Using DHT11 sensor

November 14, 2024

1 Introduction

In this tutorial you will find the instructions to assembly a sensor device as well as the implementation code for it. In order to follow this tutorial, first you need to know the required steps to setup the system that are described in the previous tutorial, which title is "Programming Raspberry Pi Pico with Arduino IDE" (available at moodle).

2 Assembling

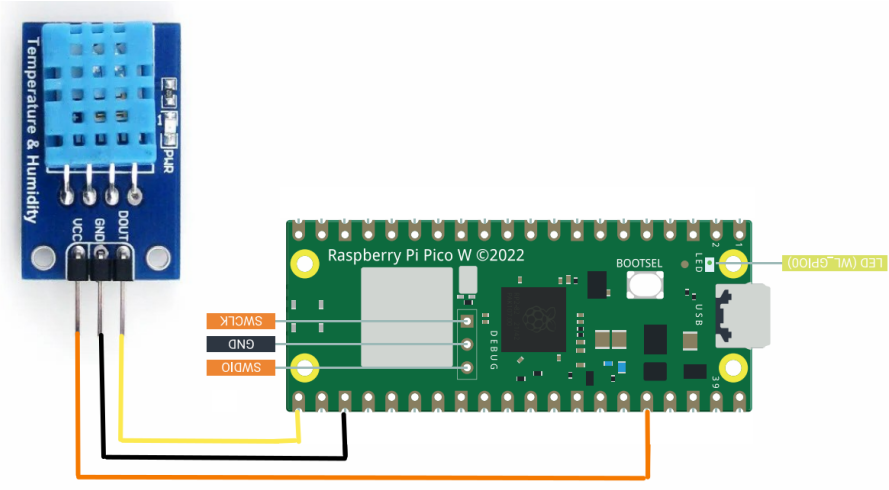
This device is composed by two hardware components:

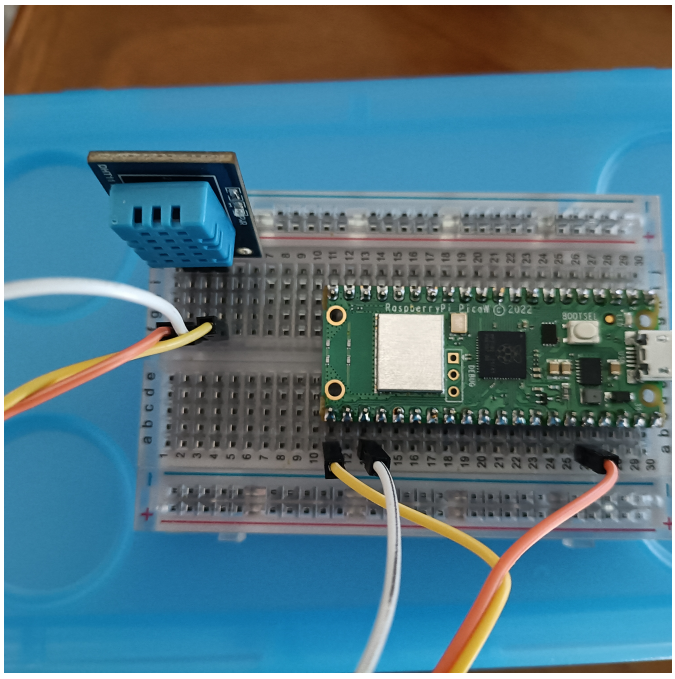
- DHT11 sensor;
- Raspberry Pi Pico W.

The figure below shows the connection scheme of the device: the Voltage (`VCC`) and `GND` pins of the DHT11 Temperature and Humidity Sensor are connected with the 3.3V and GND pins of the Raspberry Pi Pico W. While the `DOUT` pin is connected with the GPIO pin 16.

Attention

There are many types of DHT 11 sensors with connection pins in different positions. So, pay attention to the pins for connecting them correctly.



