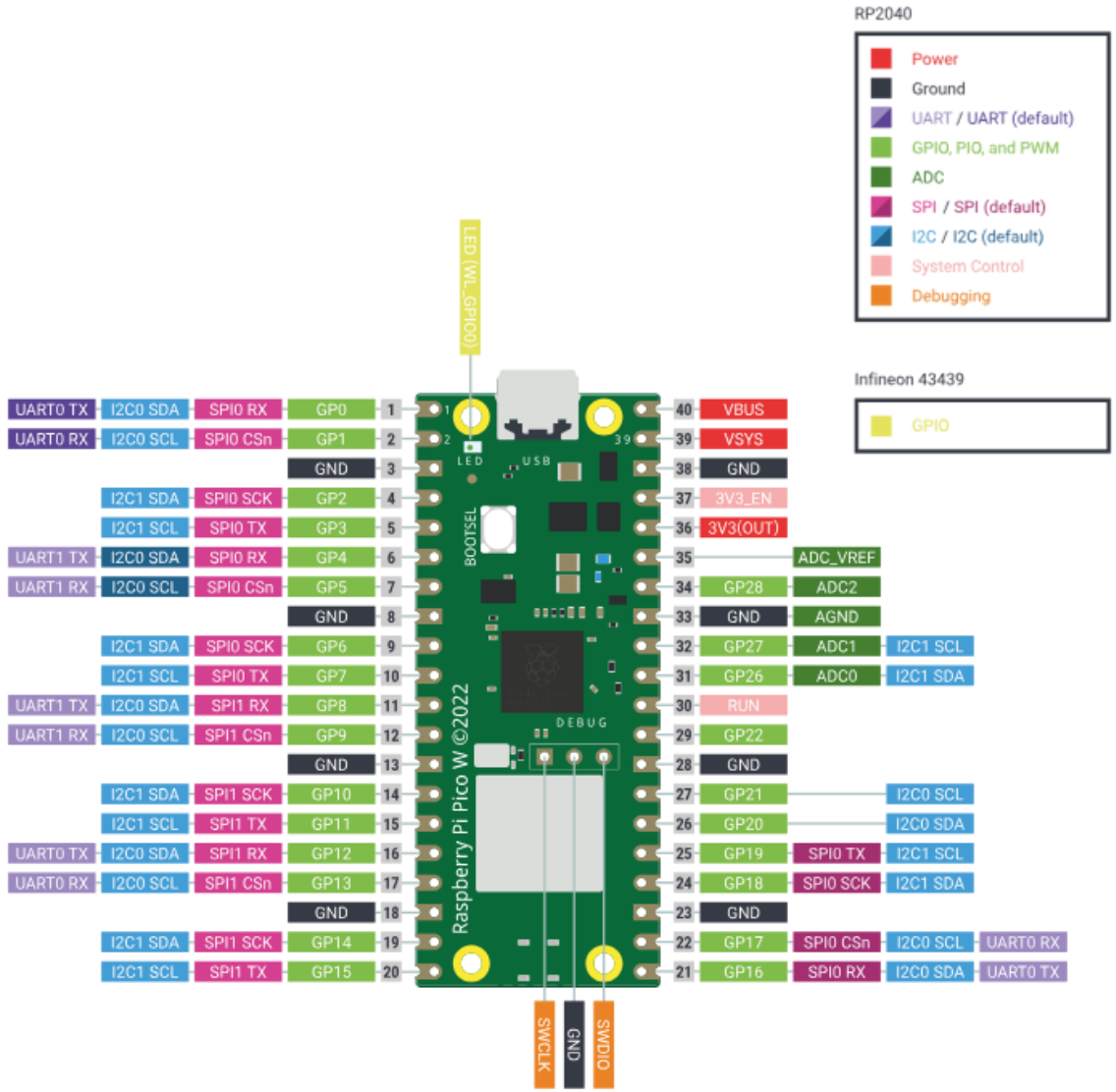
- **Overall specifications**
  - **CPU:** 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
  - **GPIO:** 26 pins
    - **ADC:** 3 × 12-bit ADC pins
    - **PWM:** 8x slices, two outputs per slice for 16 total
    - **Communication Protocols:** 2 × UART, 2 × SPI, 2 × I2C
  - **Clock:** on-chip clock and timer
  - **Sensors:** On-chip temperature sensor connected to 12-bit ADC channel)



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

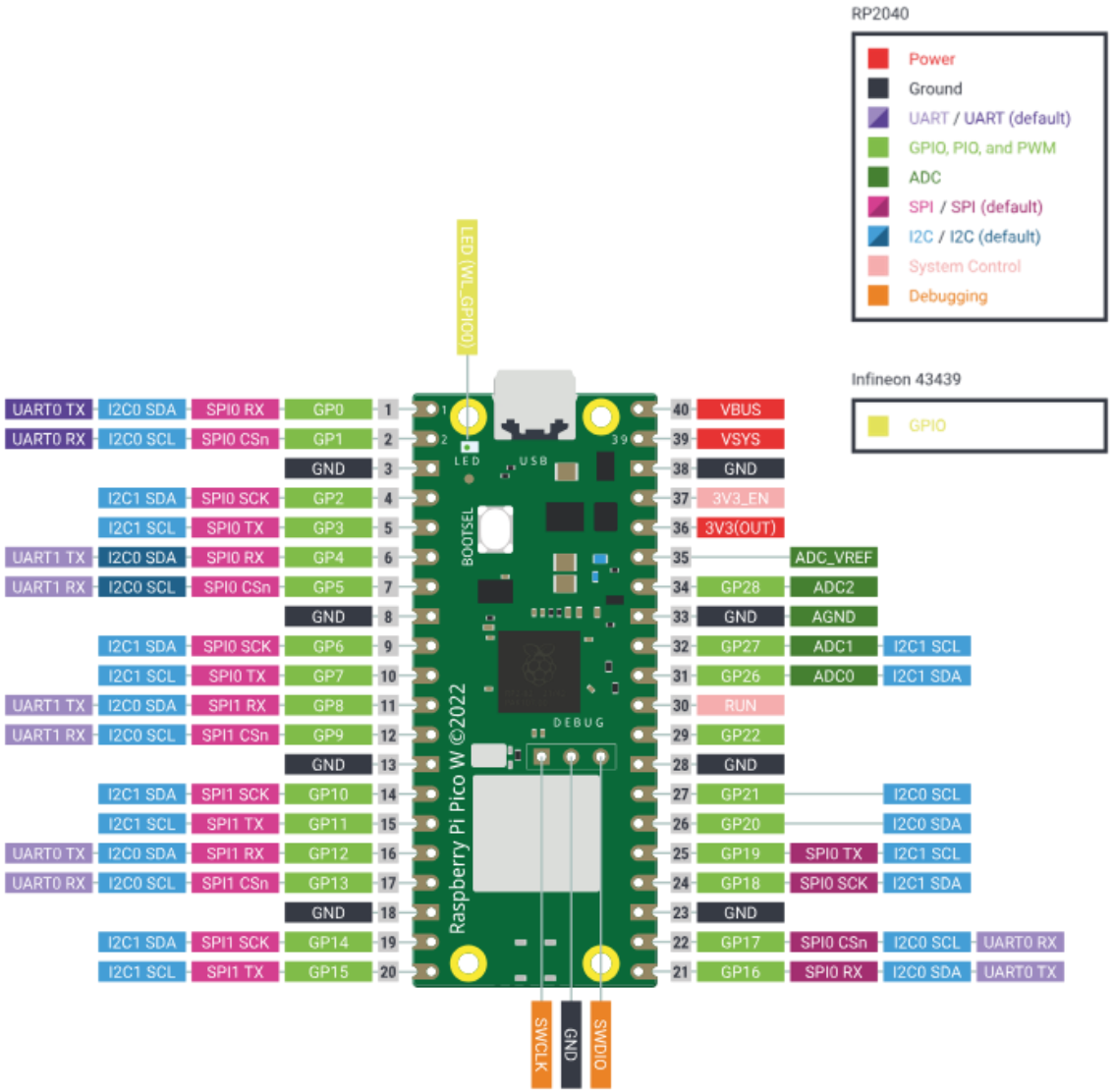
- **Overall specifications**
  - **CPU:** 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
  - **GPIO:** 26 pins
    - **ADC:** 3 × 12-bit ADC pins
    - **PWM:** 8x slices, two outputs per slice for 16 total
    - **Communication Protocols:** 2 × UART, 2 × SPI, 2 × I2C
  - **Clock:** on-chip clock and timer
  - **Sensors:** On-chip temperature sensor connected to 12-bit ADC channel)
  - **LEDs:** On-board user-addressable LED



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

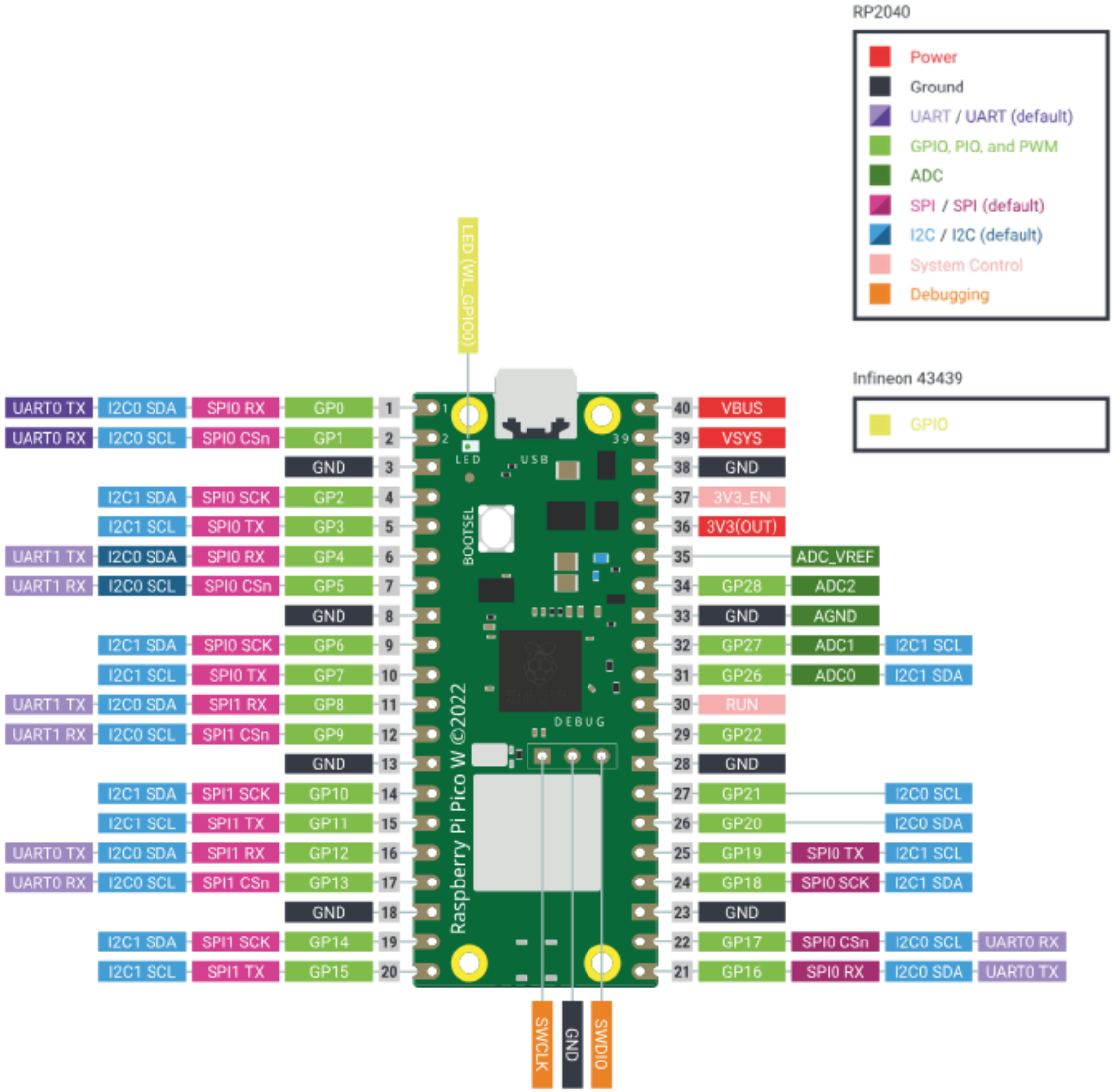
- **Overall specifications**
  - **CPU:** 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
  - **GPIO:** 26 pins
    - **ADC:** 3 × 12-bit ADC pins
    - **PWM:** 8x slices, two outputs per slice for 16 total
    - **Communication Protocols:** 2 × UART, 2 × SPI, 2 × I2C
  - **Clock:** on-chip clock and timer
  - **Sensors:** On-chip temperature sensor connected to 12-bit ADC channel)
  - **LEDs:** On-board user-addressable LED
  - **Hardware Debug:** Single-Wire Debug (SWD)



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

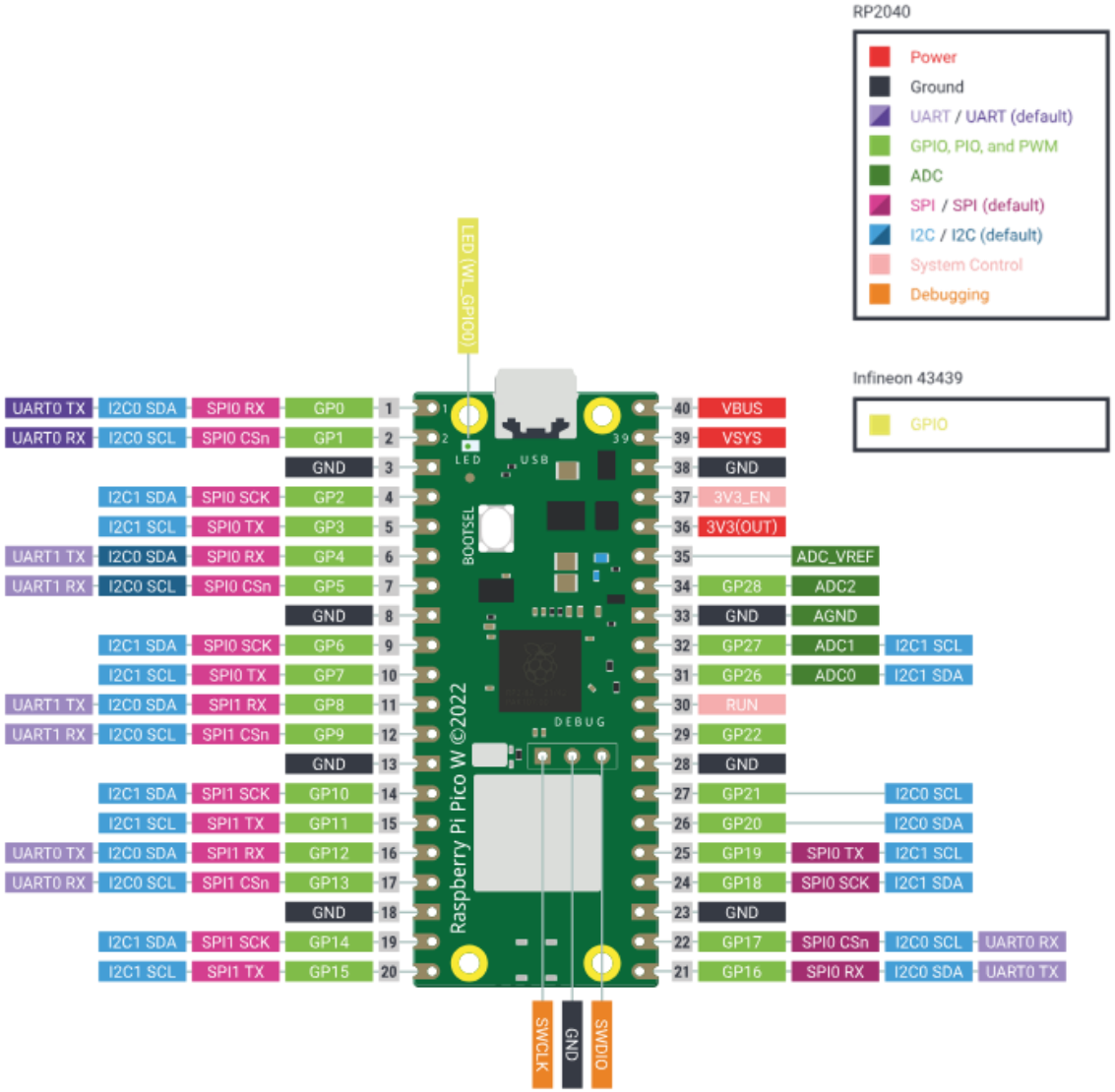
- **Overall specifications**
  - **CPU:** 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
  - **GPIO:** 26 pins
    - **ADC:** 3 × 12-bit ADC pins
    - **PWM:** 8x slices, two outputs per slice for 16 total
    - **Communication Protocols:** 2 × UART, 2 × SPI, 2 × I2C
  - **Clock:** on-chip clock and timer
  - **Sensors:** On-chip temperature sensor connected to 12-bit ADC channel)
  - **LEDs:** On-board user-addressable LED
  - **Hardware Debug:** Single-Wire Debug (SWD)
  - **Power:** 5 V via micro USB connector, 3.3 V via 3V3 pin, or 2–5V via VSYS pin



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

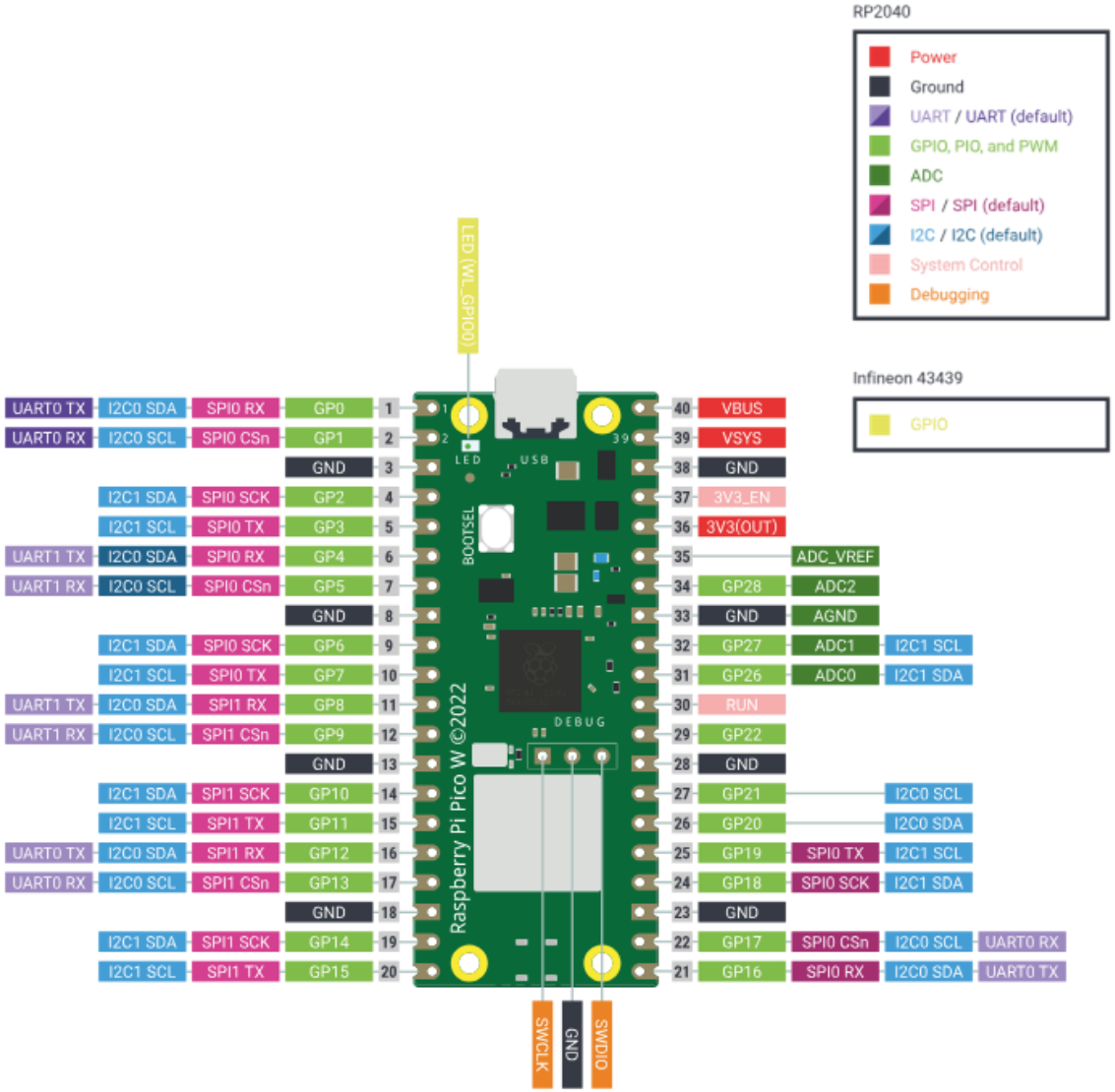
- **Power Pins:** These pins provide various voltage levels to power the Raspberry Pi Pico and connected peripherals.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