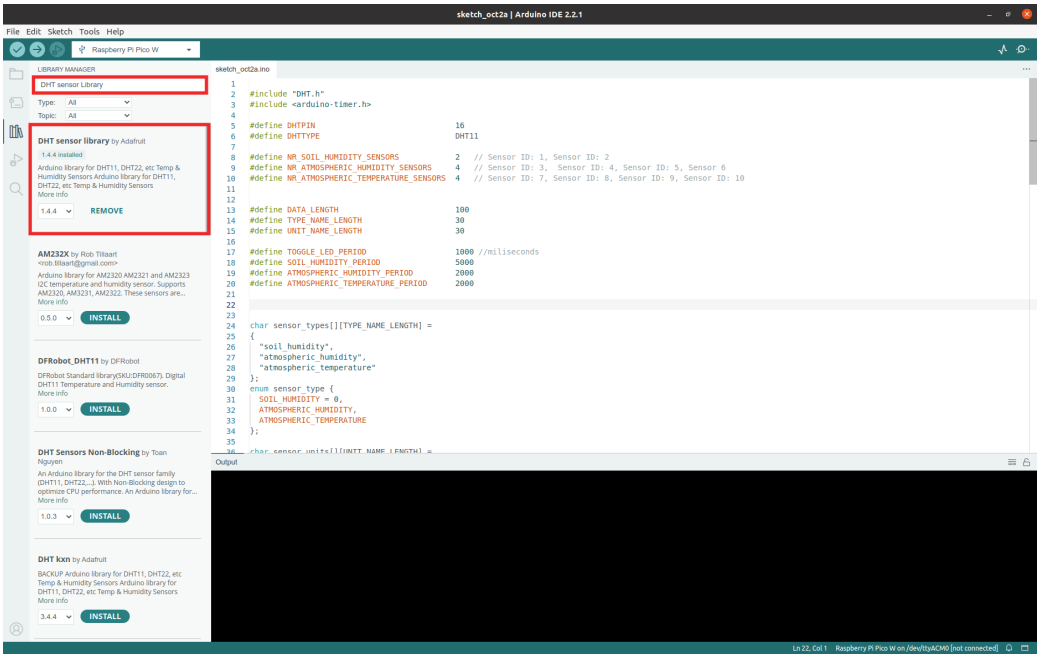


A complete description of Raspberry Pi Pico W pinout can be found at Section 6.

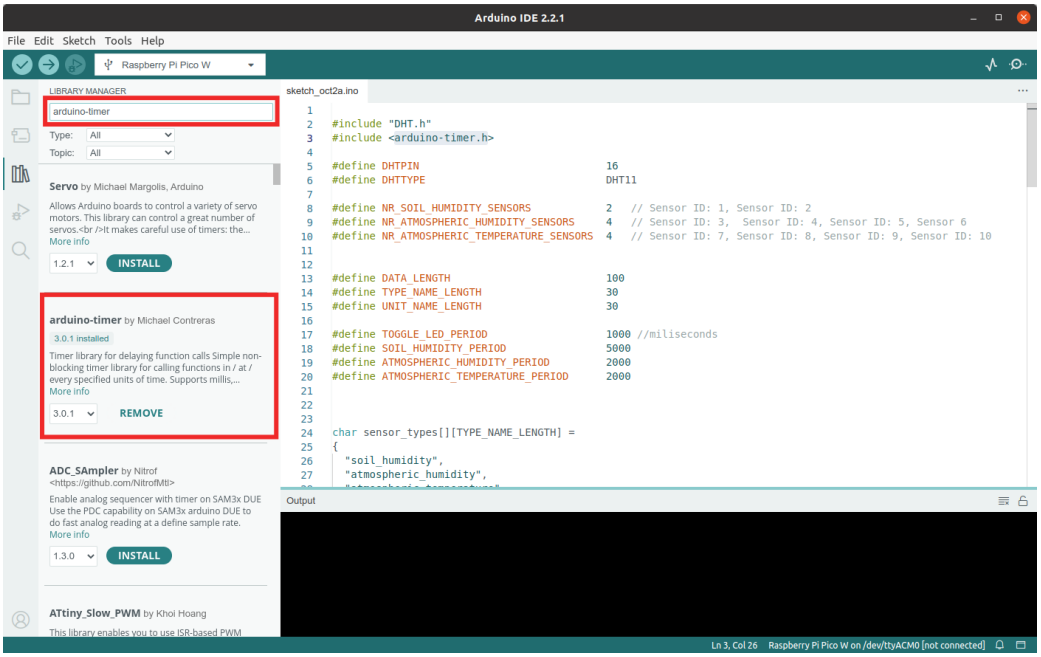
3 Coding

In this section you can find the implementation code of the sensor device. It requires two libraries:

1. `DHT sensor library` by Adafruit



2. `arduino-timer` by Michael Contreras.



You can find instruction to install a library in Section 5.