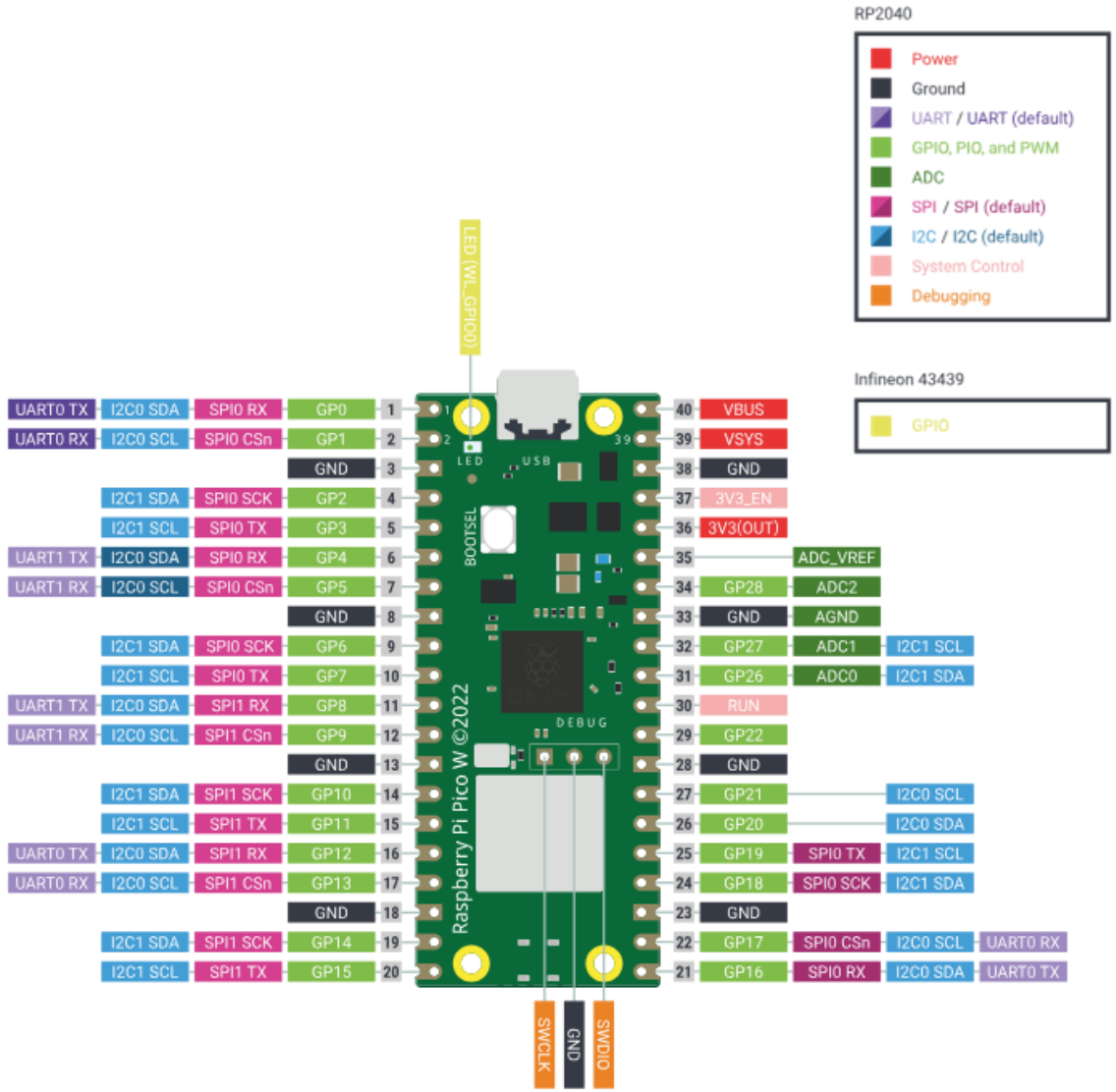
- **Power Pins:** These pins provide various voltage levels to power the Raspberry Pi Pico and connected peripherals.
  - **VSYS (Pin 39):** This is the main input power pin for the Pico, which can accept a range of voltages from 1.8V to 5.5V. It powers the onboard 3.3V regulator.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

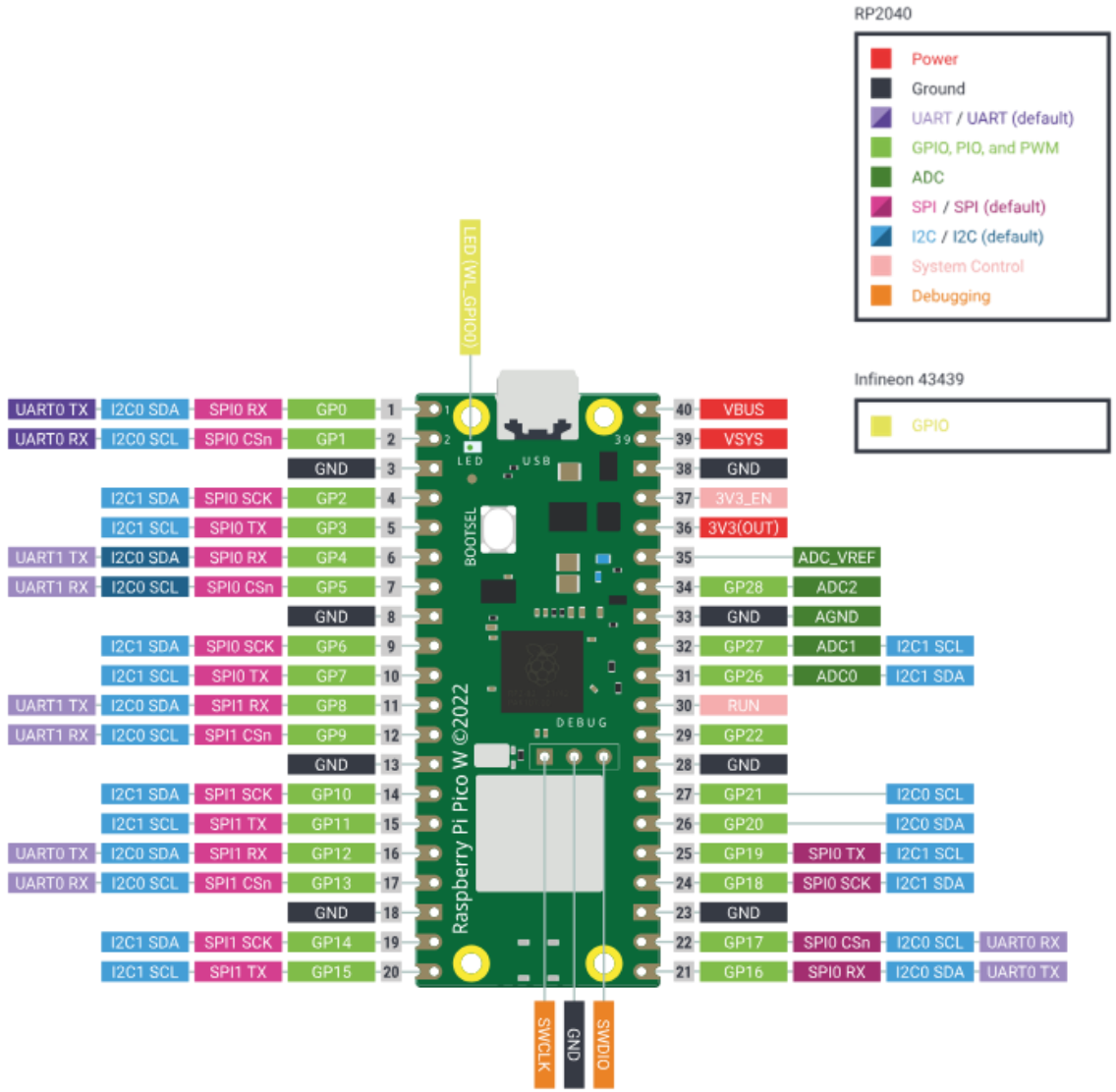
- **Power Pins:** These pins provide various voltage levels to power the Raspberry Pi Pico and connected peripherals.
  - **VSYS (Pin 39):** This is the main input power pin for the Pico, which can accept a range of voltages from 1.8V to 5.5V. It powers the onboard 3.3V regulator.
  - **3V3 (Pins 36, 37):** Provides 3.3V output, which is the regulated voltage supplied by the Pico for powering the microcontroller and any connected 3.3V devices.
  - **3V3_EN (Pin 37):** An enable pin for the 3.3V regulator. Pulling this pin low disables the 3.3V output.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

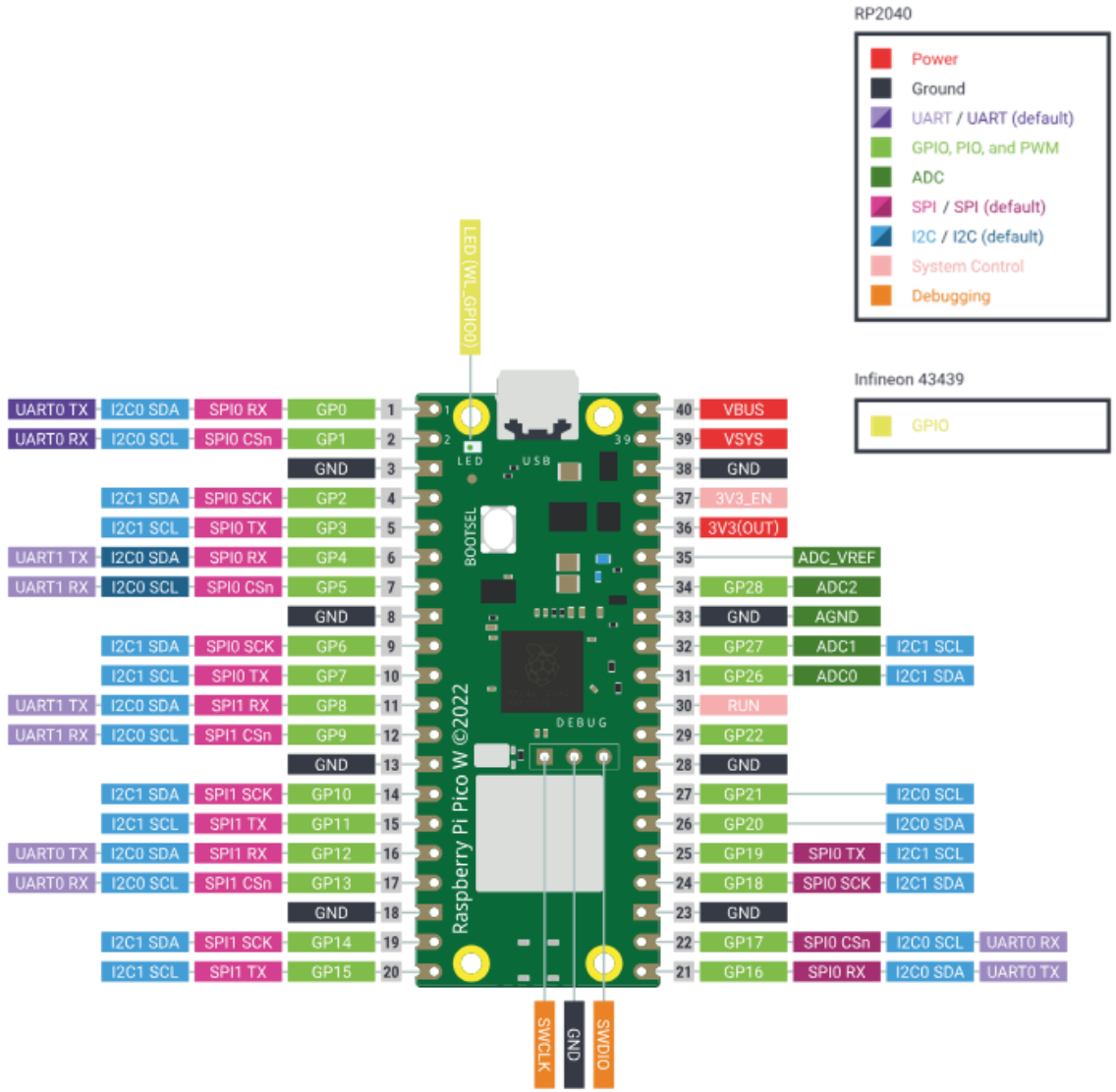
- **Power Pins:** These pins provide various voltage levels to power the Raspberry Pi Pico and connected peripherals.
  - **VSYS (Pin 39):** This is the main input power pin for the Pico, which can accept a range of voltages from 1.8V to 5.5V. It powers the onboard 3.3V regulator.
  - **3V3 (Pins 36, 37):** Provides 3.3V output, which is the regulated voltage supplied by the Pico for powering the microcontroller and any connected 3.3V devices.
  - **3V3_EN (Pin 37):** An enable pin for the 3.3V regulator. Pulling this pin low disables the 3.3V output.
  - **VBUS (Pin 40):** This pin provides the bus voltage (typically 5V output) when the Pico is connected to a computer or power supply.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

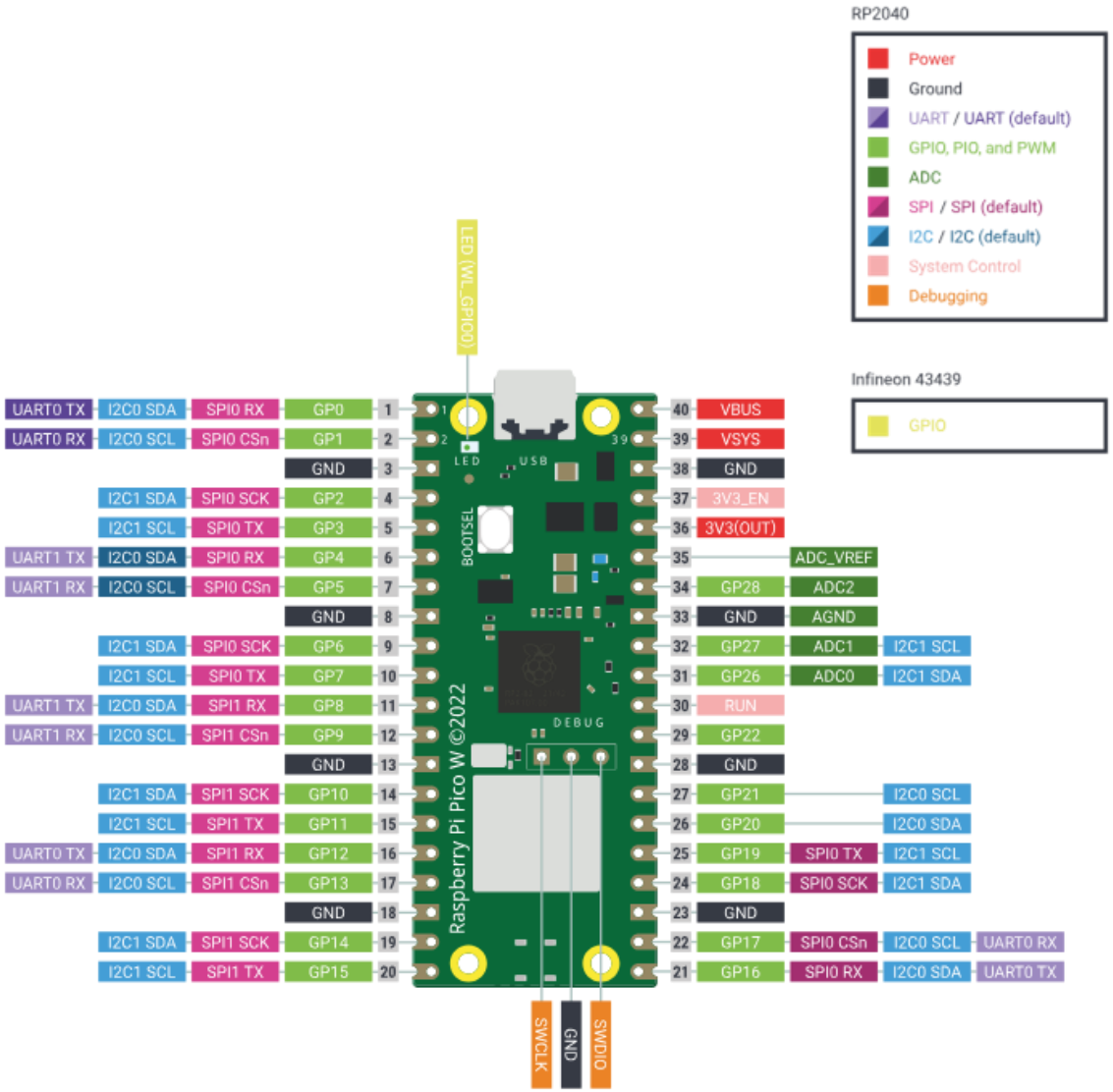
- **Power Pins:** These pins provide various voltage levels to power the Raspberry Pi Pico and connected peripherals.
  - **VSYS (Pin 39):** This is the main input power pin for the Pico, which can accept a range of voltages from 1.8V to 5.5V. It powers the onboard 3.3V regulator.
  - **3V3 (Pins 36, 37):** Provides 3.3V output, which is the regulated voltage supplied by the Pico for powering the microcontroller and any connected 3.3V devices.
  - **3V3_EN (Pin 37):** An enable pin for the 3.3V regulator. Pulling this pin low disables the 3.3V output.
  - **VBUS (Pin 40):** This pin provides the bus voltage (typically 5V output) when the Pico is connected to a computer or power supply.
  - **Ground Pins (GND):** There are several GND pins on the board, providing a common ground reference for the Pico and connected components. **VERY IMPORTANT!**



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

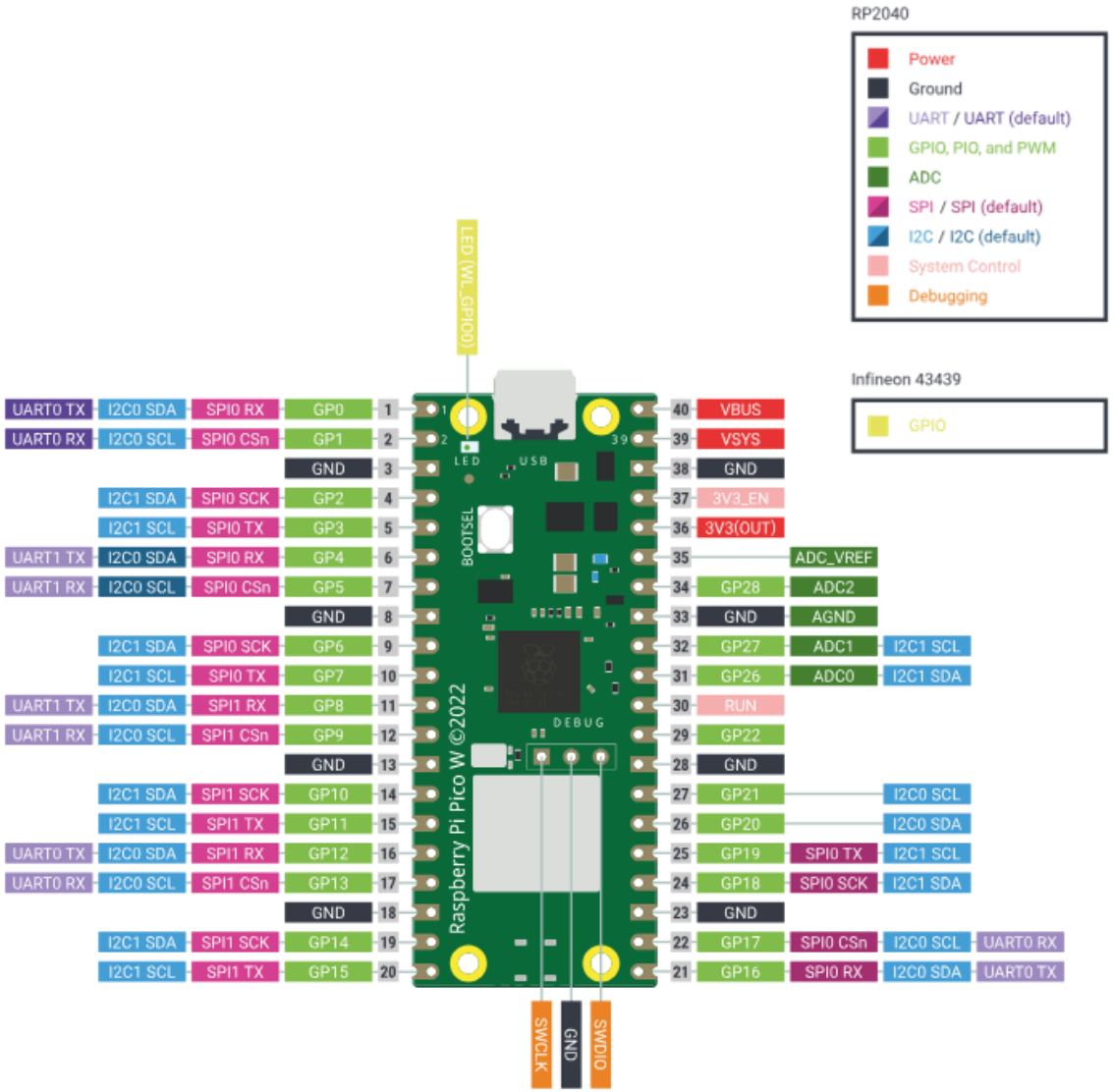
- **Special Function Pins:** Serve specific purposes beyond general GPIO.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

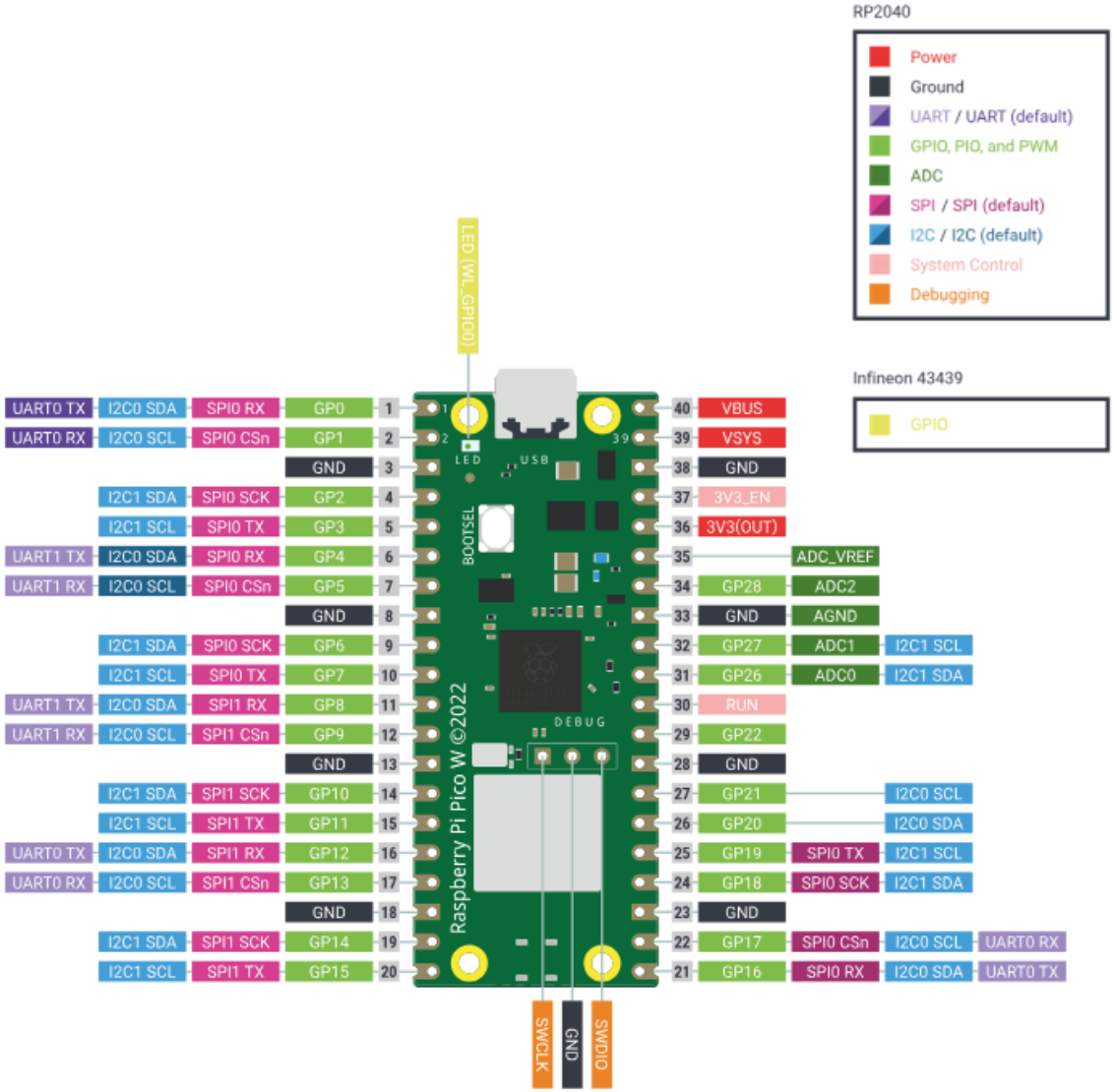
- **Special Function Pins:** Serve specific purposes beyond general GPIO.
  - **RUN (Pin 30):** This pin can be used to reset the microcontroller. Pulling it low will reset the Pico.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

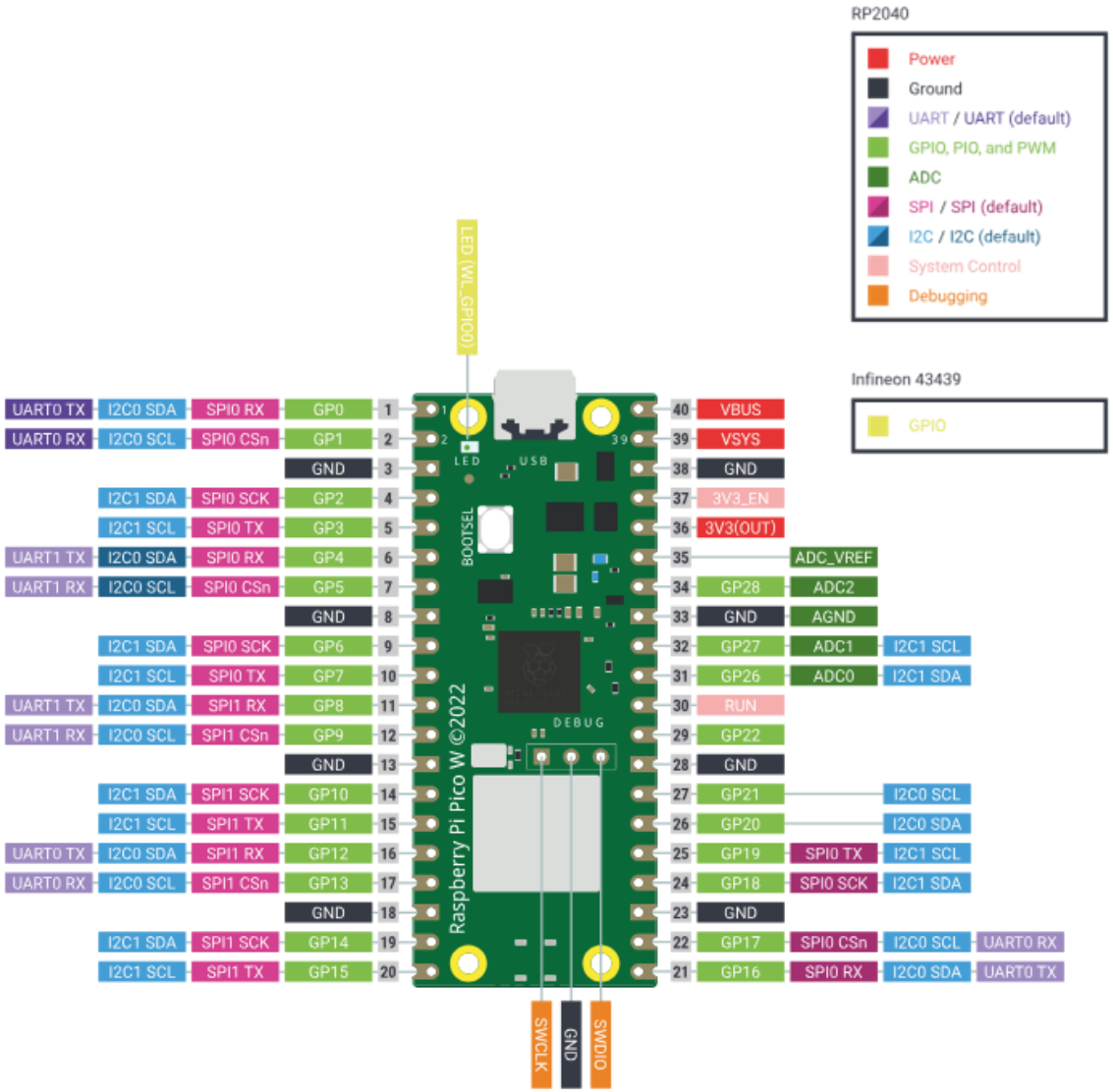
- **Special Function Pins:** Serve specific purposes beyond general GPIO.
  - **RUN (Pin 30):** This pin can be used to reset the microcontroller. Pulling it low will reset the Pico.
  - **ADC_VREF (Pin 35):** This pin provides an external voltage reference for the ADCs, allowing more accurate analog readings if needed.



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

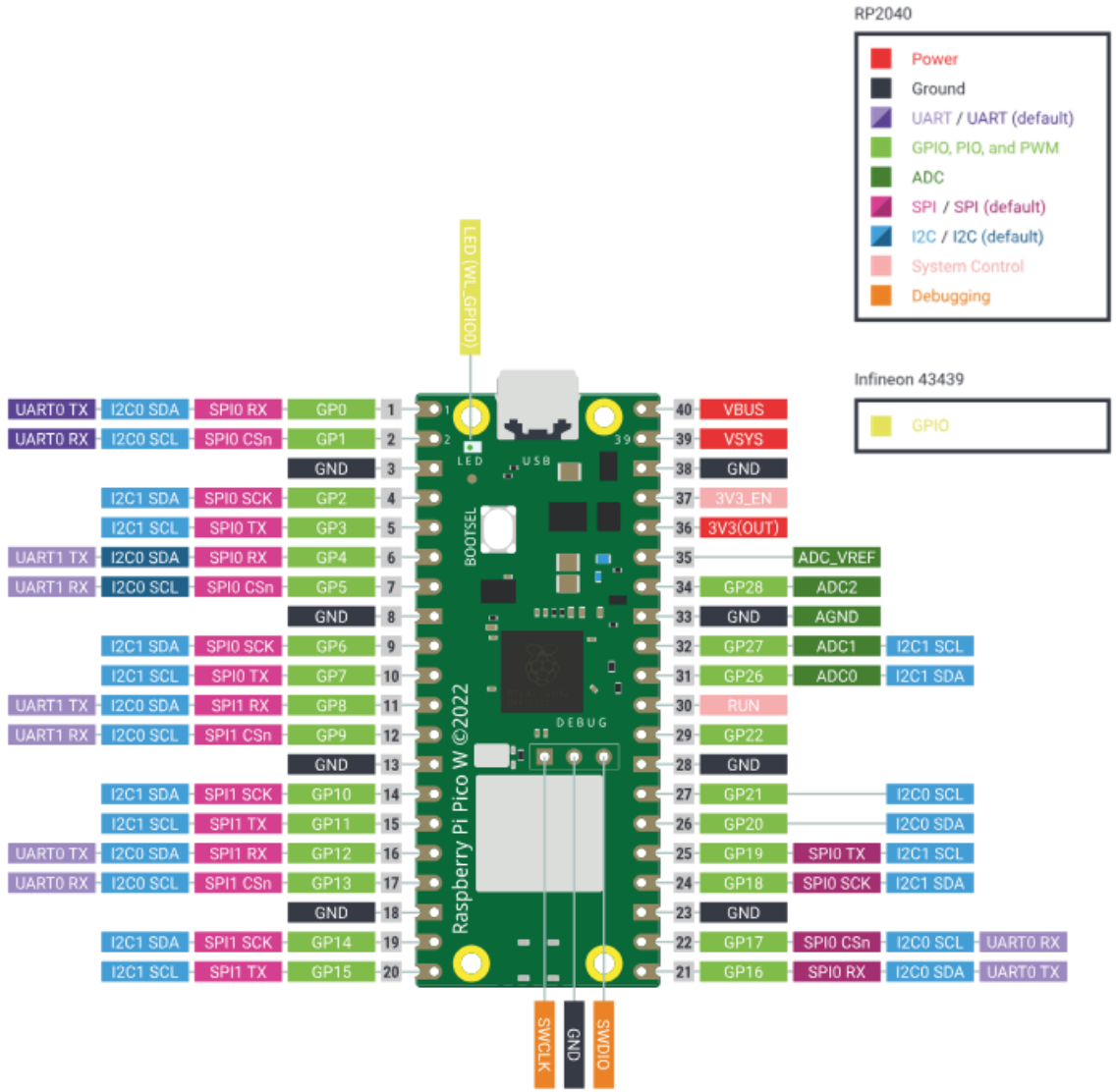
- **Special Function Pins:** Serve specific purposes beyond general GPIO.
  - **RUN (Pin 30):** This pin can be used to reset the microcontroller. Pulling it low will reset the Pico.
  - **ADC_VREF (Pin 35):** This pin provides an external voltage reference for the ADCs, allowing more accurate analog readings if needed.
  - **Serial Wire Debug (SWDIO / SWCLK):** These pins are used for SWD interface, useful for debugging and programming the Pico. (PicoProbe)



Pico W: https://picow.pinout.xyz/

Raspberry Pi Pico and Pico W

- **Special Function Pins:** Serve specific purposes beyond general GPIO.
  - **RUN (Pin 30):** This pin can be used to reset the microcontroller. Pulling it low will reset the Pico.
  - **ADC_VREF (Pin 35):** This pin provides an external voltage reference for the ADCs, allowing more accurate analog readings if needed.
  - **Serial Wire Debug (SWDIO / SWCLK):** These pins are used for SWD interface, useful for debugging and programming the Pico. (PicoProbe)
  - **LED (GP25 / WL_GP00):** This pin controls the onboard LED, and is very useful for simple status indicators or debugging purposes, allowing you to easily provide visual feedback from your code.

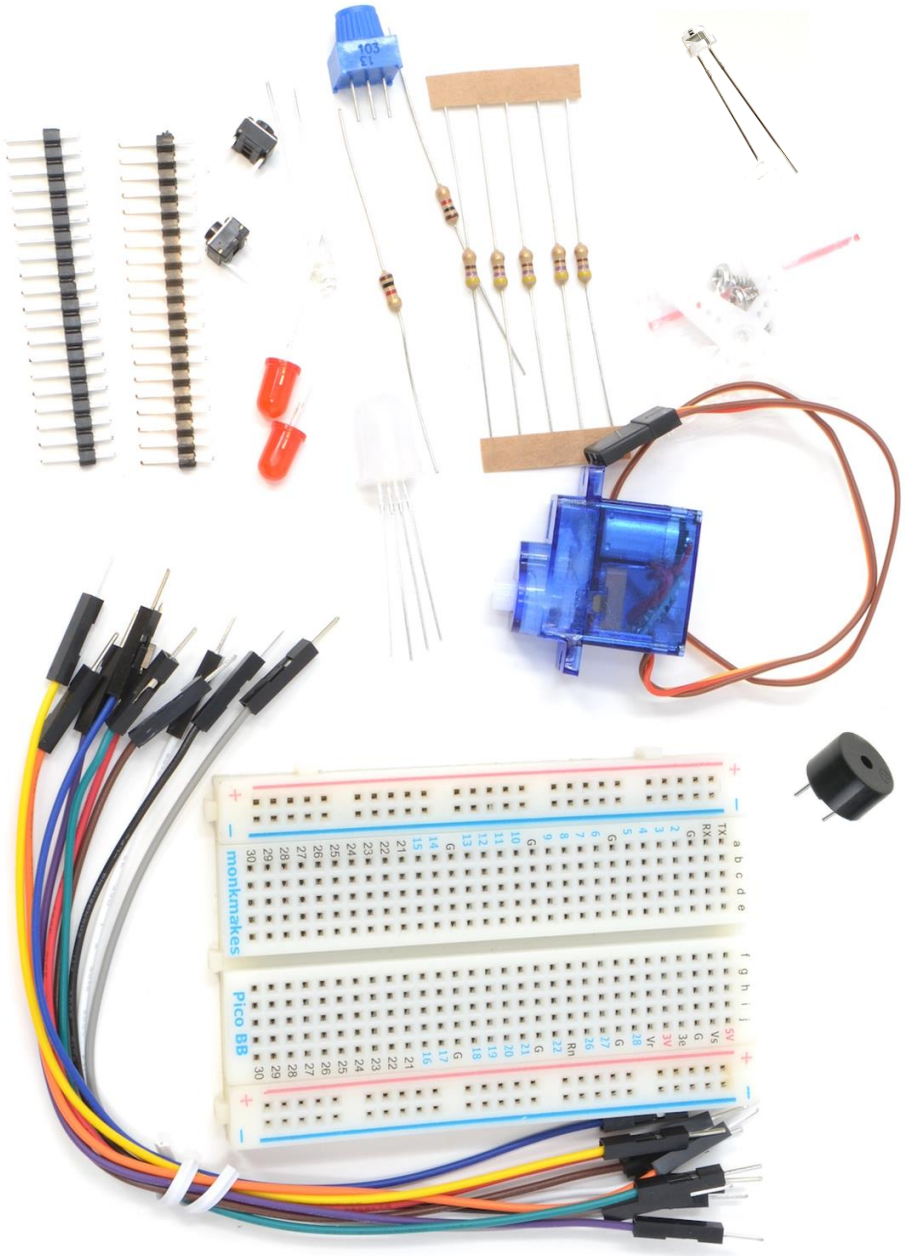


Pico W: https://picow.pinout.xyz/

Monk Makes Electronics Kit 1 for Pico (lite edition)

- A **beginner-friendly electronics kit** designed to work with the Raspberry Pi Pico.