The sensor device sketch code is as follows. It blinks the builtin led at every one second (1000 milisecond) and it simulates 10 sensors (two soil humidity, four atmospheric humidity, and four atmospheric temperature). Each simulated sensor has an identifier and a period. After each sensor read, the device sends a string formatted as follows:

```
"sensor_id:"+id+"#type:"+type+"#value:"+value+"#unit:"+unit+"#time:"+
    time
```

where:

- `id`: is the sensor identifier
- `type`: sensor type
- `value`: sensor value
- `unit`: sensor value unit
- `time`: time in milliseconds (number of milliseconds from current device boot)

Here you have an example:
sensor_id:8#type:atmospheric_temperature#value:21.60#unit:celsius#time:2470030

```
#include "DHT.h"
#include <arduino-timer.h>

#define DHTPIN                          16
#define DHTTYPE                         DHT11

#define NR_SOIL_HUMIDITY_SENSORS        2   // Sensor ID: 1, Sensor ID: 2
#define NR_ATMOSPHERIC_HUMIDITY_SENSORS 4   // Sensor ID: 3, Sensor ID:
    4, Sensor ID: 5, Sensor 6
#define NR_ATMOSPHERIC_TEMPERATURE_SENSORS 4 // Sensor ID: 7, Sensor ID:
    8, Sensor ID: 9, Sensor ID: 10


#define DATA_LENGTH                     100
#define TYPE_NAME_LENGTH                30
#define UNIT_NAME_LENGTH                30

#define TOGGLE_LED_PERIOD               1000 //miliseconds
#define SOIL_HUMIDITY_PERIOD            5000
#define ATMOSPHERIC_HUMIDITY_PERIOD     2000
#define ATMOSPHERIC_TEMPERATURE_PERIOD  2000



char sensor_types[][TYPE_NAME_LENGTH] =
{
   "soil_humidity",
   "atmospheric_humidity",
   "atmospheric_temperature"
};
enum sensor_type {
   SOIL_HUMIDITY = 0,
   ATMOSPHERIC_HUMIDITY,
   ATMOSPHERIC_TEMPERATURE
};

char sensor_units[][UNIT_NAME_LENGTH] =
{
   "percentage",
   "celsius",
};
```

```
enum sensor_unit {
   PERCENTAGE = 0,
   CELSIUS,
};


DHT dht(DHTPIN, DHTTYPE);

auto timer = timer_create_default(); // create a timer with default
    settings

bool toggle_led(void *) {
   digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN)); // toggle the LED
   return true; // keep timer active? true
}

int get_sensor_id(int min, int max){
   int id = random(min, max); //generate a random value [min, max[
   return id;
}

bool read_soil_humidity(void *){

   int min = 1;
   int max = min + NR_SOIL_HUMIDITY_SENSORS;
   int id = get_sensor_id(min, max);

   float r = dht.readHumidity();
   if (!isnan(r)) {
      send_data(id,SOIL_HUMIDITY, r, PERCENTAGE, millis());
   }
   return true;
}


bool read_atmospheric_humidity(void *){

   int min = NR_SOIL_HUMIDITY_SENSORS +1;
   int max = min + NR_ATMOSPHERIC_HUMIDITY_SENSORS;
   int id = get_sensor_id(min, max);

   float r = dht.readHumidity();
   if (!isnan(r)) {
```

```
      send_data(id,ATMOSPHERIC_HUMIDITY, r, PERCENTAGE, millis());
   }
   return true;
}

bool read_atmospheric_temperature(void *){
   int min = NR_SOIL_HUMIDITY_SENSORS + NR_ATMOSPHERIC_HUMIDITY_SENSORS +
       1;
   int max = min + NR_ATMOSPHERIC_TEMPERATURE_SENSORS;
   int id = get_sensor_id(min, max);

   float r = dht.readTemperature();
   if (!isnan(r)) {
      send_data(id,ATMOSPHERIC_TEMPERATURE, r, CELSIUS, millis());
   }

   return true;
}

void send_data(int id, sensor_type t, float r, sensor_unit u, unsigned
    long m){
   char buffer[DATA_LENGTH];
   snprintf(buffer, DATA_LENGTH- 1,
       "sensor_id:%d#type:%s#value:%.2f#unit:%s#time:%lu",
       id,sensor_types[t],r, sensor_units[u], m);
   Serial.println(buffer);
}

// the setup function runs once when you press reset or power the board
void setup() {
   // initialize digital pin LED_BUILTIN as an output.
   pinMode(LED_BUILTIN, OUTPUT);
   randomSeed(analogRead(0));
   Serial.begin(9600);
   dht.begin();
   timer.every(TOGGLE_LED_PERIOD, toggle_led);
   timer.every(ATMOSPHERIC_TEMPERATURE_PERIOD,
       read_atmospheric_temperature);
   timer.every(ATMOSPHERIC_HUMIDITY_PERIOD, read_atmospheric_humidity);
   timer.every(SOIL_HUMIDITY_PERIOD, read_soil_humidity);
}

// the loop function runs over and over again forever
```