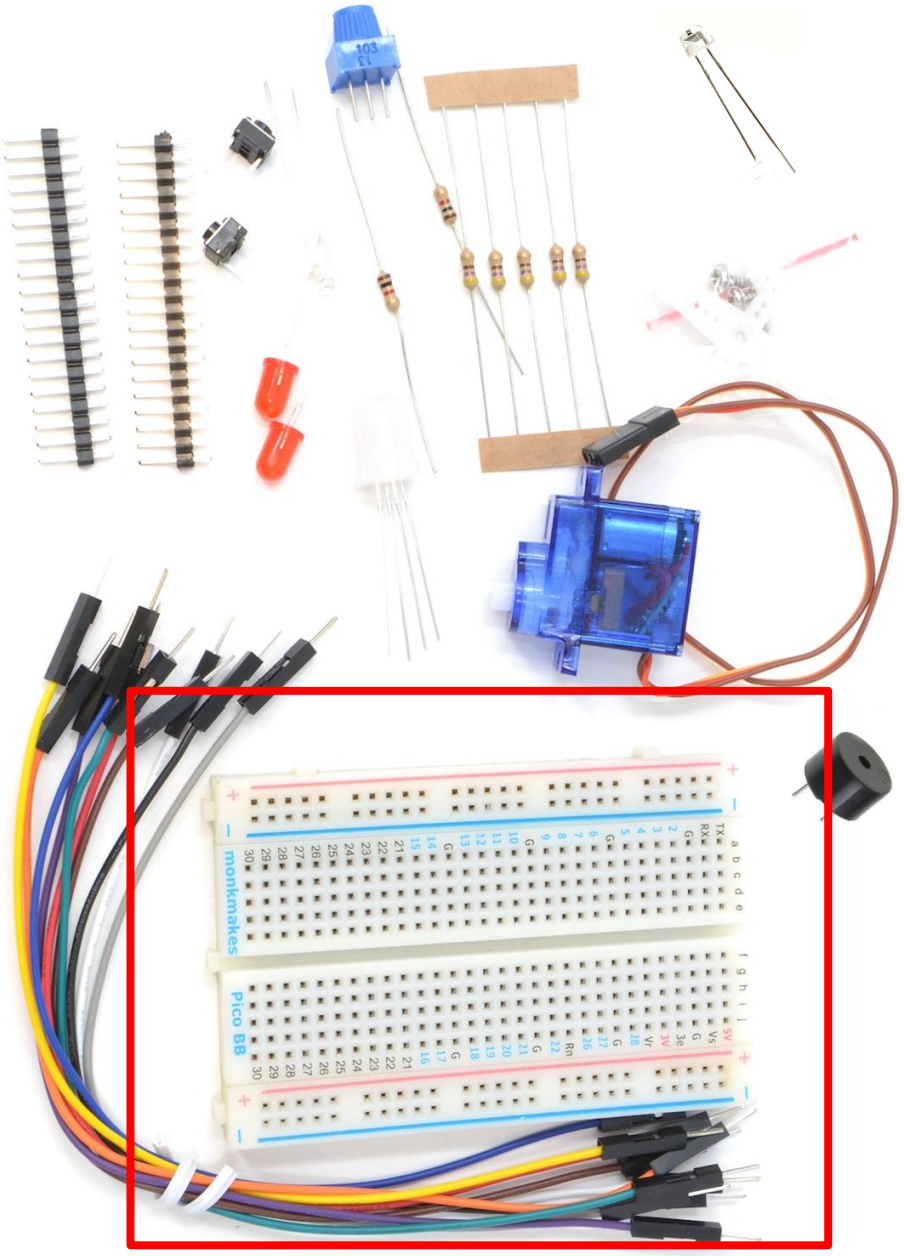
- The kit provides an introduction to electronics and programming, offering a variety of components and projects to help users get started with the Raspberry Pi Pico.



More details at [Instructions: Electronics Kit 1 for Pico]

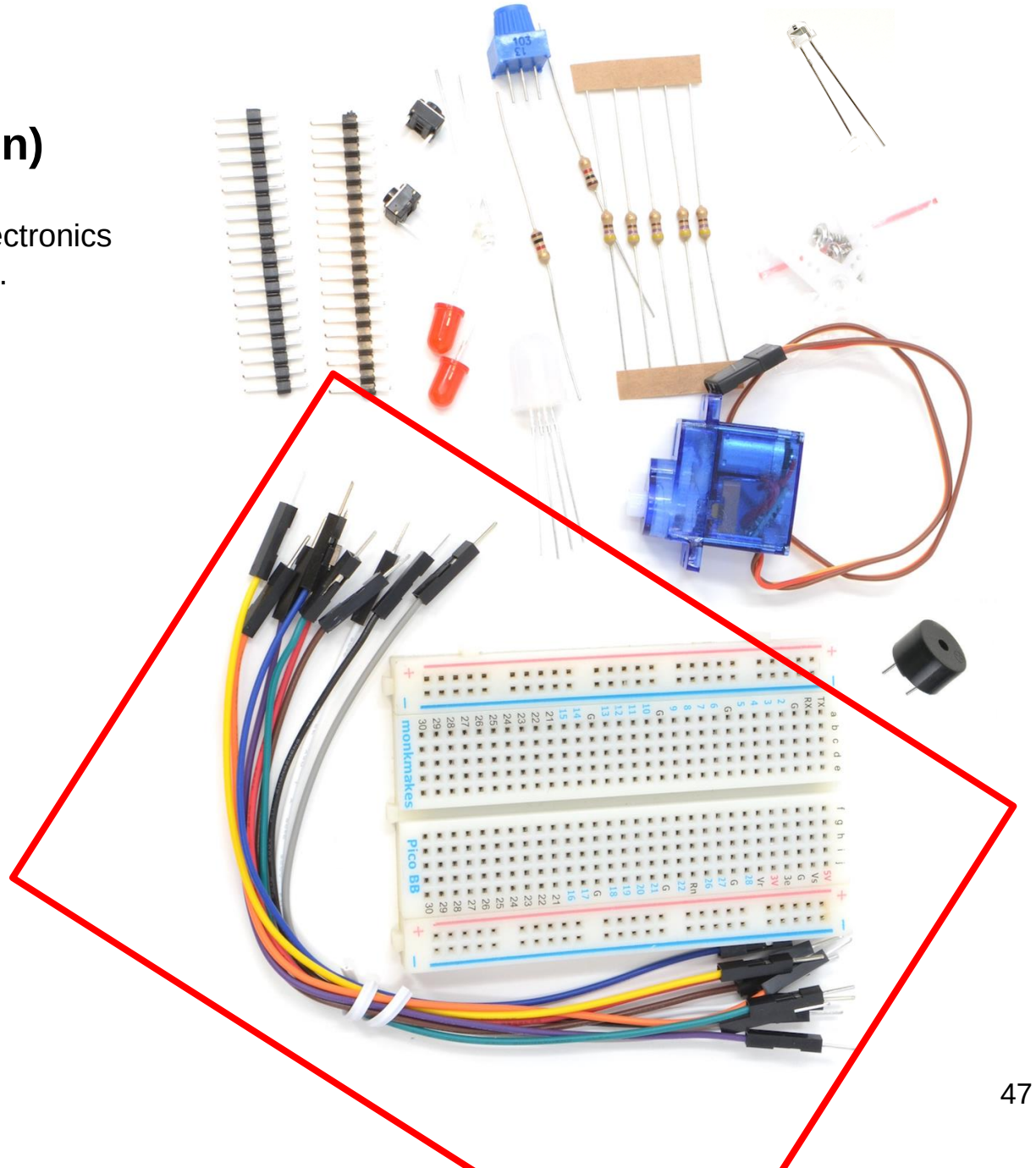
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.



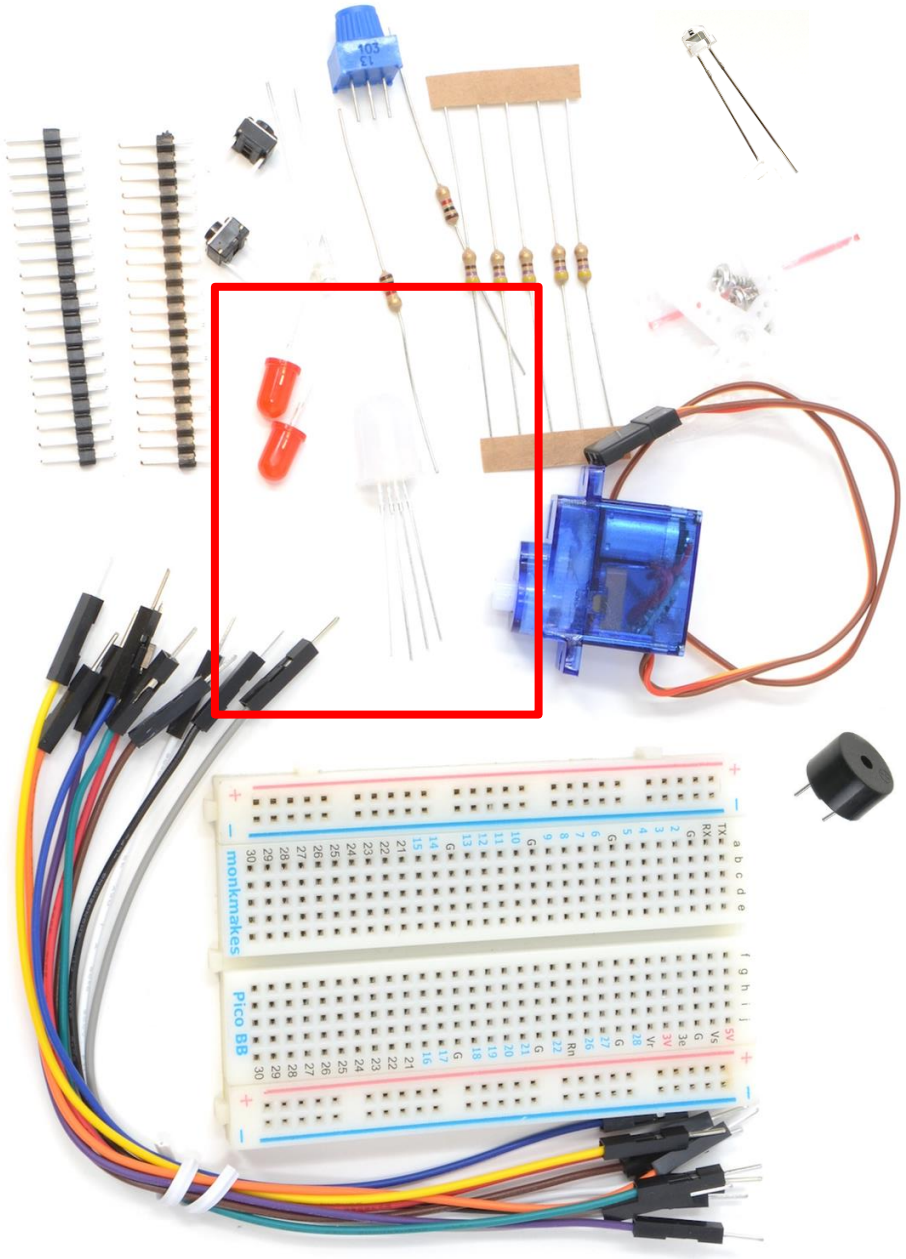
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.



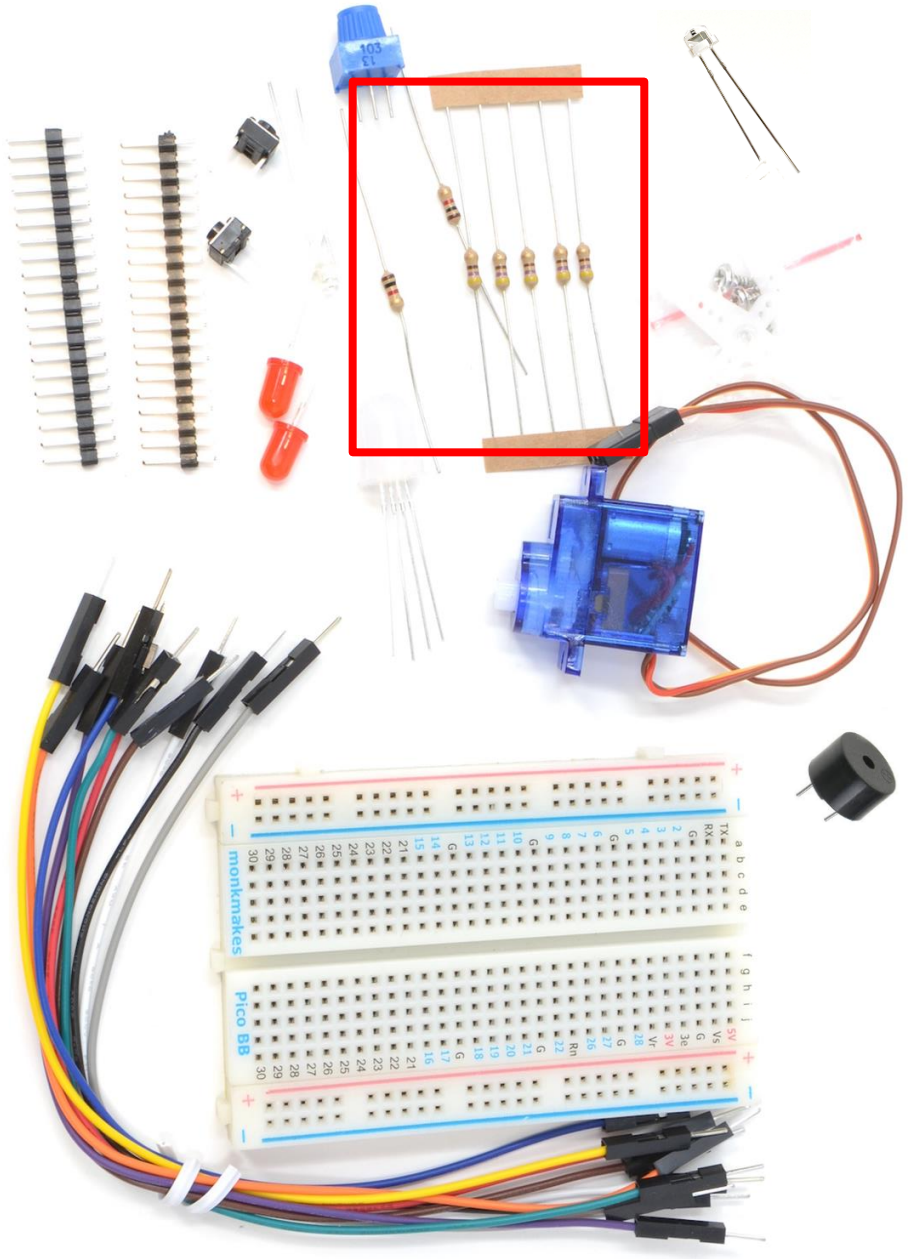
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.



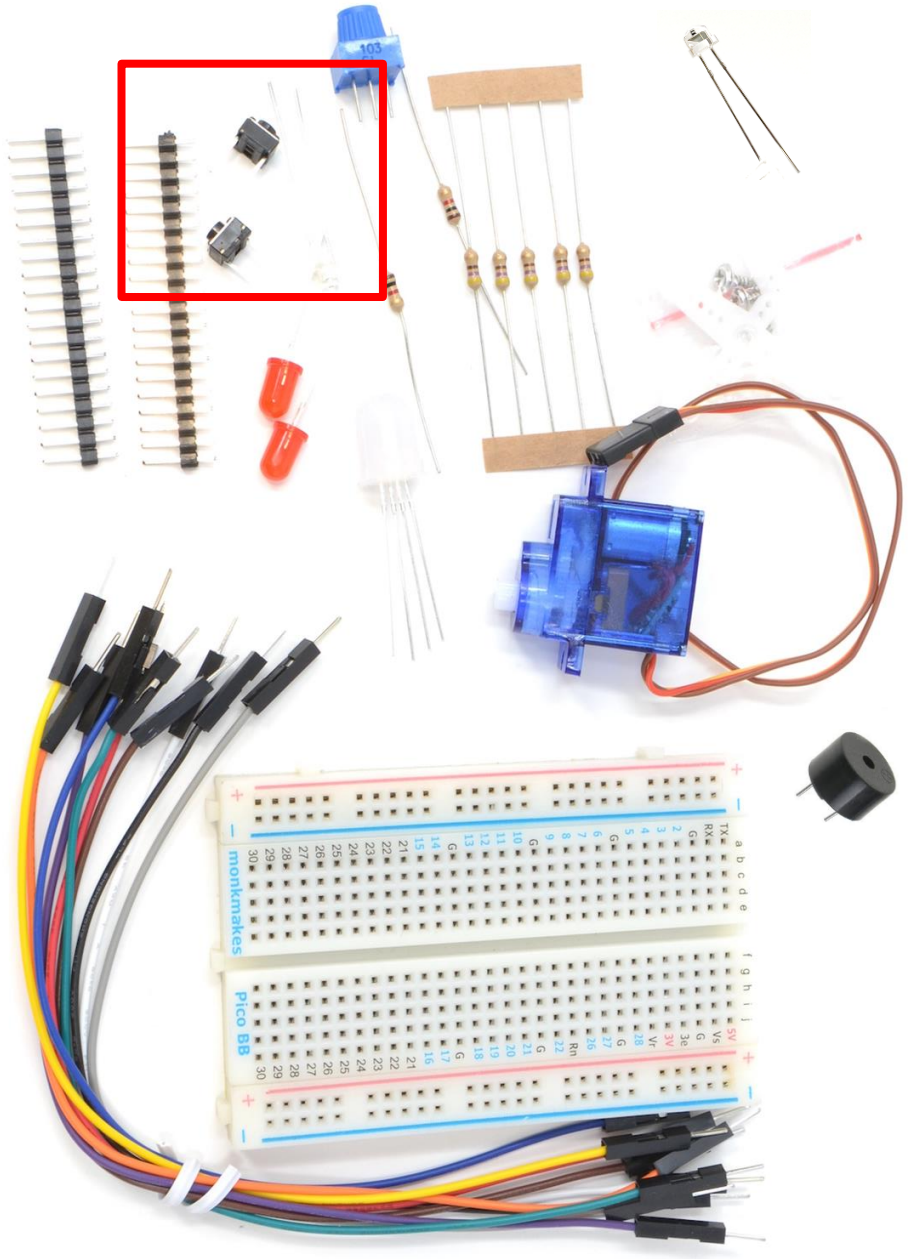
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.
  - **Resistors:** Components that limit current flow, protecting other components and controlling voltages.



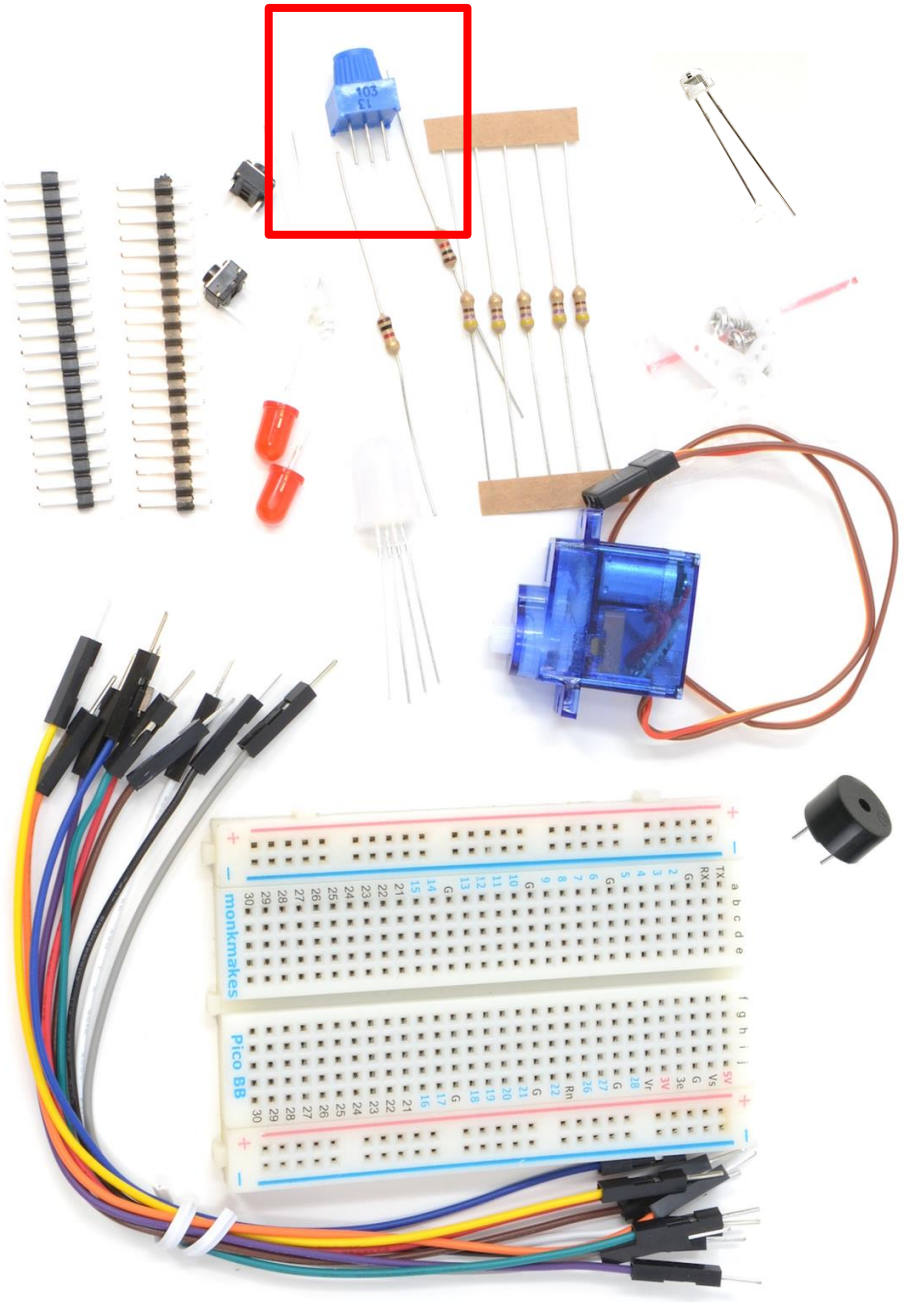
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.
  - **Resistors:** Components that limit current flow, protecting other components and controlling voltages.
  - **Push Button:** A momentary switch used to interact with the circuits.



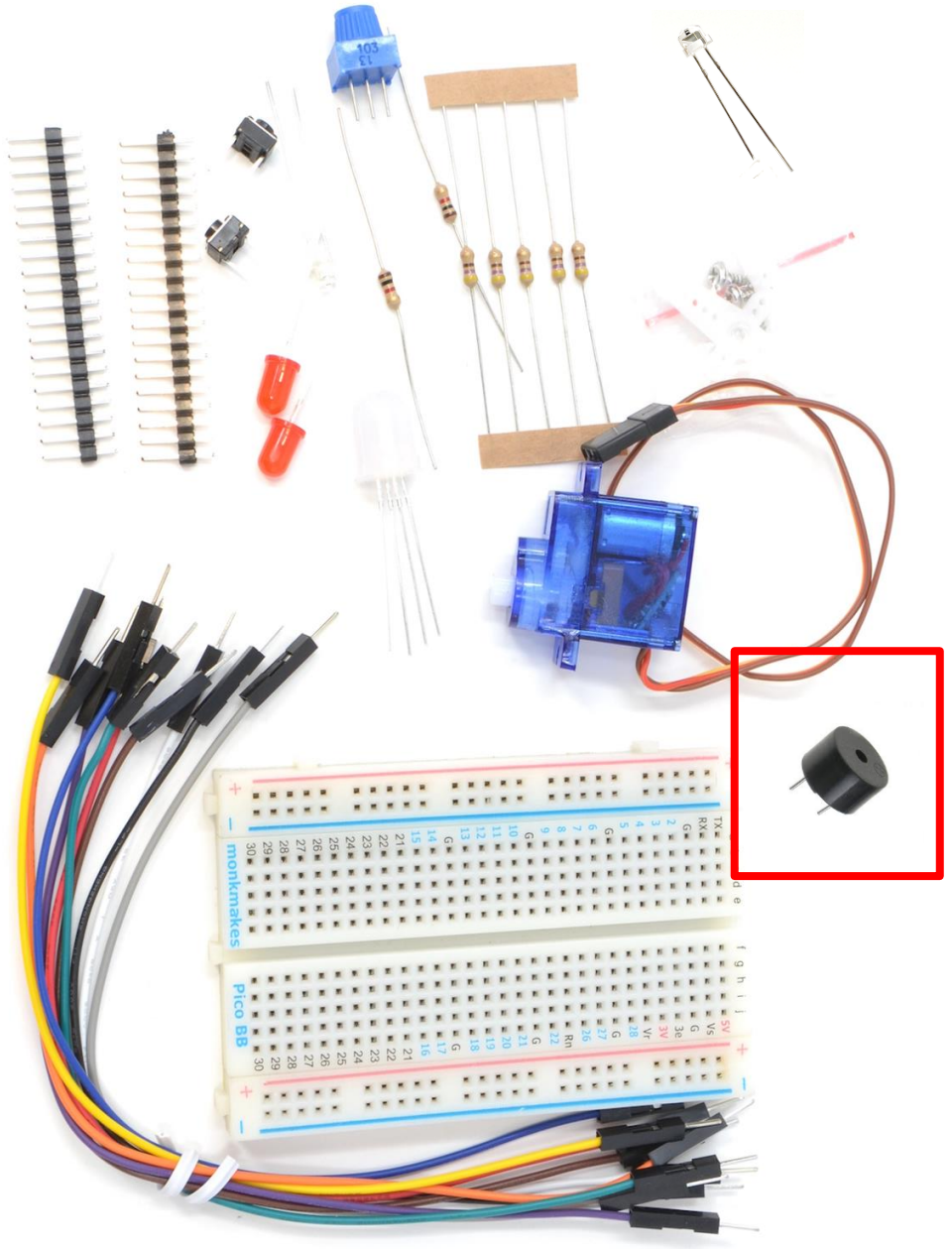
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.
  - **Resistors:** Components that limit current flow, protecting other components and controlling voltages.
  - **Push Button:** A momentary switch used to interact with the circuits.
  - **Potentiometer:** A variable resistor used to adjust voltage levels.



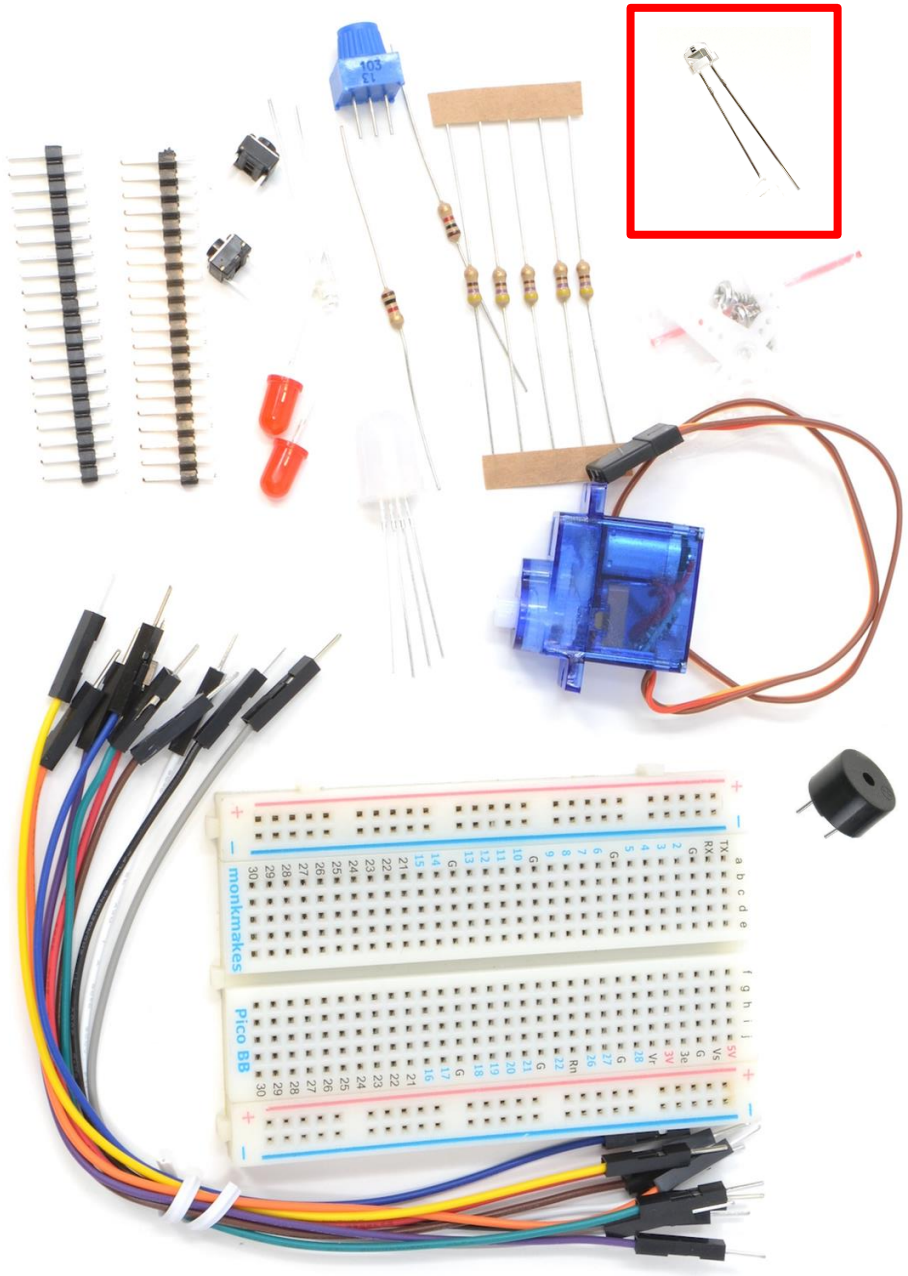
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.
  - **Resistors:** Components that limit current flow, protecting other components and controlling voltages.
  - **Push Button:** A momentary switch used to interact with the circuits.
  - **Potentiometer:** A variable resistor used to adjust voltage levels.
  - **Piezo Buzzer:** A component that produces sound, used for audio output.



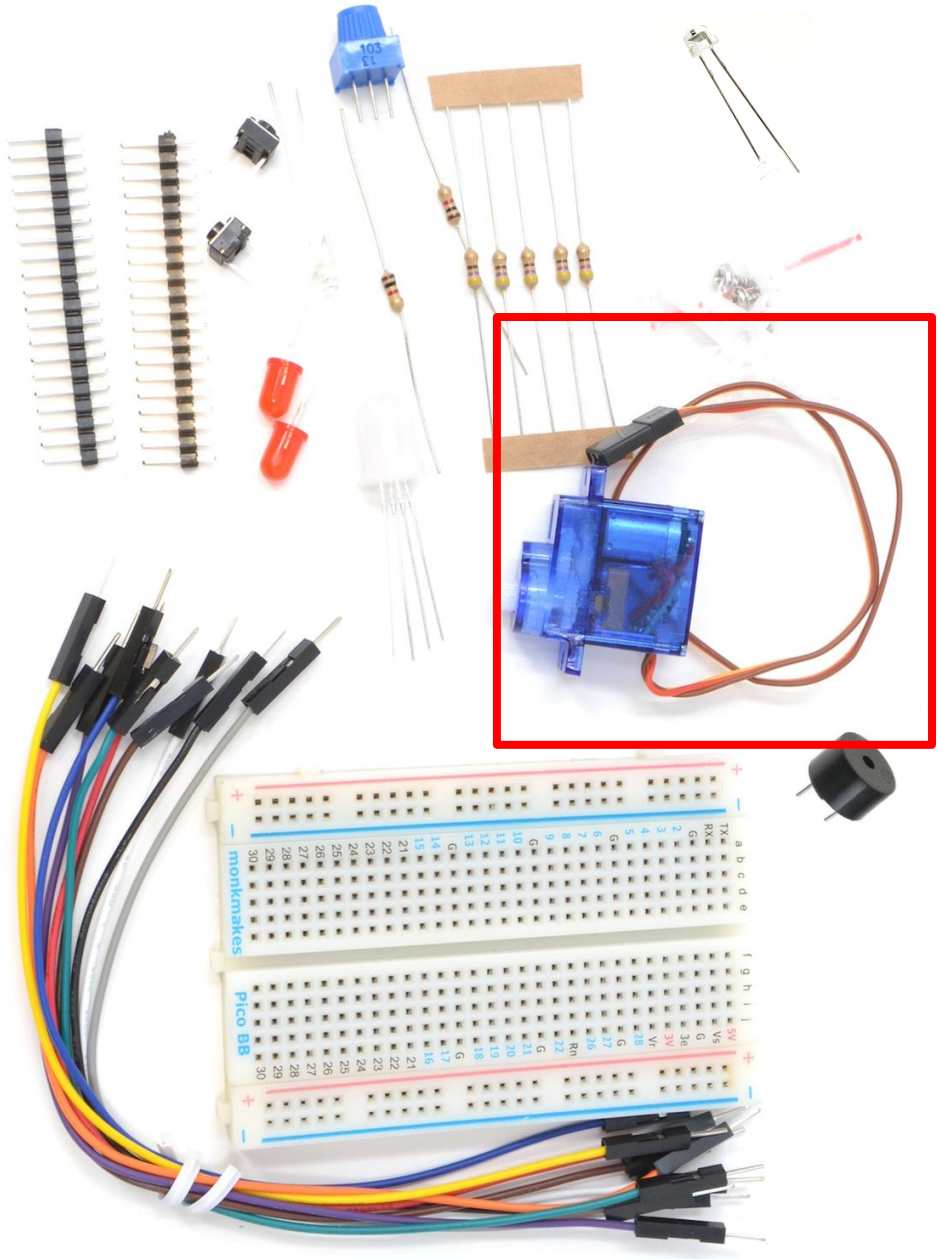
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.
  - **Resistors:** Components that limit current flow, protecting other components and controlling voltages.
  - **Push Button:** A momentary switch used to interact with the circuits.
  - **Potentiometer:** A variable resistor used to adjust voltage levels.
  - **Piezo Buzzer:** A component that produces sound, used for audio output.
  - **Phototransistor:** A light-sensitive component used for detecting light levels.



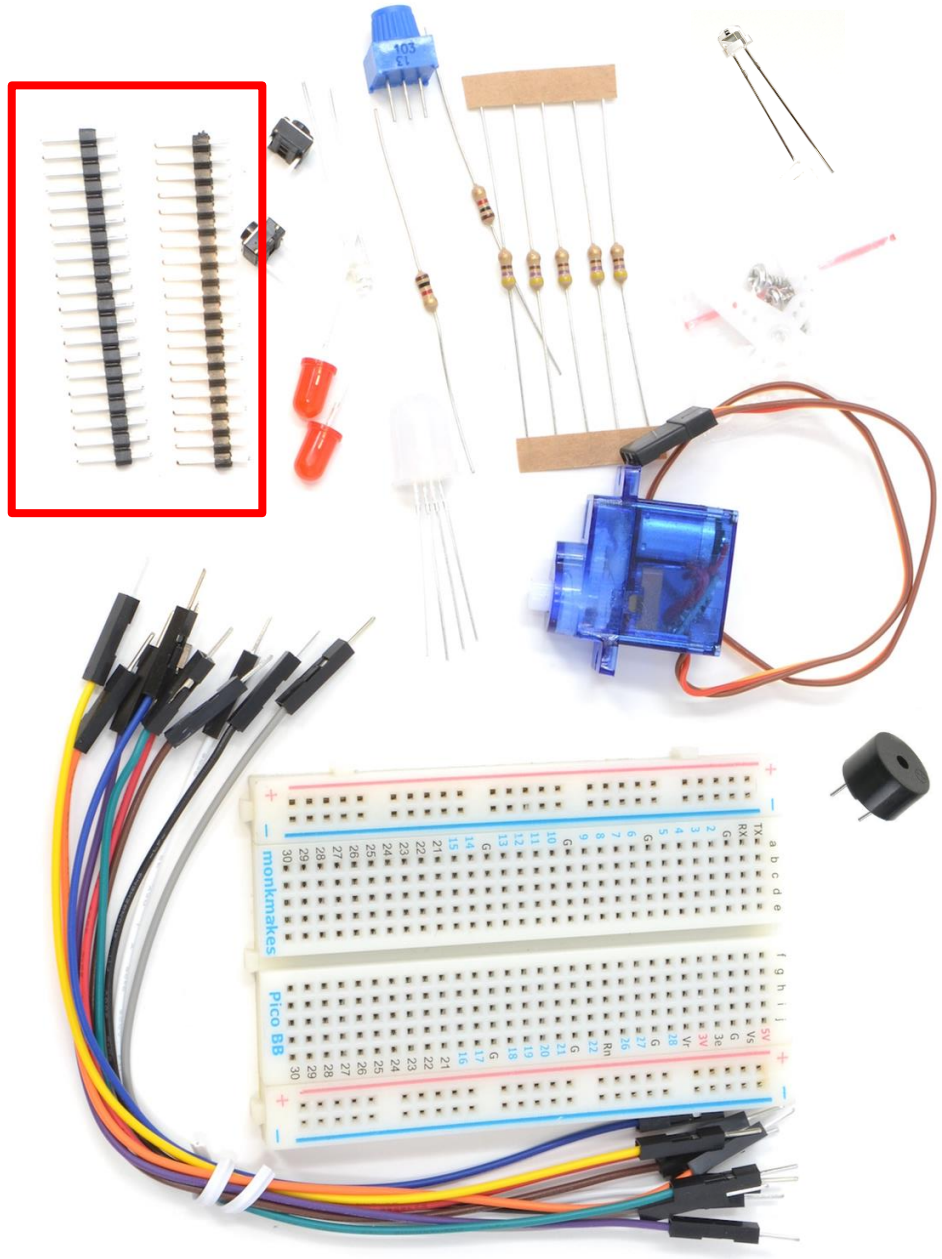
Monk Makes Electronics Kit 1 for Pico (lite edition)

- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.
  - **Resistors:** Components that limit current flow, protecting other components and controlling voltages.
  - **Push Button:** A momentary switch used to interact with the circuits.
  - **Potentiometer:** A variable resistor used to adjust voltage levels.
  - **Piezo Buzzer:** A component that produces sound, used for audio output.
  - **Phototransistor:** A light-sensitive component used for detecting light levels.
  - **Servo motor:** A motor designed to provide precise control of angular or linear position, controlled by PWM (Pulse Width Modulation).



Monk Makes Electronics Kit 1 for Pico (lite edition)

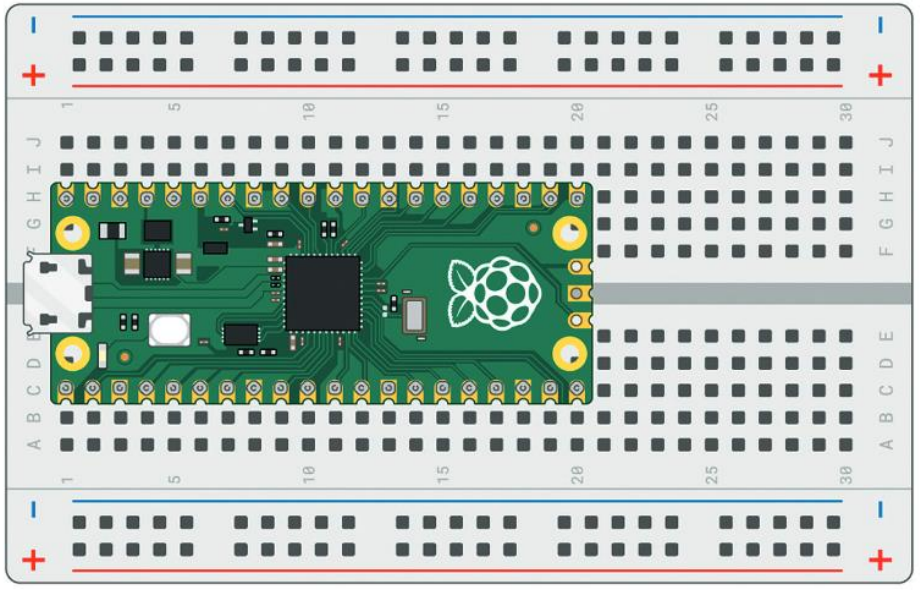
- **Components**
  - **Breadboard with Pico pin names:** A board for prototyping electronics without soldering, allowing easy connections and modifications.
  - **Jumper Wires:** Used to connect different components on the breadboard and to the Pico.
  - **LEDs:** Light-emitting diodes in various colors for visual output in projects.
  - **Resistors:** Components that limit current flow, protecting other components and controlling voltages.
  - **Push Button:** A momentary switch used to interact with the circuits.
  - **Potentiometer:** A variable resistor used to adjust voltage levels.
  - **Piezo Buzzer:** A component that produces sound, used for audio output.
  - **Phototransistor:** A light-sensitive component used for detecting light levels.
  - **Servo motor:** A motor designed to provide precise control of angular or linear position, controlled by PWM (Pulse Width Modulation).
  - **Header pins:** Can be soldered to your Raspberry Pi Pico or sensors.



More details at [Instructions: Electronics Kit 1 for Pico]

Breadboard layout

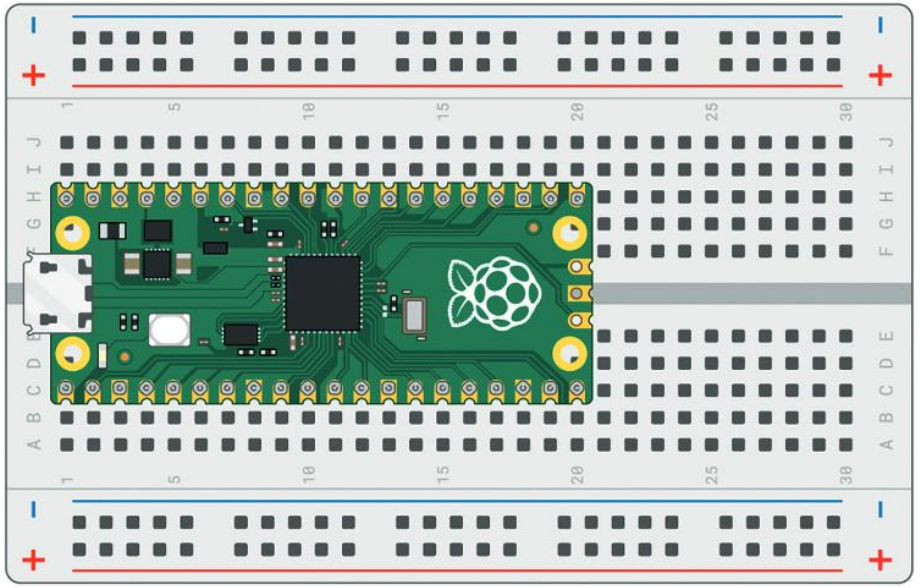
"Great for quickly building and testing circuits, definitely not for production…"



Breadboard layout

"Great for quickly building and testing circuits, definitely not for production…"

- **Terminal Strips**
  - Divided into two sections by a center channel.
  - Each row of **five holes** is **electrically connected**.
  - … and are used for placing components.

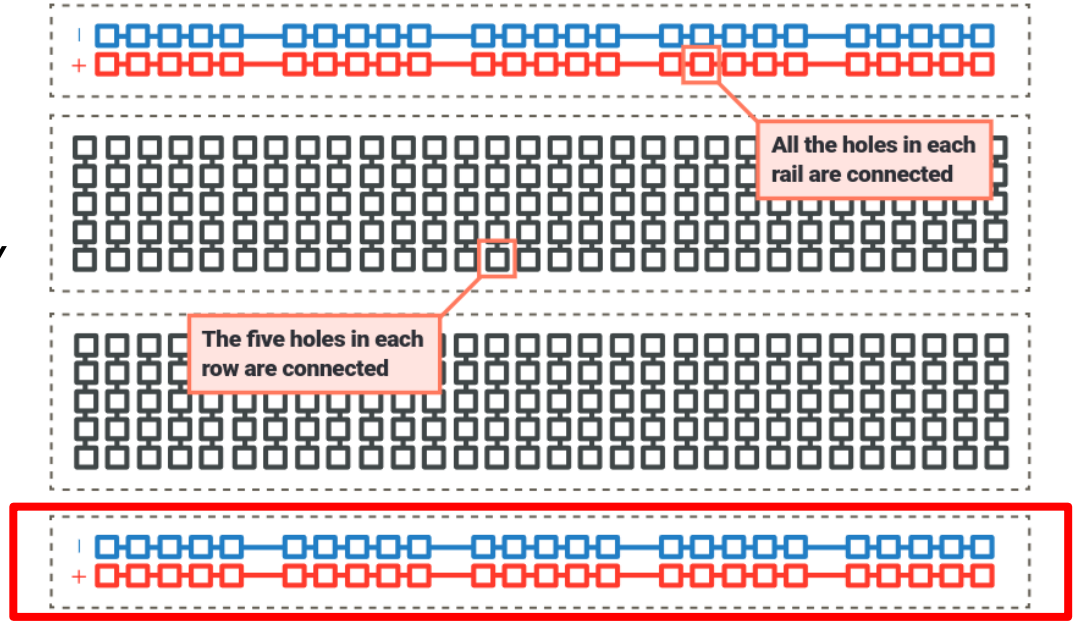


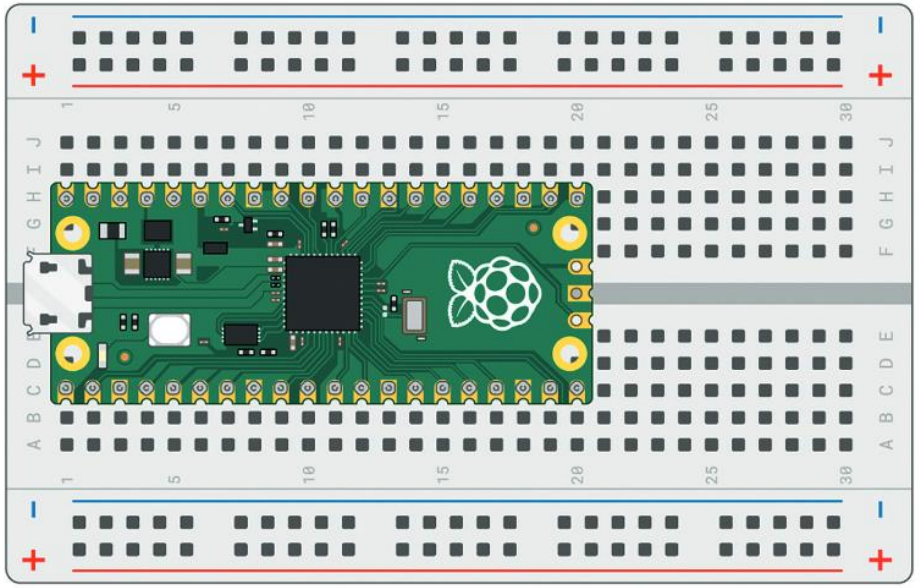
Breadboard layout

"Great for quickly building and testing circuits, definitely not for production…"

- **Terminal Strips**
  - Divided into two sections by a center channel.
  - Each row of **five holes** is **electrically connected**.
  - … and are used for placing components.

- **Power Rails**
  - Usually marked with
    - a red line (for positive voltage), and
    - a blue or black line (for ground).
  - They run along the sides and are connected along their length.
  - …and are used to distribute power and ground





Breadboard layout

"Great for quickly building and testing circuits, definitely not for production…"

- **Example:** Circuit for blinking with an LED
  - Place the LED on the breadboard with
    - the long leg (anode)
    - short leg (cathode)
  - Place a resistor with one end to the anode and the other to a GPIO
  - Use jumpwire to connect the rest…
  - …and run code that makes it blink! **(later…)**

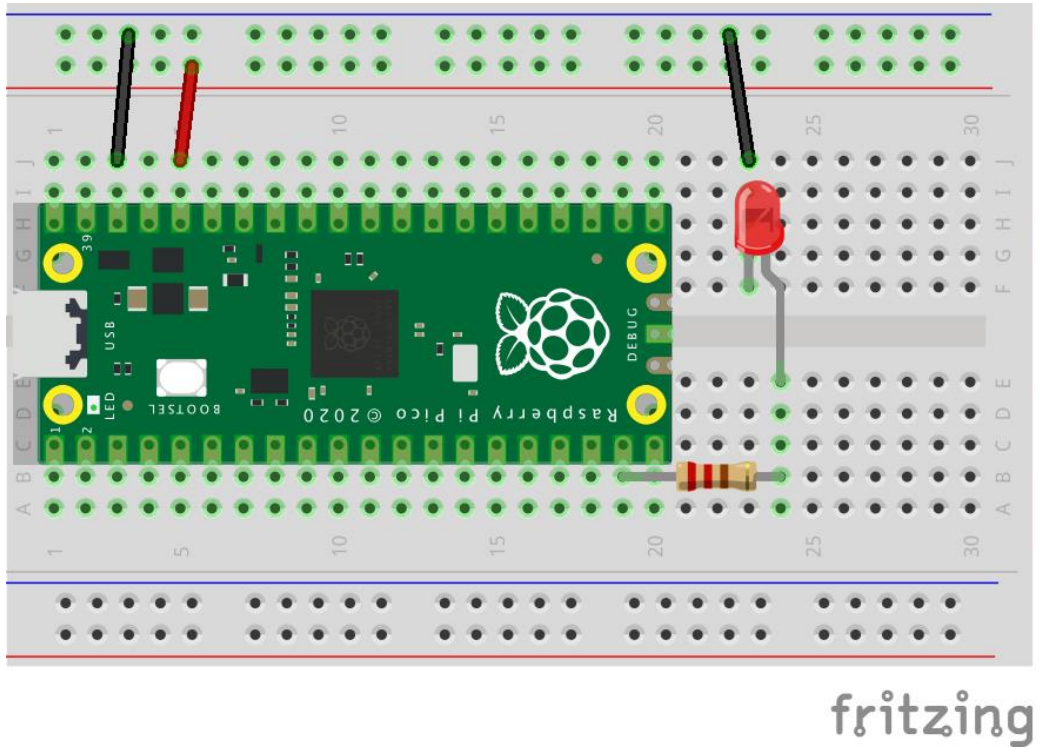


Illustration: https://scruss.com/blog/2021/02/14/quick-labelled-fritzing-raspberry-pi-pico-layout/

Software walk-through

Thonny Python IDE (Demo)

…and flash the latest firmware

REPL (Read-Eval-Print Loop)

"...an interactive programming environment that allows you to enter individual lines or commands, execute them immediately, and see the results."

= very useful for testing and experimenting with code.

- **Steps to use the REPL:**
  1. **Read**: Read the user input.
     a. You type in a Python statement or expression (e.g., `print("Hello, Pico!")`).
  2. **Eval**: Evaluate your code
     a. The Python interpreter on the Pico evaluates the statement or expression.
  3. **Print**: Print any results
     a. The result is printed to the terminal (if applicable), so you'll see the output immediately.
  4. **Loop**: Loop back to step 1
     a. The REPL stays active, waiting for more input from the user.

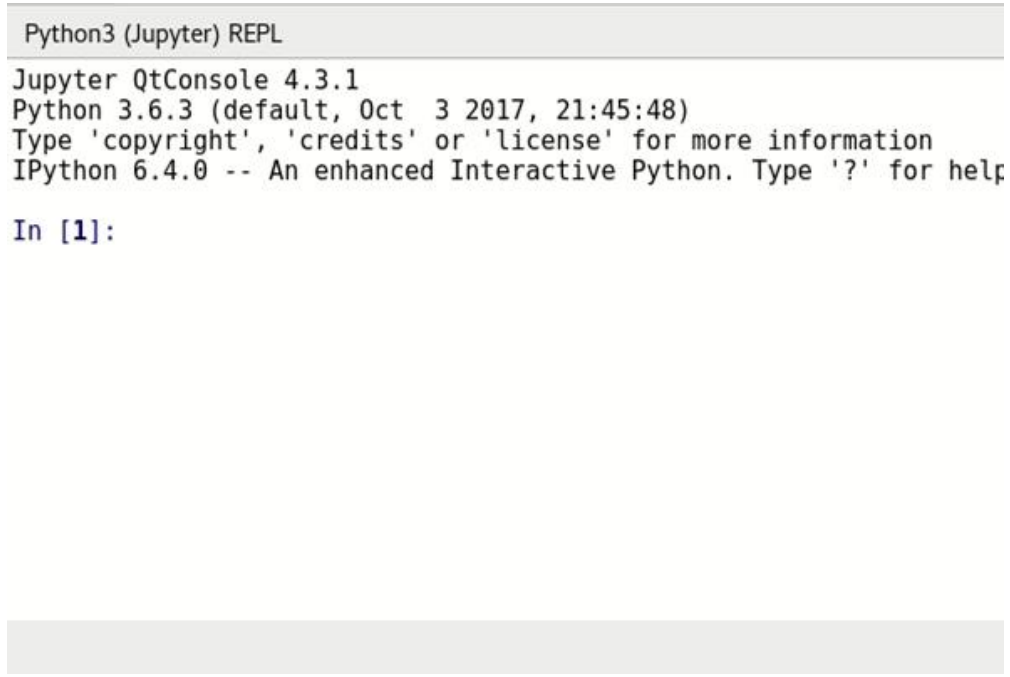


Illustration: https://codewith.mu/en/tutorials/1.2/repl

REPL (Read-Eval-Print Loop)

"...an interactive programming environment that allows you to enter individual lines or commands, execute them immediately, and see the results."

= very useful for testing and experimenting with code.

- **Using the REPL** (eg. via Thonny)**:**
  1. **Connect you Pico to the laptop** using a USB cable
  2. **Open Thonny.**
  3. Select the **MicroPython (Raspberry Pi Pico)** Interpreter
  4. Test the REPL (Shell) window of Thonny with a Python Command, e.g., `print("Hello, Pico!")`.

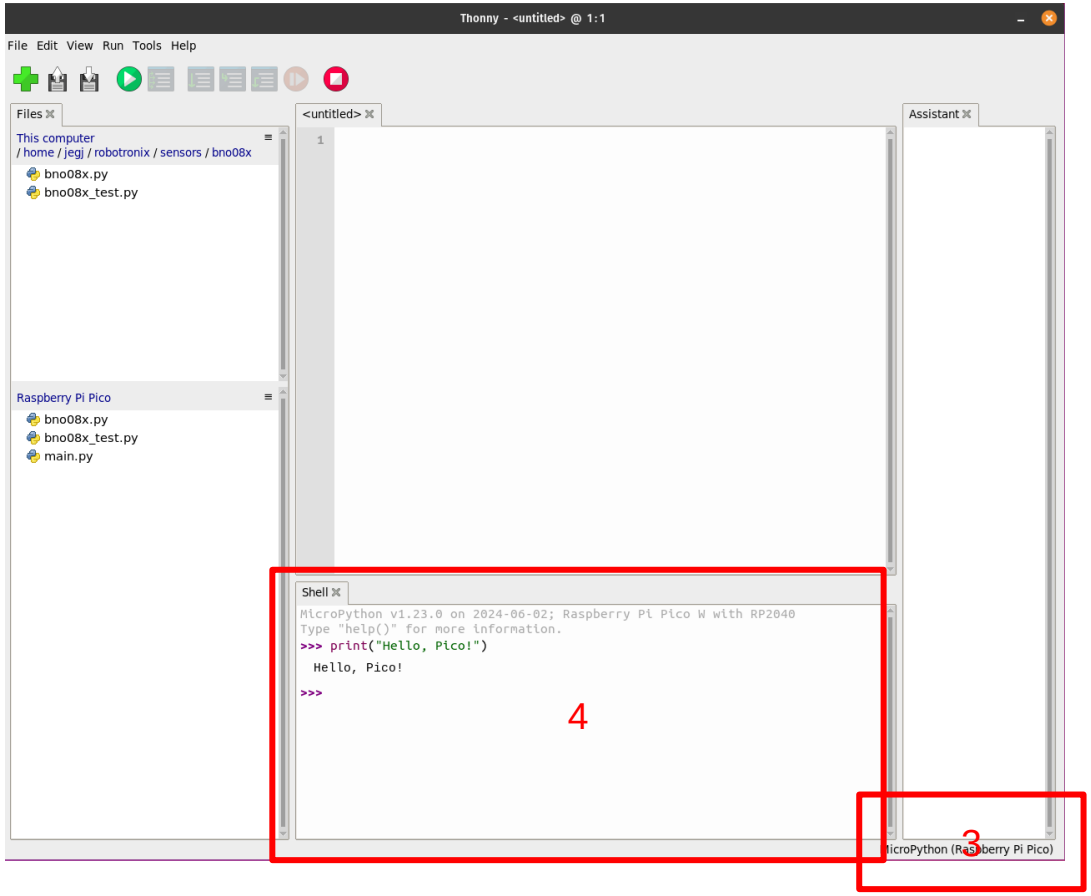
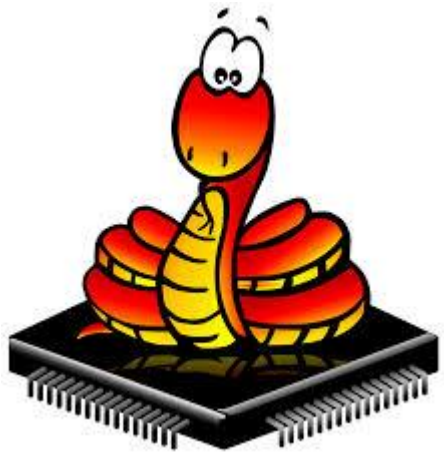


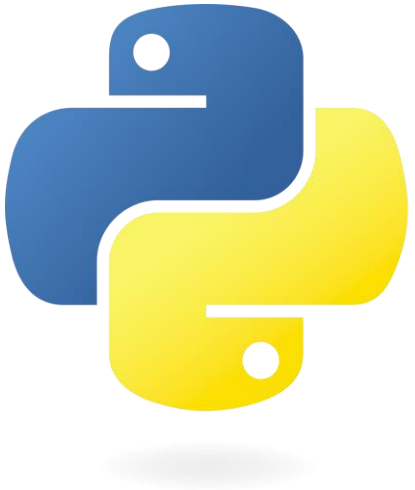
Illustration: https://scruss.com/blog/2021/02/14/quick-labelled-fritzing-raspberry-pi-pico-layout/

Standard libraries
and
Micro-libraries

Python standard libraries and micro-libraries

- built-in modules that mirror the functionality of the standard Python libraries.
    - **Python standard library modules**
        - Most modules implement a subset of the functionality of the equivalent Python module, and in a few cases provide some MicroPython-specific extensions (e.g. os, time)
    - **MicroPython-specific modules** (e.g. bluetooth, machine)
    - **Platform-specific library modules** (e.g. rp2, esp32)





Python standard libraries and micro-libraries

Python standard libraries and micro-libraries

- `array` – arrays of numeric data
- `asyncio` — asynchronous I/O scheduler
- `binascii` – binary/ASCII conversions
- `builtins` – builtin functions and exceptions
- `cmath` – mathematical functions for complex numbers
- `collections` – collection and container types
- `errno` – system error codes
- `gc` – control the garbage collector
- `gzip` – gzip compression & decompression
- `hashlib` – hashing algorithms
- `heapq` – heap queue algorithm
- `io` – input/output streams
- `json` – JSON encoding and decoding
- `math` – mathematical functions
- `os` – basic “operating system” services
- `platform` – access to underlying platform’s identifying data
- `random` – generate random numbers
- `re` – simple regular expressions
- `select` – wait for events on a set of streams
- `socket` – socket module
- `ssl` – SSL/TLS module
- `struct` – pack and unpack primitive data types
- `sys` – system specific functions
- `time` – time related functions
- `zlib` – zlib compression & decompression
- `_thread` – multithreading support

MicroPython-specific libraries

- `bluetooth` — low-level Bluetooth
- `btree` – simple BTree database
- `cryptolib` – cryptographic ciphers
- `deflate` – deflate compression & decompression
- `framebuf` — frame buffer manipulation
- `machine` — functions related to the hardware
- `rp2` — functionality specific to the RP2040
- `micropython` – access and control MicroPython internals
- `neopixel` — control of WS2812 / NeoPixel LEDs
- `network` — network configuration
- `openamp` – provides standard Asymmetric Multiprocessing (AMP) support
- `uctypes` – access binary data in a structured way
- `vfs` – virtual filesystem control

Python standard libraries and micro-libraries **(...probably use during the course)**

Python standard libraries and micro-libraries

- `array` – arrays of numeric data
- `asyncio` — asynchronous I/O scheduler
- `binascii` – binary/ASCII conversions
- `builtins` – builtin functions and exceptions
- `cmath` – mathematical functions for complex numbers
- `collections` – collection and container types
- `errno` – system error codes
- `gc` – control the garbage collector
- `gzip` – gzip compression & decompression
- `hashlib` – hashing algorithms
- `heapq` – heap queue algorithm
- `io` – input/output streams
- `json` – JSON encoding and decoding
- `math` – mathematical functions
- `os` – basic “operating system” services
- `platform` – access to underlying platform’s identifying data
- `random` – generate random numbers
- `re` – simple regular expressions
- `select` – wait for events on a set of streams
- `socket` – socket module
- `ssl` – SSL/TLS module
- `struct` – pack and unpack primitive data types
- `sys` – system specific functions
- `time` – time related functions
- `zlib` – zlib compression & decompression
- `_thread` – multithreading support

MicroPython-specific libraries

- `bluetooth` — low-level Bluetooth
- `btree` – simple BTree database
- `cryptolib` – cryptographic ciphers
- `deflate` – deflate compression & decompression
- `framebuf` — frame buffer manipulation
- `machine` — functions related to the hardware
- `rp2` — functionality specific to the RP2040
- `micropython` – access and control MicroPython internals
- `neopixel` — control of WS2812 / NeoPixel LEDs
- `network` — network configuration
- `openamp` – provides standard Asymmetric Multiprocessing (AMP) support
- `uctypes` – access binary data in a structured way
- `vfs` – virtual filesystem control

Python standard libraries and micro-libraries **(...you will use today)**

Python standard libraries and micro-libraries

- `array` – arrays of numeric data
- `asyncio` — asynchronous I/O scheduler
- `binascii` – binary/ASCII conversions
- `builtins` – builtin functions and exceptions
- `cmath` – mathematical functions for complex numbers
- `collections` – collection and container types
- `errno` – system error codes
- `gc` – control the garbage collector
- `gzip` – gzip compression & decompression
- `hashlib` – hashing algorithms
- `heapq` – heap queue algorithm
- `io` – input/output streams
- `json` – JSON encoding and decoding
- `math` – mathematical functions
- `os` – basic “operating system” services
- `platform` – access to underlying platform’s identifying data
- `random` – generate random numbers
- `re` – simple regular expressions
- `select` – wait for events on a set of streams
- `socket` – socket module
- `ssl` – SSL/TLS module
- `struct` – pack and unpack primitive data types
- `sys` – system specific functions
- `time` – time related functions
- `zlib` – zlib compression & decompression
- `_thread` – multithreading support

MicroPython-specific libraries

- `bluetooth` — low-level Bluetooth
- `btree` – simple BTree database
- `cryptolib` – cryptographic ciphers
- `deflate` – deflate compression & decompression
- `framebuf` — frame buffer manipulation
- `machine` — functions related to the hardware
- `rp2` — functionality specific to the RP2040
- `micropython` – access and control MicroPython internals
- `neopixel` — control of WS2812 / NeoPixel LEDs
- `network` — network configuration
- `openamp` – provides standard Asymmetric Multiprocessing (AMP) support
- `uctypes` – access binary data in a structured way
- `vfs` – virtual filesystem control

Key modules, classes and functions

- **`machine` module:** The module for interfacing with the hardware of a microcontroller.

Illustration: https://i.redd.it/76oyebb1i1a61.png

Classes (`machine` module)
`class Pin` – control I/O pins
`class Signal` – control and sense external I/O devices
`class ADC` – analog to digital conversion
`class ADCBlock` – control ADC peripherals
`class PWM` – pulse width modulation
`class UART` – duplex serial communication bus
`class SPI` – a Serial Peripheral Interface bus protocol (controller side)
`class I2C` – a two-wire serial protocol
`class I2S` – Inter-IC Sound bus protocol
`class RTC` – real time clock
`class Timer` – control hardware timers
`class WDT` – watchdog timer
`class SD` – secure digital memory card (cc3200 port only)
`class SDCard` – secure digital memory card
`class USBDevice` – USB Device driver

Key modules, classes and functions

- **`machine` module:** The module for interfacing with the hardware of a microcontroller.

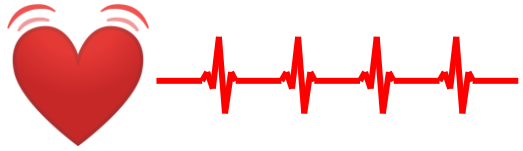
- It provides classes and functions that allow direct control over the hardware, such as:
  - Pins, Timers, ADC (Analog to Digital Converter), PWM (Pulse Width Modulation), and other peripherals.

Example: Import the **`machine`** module

```
import machine

machine.freq() # get the current frequency of the CPU
machine.freq(100_000_000) # set the CPU frequency to 100 MHz
machine.deepsleep(10_000) # Sleep for 10 seconds (low power state)
```

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver



CPU Clock

(The “heartbeat” of the microcontroller)

Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

- **Constructor**
  - `class machine.Pin(id, mode=-1, pull=-1, *, value=None, drive=0, alt=-1)`
- **Methods**:
  - `init()`: Initialize or re-initialize pin parameters.
  - `value()`: Set or get the pin's value.
  - `irq()`: Set up an interrupt handler.
  - `on()` / `off()` / `toggle()`: Set pin high / low / switch to the opposite.

Example: Import **`machine`** module

```
import machine

p0 = machine.Pin("GP0", machine.Pin.OUT) # create an output pin on GP0

p0.value(0) # set the value low / False
p0.value(1) # set the value high / True

# create an input pin on pin #2, with a pull up resistor
p2 = machine.Pin(2, machine.Pin.IN, machine.Pin.PULL_UP)

print(p2.value()) # read and print the pin value
```

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

- **Constructor**
  - `class machine.Pin(id, mode=-1, pull=-1, *, value=None, drive=0, alt=-1)`
- **Methods**:
  - `init()`: Initialize or re-initialize pin parameters.
  - `value()`: Set or get the pin's value.
  - `irq()`: Set up an interrupt handler.
  - `on()` / `off()` / `toggle()`: Set pin high / low / switch to the opposite.

Example: Import **`machine`** module

```
import machine

p0 = machine.Pin("GP0", machine.Pin.OUT) # create an output pin on GP0

p0.value(0) # set the value low / False
p0.value(1) # set the value high / True

# create an input pin on pin #2, with a pull up resistor
p2 = machine.Pin(2, machine.Pin.IN, machine.Pin.PULL_UP)

print(p2.value()) # read and print the pin value
```

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

Constants (What and how?)

- Pin mode
  - Pin.IN
  - Pin.OUT
- Pull up/down resistor
  - Pin.PULL_UP
  - Pin.PULL_DOWN
- IRQ trigger type
  - Pin.IRQ_FALLING
  - Pin.IRQ_RISING

- More available in the documentation

Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

- **Constructor**
  - `class machine.Pin(id, mode=-1, pull=-1, *, value=None, drive=0, alt=-1)`
- **Methods**:
  - `init()`: Initialize or re-initialize pin parameters.
  - `value()`: Set or get the pin's value.
  - `irq()`: Set up an interrupt handler.
  - `on()` / `off()` / `toggle()`: Set pin high / low / switch to the opposite.

Example: Import **`Pin`** for **`machine`** module

```
from machine import Pin # import from machine

p0 = Pin("GP0", Pin.OUT) # create an output pin on GP0

p0.value(0) # set the value low / False
p0.value(1) # set the value high / True

# create an input pin on pin #2, with a pull up resistor
p2 = Pin(2, Pin.IN, Pin.PULL_UP)

print(p2.value()) # read and print the pin value
```

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

`Constants`

- Pin mode
  - Pin.IN
  - Pin.OUT
- Pull up/down resistor (to ensure a known state for a signal)
  - Pin.PULL_UP
  - Pin.PULL_DOWN
- IRQ trigger type (later)
  - Pin.IRQ_FALLING
  - Pin.IRQ_RISING

- More available in the documentation

Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

Example: Blink the onboard LED (Output)

```
from machine import Pin
import time

led = Pin("LED", Pin.OUT)

while True:
    led.value(True)   #turn on the LED
    time.sleep(1)    #wait for one second
    led.value(False)   #turn off the LED
    time.sleep(1)    #wait for one second
```

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

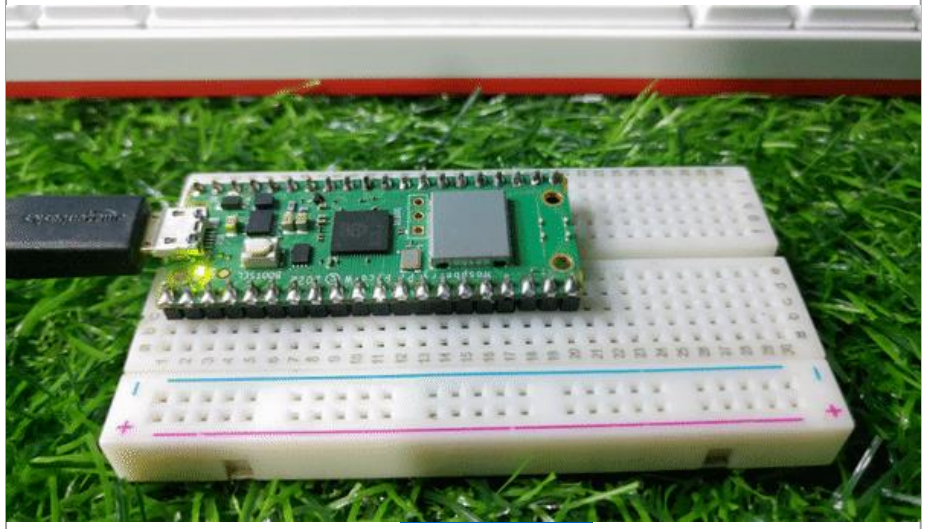


Image: https://diyprojectslab.com/raspberry-pi-pico-w-led-blink-micropython/

Key modules, classes and functions

- `Timer` **class:** A pin object is used to control digital I/O pins

- **Constructor**
  - `class machine.Timer(id, /, ...)`
- **Methods**:
  - `init()`: Initialize the timer.
    - `Timer.init(*, mode=Timer.PERIODIC, freq=-1, period=-1, callback=None)`
  - `deinit()`: De-initializes the timer. Stops the timer, and disables the timer peripheral.

Example: Blink the onboard LED
FRANKENSTEIN (I'm alive)

```
from machine import Pin, Timer

led = Pin("LED", Pin.OUT) # Initiate the LED pin
timer = Timer() # Create a timer object

def blink(timer):
   led.toggle() #

# Blink once a second (Initiate the timer)
timer.init(freq=1.0, mode=Timer.PERIODIC, callback=blink)
```

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

`Constants`

- Timer.ONE_SHOT
- Timer.PERIODIC

Key modules, classes and functions

- `Timer` **class:** A pin object is used to control digital I/O pins

- **Constructor**
  - `class machine.Timer(id, /, ...)`
- **Methods**:
  - `init()`: Initialize the timer.
    - `Timer.init(*, mode=Timer.PERIODIC, freq=-1, period=-1, callback=None)`
  - `deinit()`: De-initializes the timer. Stops the timer, and disables the timer peripheral.

Example: Period and frequency

```
# periodic at 1kHz
tim.init(mode=Timer.PERIODIC, freq=1000, callback=mycallback)

# periodic with 100ms period
tim.init(period=100, callback=mycallback)

# one shot firing after 1000ms
tim.init(mode=Timer.ONE_SHOT, period=1000, callback=mycallback)
```

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

`Constants`

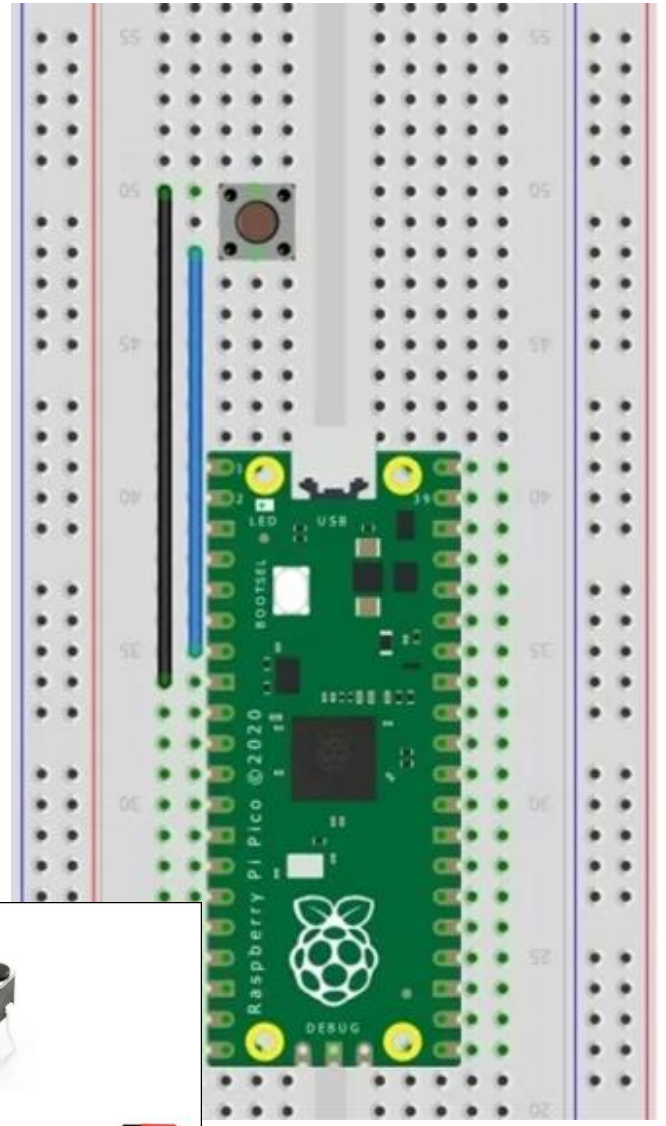
- Timer.ONE_SHOT
- Timer.PERIODIC

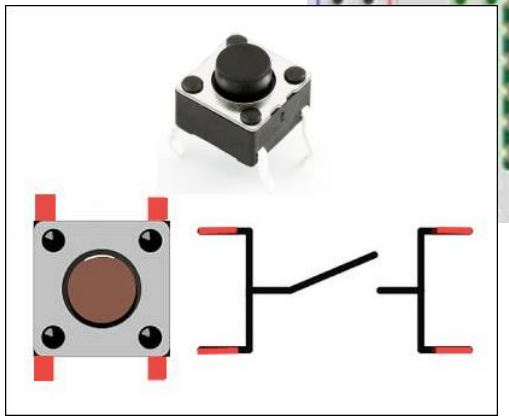
Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

Example: Button press (Input)

```
from machine import Pin

button = Pin(”GP5”, Pin.IN, Pin.PULL_UP)

while True:
    if not button.value():
        print('Button pressed!')
```



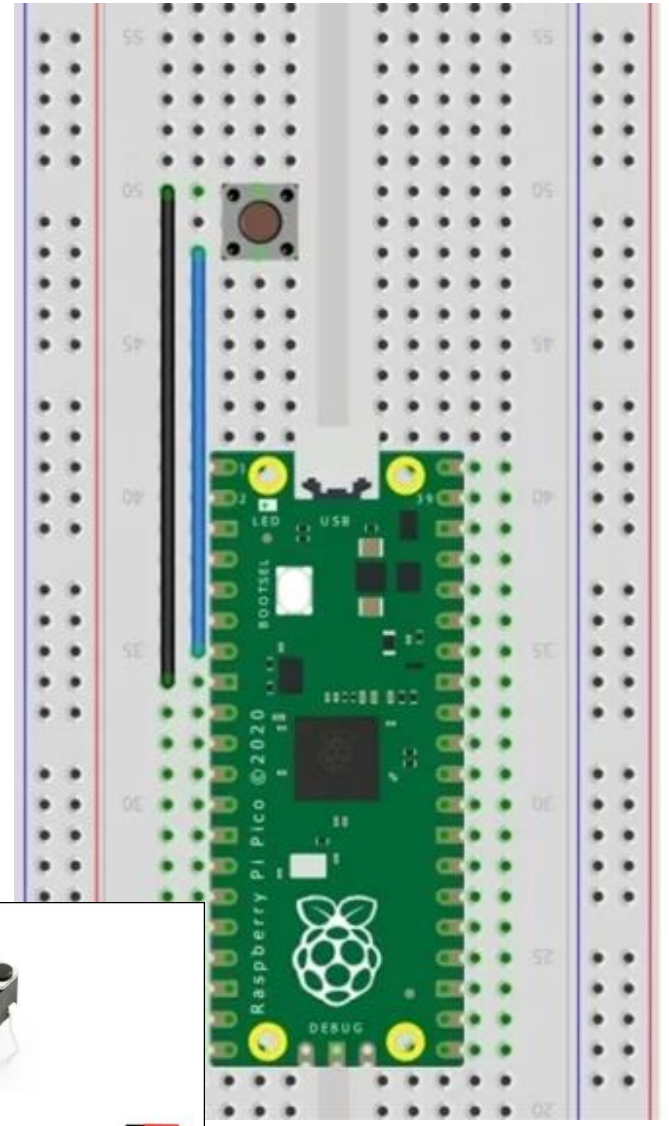


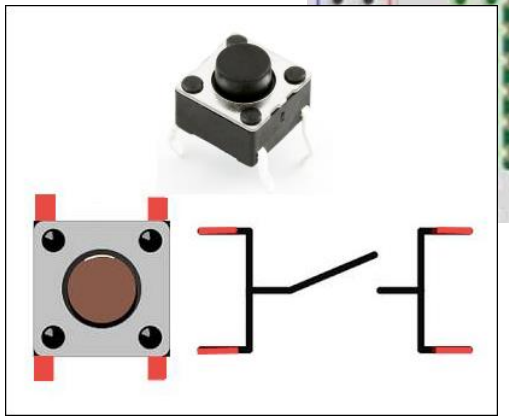
Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

Example: Button press with debounce (Input)

```
from machine import Pin
import time

button = Pin(”GP5”, Pin.IN, Pin.PULL_UP)

while True:
    first = button.value()
    time.sleep(0.01)
    second = button.value()
    if first and not second:
        print('Button pressed!') # Print once the button is pressed
    elif not first and second:
        print('Button released!') # ... and when it is released
```





Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

Interrupts

"a fundamental concept in microcontroller programming that allows your code to respond immediately to external events, such as a button press, without continuously polling the state of the input."

…so instead of **polling**

- …where the microcontroller checks the status of the input pin in a loop (as previous example)

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

`Constants`

- IRQ trigger type
  - Pin.IRQ_FALLING
  - Pin.IRQ_RISING
  - Pin.IRQ_LOW_LEVEL
  - Pin.IRQ_HIGH_LEVEL

- More available in the documentation

Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

Interrupts

"a fundamental concept in microcontroller programming that allows your code to respond immediately to external events, such as a button press, without continuously polling the state of the input."

…so instead of **polling**

- …where the microcontroller checks the status of the input pin in a loop (as previous example)

… it can **"interrupt"** its current operation to execute a function

- called an Interrupt Service Routine (ISR)

Classes (`machine` module)

- `class Pin` – control I/O pins
- `class Signal` – control and sense external I/O devices
- `class ADC` – analog to digital conversion
- `class ADCBlock` – control ADC peripherals
- `class PWM` – pulse width modulation
- `class UART` – duplex serial communication bus
- `class SPI` – a Serial Peripheral Interface bus protocol (controller side)
- `class I2C` – a two-wire serial protocol
- `class I2S` – Inter-IC Sound bus protocol
- `class RTC` – real time clock
- `class Timer` – control hardware timers
- `class WDT` – watchdog timer
- `class SD` – secure digital memory card (cc3200 port only)
- `class SDCard` – secure digital memory card
- `class USBDevice` – USB Device driver

`Constants`

- IRQ trigger type
  - Pin.IRQ_FALLING
  - Pin.IRQ_RISING
  - Pin.IRQ_LOW_LEVEL
  - Pin.IRQ_HIGH_LEVEL
- More available in the documentation

Key modules, classes and functions

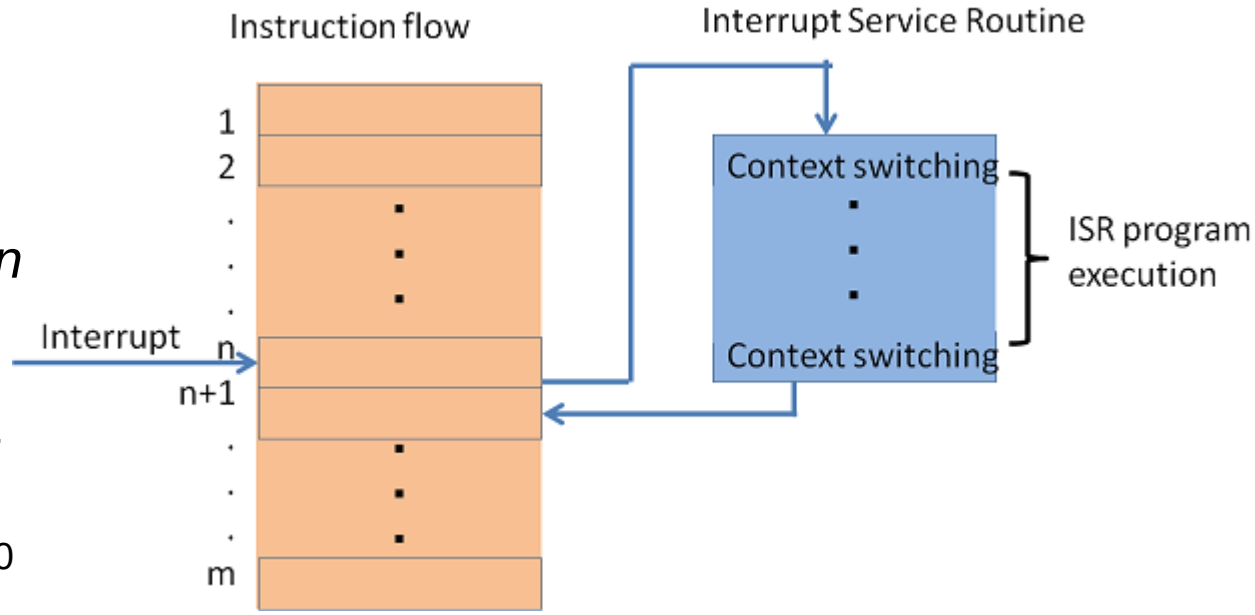
- `Pin` **class:** A pin object is used to control digital I/O pins

Interrupt Service Routine (ISR)

"A function that is automatically executed when an interrupt occurs."

An interrupt can be generated for every GPIO pin in four scenarios:

- **Edge Low:** the GPIO has transitioned from a logical 1 to a logical 0
  - Pin.IRQ_FALLING (*Edge Triggering*)
- **Edge High:** the GPIO has transitioned from a logical 0 to a logical 1
  - Pin.IRQ_RISING (*Edge Triggering*)
- **Level Low:** the GPIO pin is a logical 0
  - Pin.IRQ_LOW_LEVEL
- **Level High:** the GPIO pin is a logical 1
  - Pin.IRQ_HIGH_LEVEL



`Constants`

- IRQ trigger type
  - **Pin.IRQ_FALLING**
  - **Pin.IRQ_RISING**
  - **Pin.IRQ_LOW_LEVEL**
  - **Pin.IRQ_HIGH_LEVEL**
- More available in the documentation

Illustration: https://witscad.com/course/computer-architecture/chapter/cpu-interrupts-and-interrupt-handling

Key modules, classes and functions

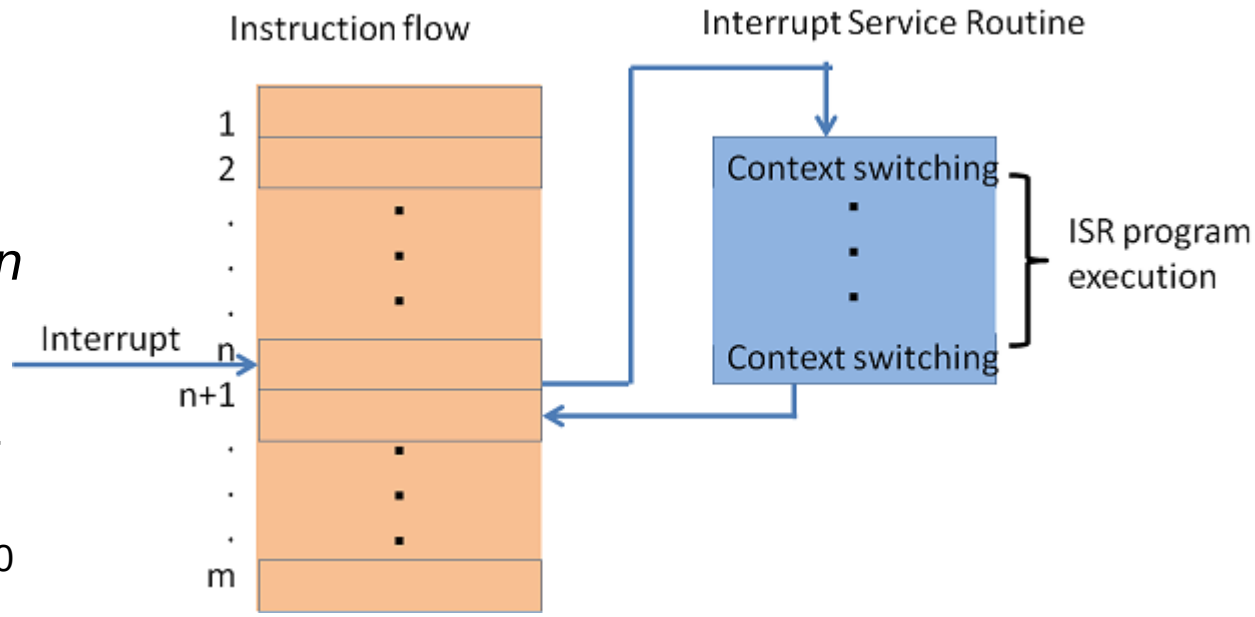
- `Pin` **class:** A pin object is used to control digital I/O pins

Interrupt Service Routine (ISR)

"A function that is automatically executed when an interrupt occurs."

An interrupt can be generated for every GPIO pin in four scenarios:

- **Edge Low:** the GPIO has transitioned from a logical 1 to a logical 0
  - Pin.IRQ_FALLING (*Edge Triggering*)
- **Edge High:** the GPIO has transitioned from a logical 0 to a logical 1
  - Pin.IRQ_RISING (*Edge Triggering*)
- ~~**Level Low:** the GPIO pin is a logical 0~~
  - ~~Pin.IRQ_LOW_LEVEL~~
- ~~**Level High:** the GPIO pin is a logical 1~~
  - ~~Pin.IRQ_HIGH_LEVEL~~



`Constants`

- IRQ trigger type
  - **Pin.IRQ_FALLING**
  - **Pin.IRQ_RISING**
  - **Pin.IRQ_LOW_LEVEL**
  - **Pin.IRQ_HIGH_LEVEL**
- More available in the documentation

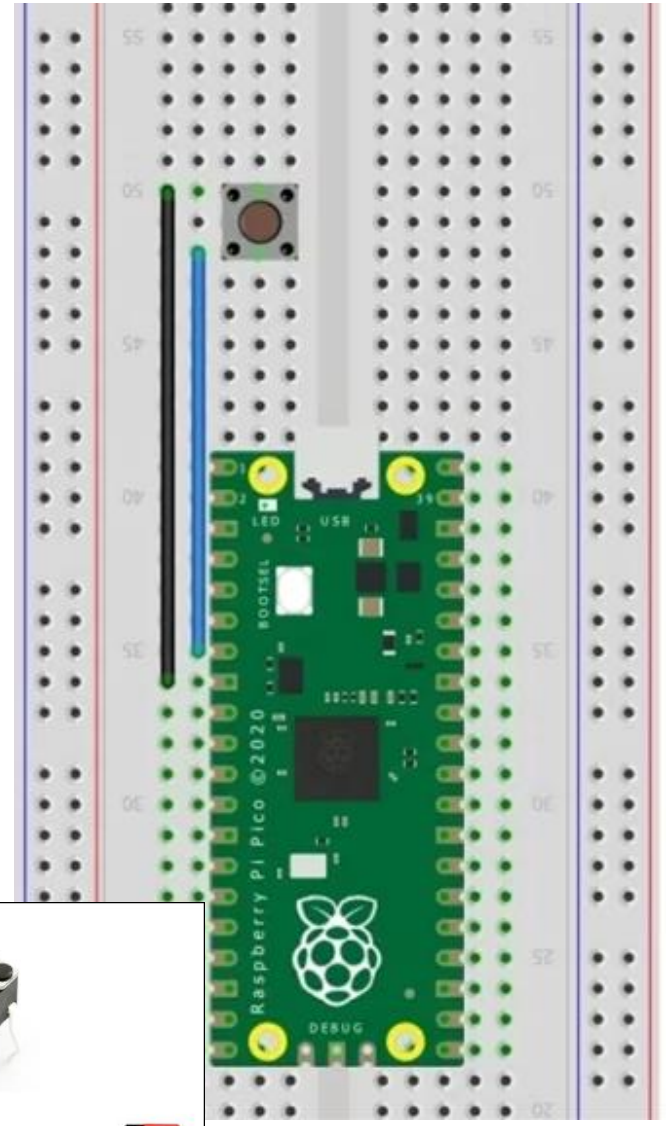
Illustration: https://witscad.com/course/computer-architecture/chapter/cpu-interrupts-and-interrupt-handling

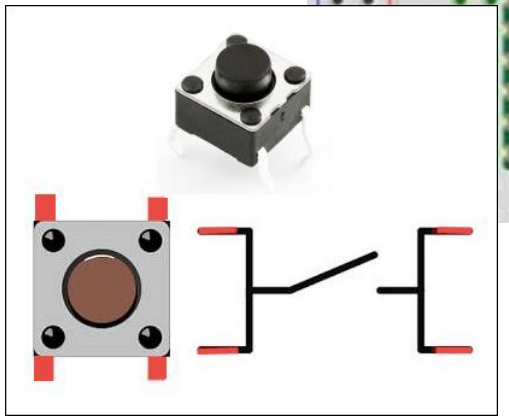
Key modules, classes and functions

- `Pin` **class:** A pin object is used to control digital I/O pins

Example: Button press using Interrupts

```
from machine import Pin

# Initialize the button pin as input with an internal pull-up resistor
button = Pin('GP5', Pin.IN, Pin.PULL_UP)

# Define the ISR for button press
def handle_button_interrupt(pin):
    if pin.value() == 0:  # Button pressed (active low)
        print('Button pressed!')
    else:  # Button released
        print('Button released!')

# Attach the interrupt to the button pin
# Trigger on both falling and rising edges to detect both press and release
button.irq(trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING,
           handler=handle_button_interrupt)

# Main loop does nothing; all action is handled by the interrupt
while True:
    pass  # Keep the program running; the ISR handles button events
```





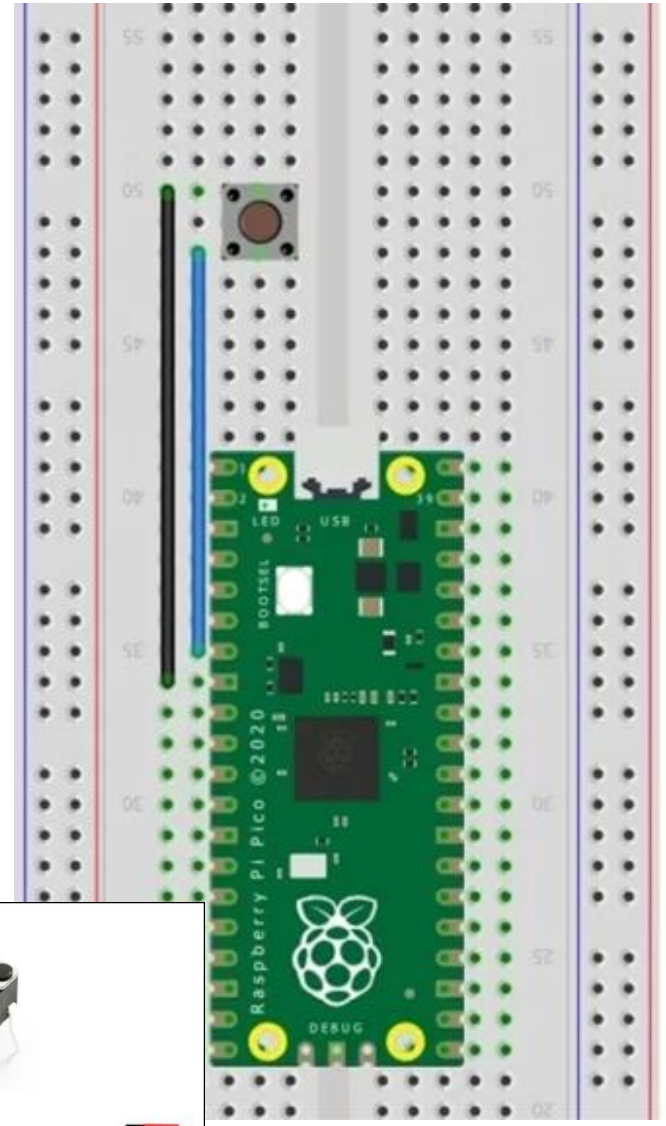
Key modules, classes and functions

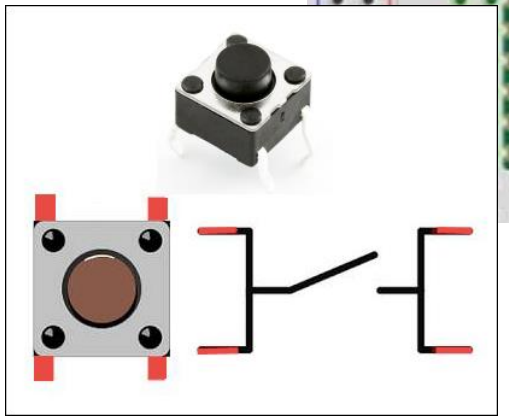
- `Pin` **class:** A pin object is used to control digital I/O pins

Example: Button press using Interrupts

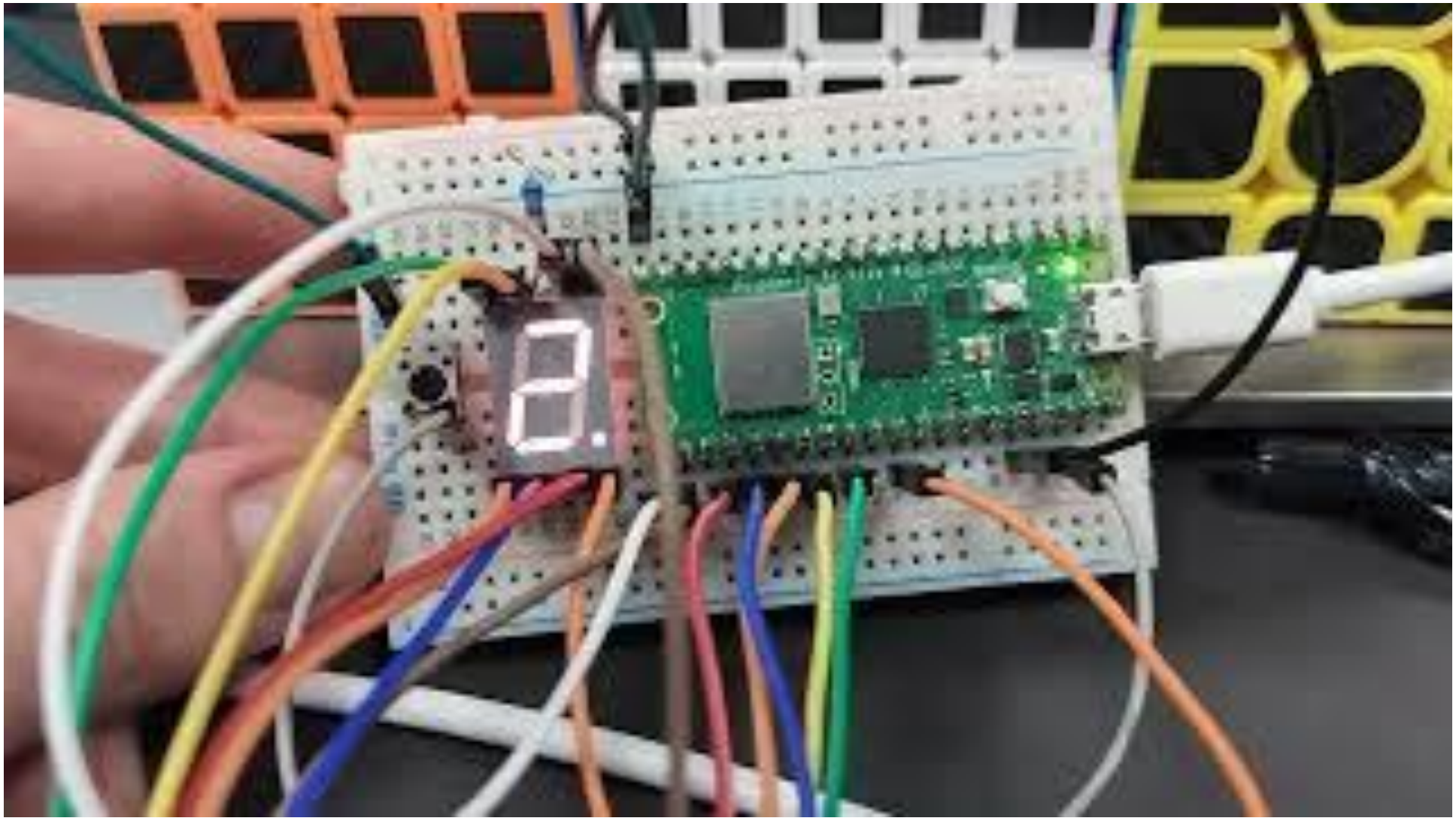
```
from machine import Pin

# I
but

# D
def

# A
# Trigger on both falling and rising edges to detect both press and release
button.irq(trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING,
handler=handle_button_interrupt)

# Main loop does nothing; all action is handled by the interrupt
while True:
    pass  # Keep the program running; the ISR handles button events
```

What if only **Pin.IRQ_RISING?**
…or only **Pin.IRQ_FALLING?**





Portfolio 1: 7-Segment Display controller



Portfolio 1: 7-Segment Display controller

- Hand-in via **https://gitlab.sdu.dk/**
  - Use the following template: https://gitlab.sdu.dk/mobin16/portfolio_template

Assignments