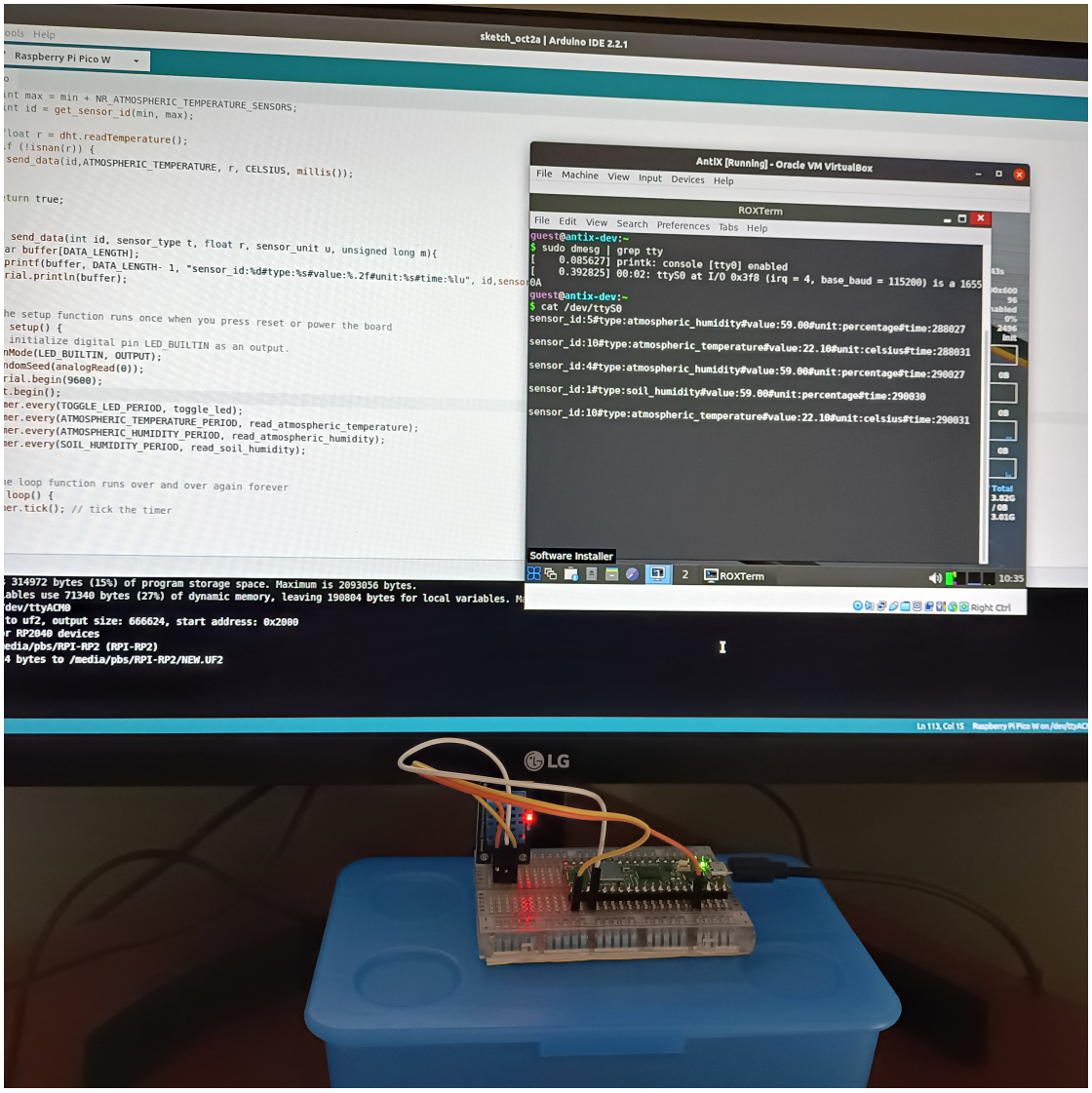
```
void loop() {
   timer.tick(); // tick the timer

}
```

4 Connect to AntiX ARQCP Virtual Machine

First compile and upload the sketch.

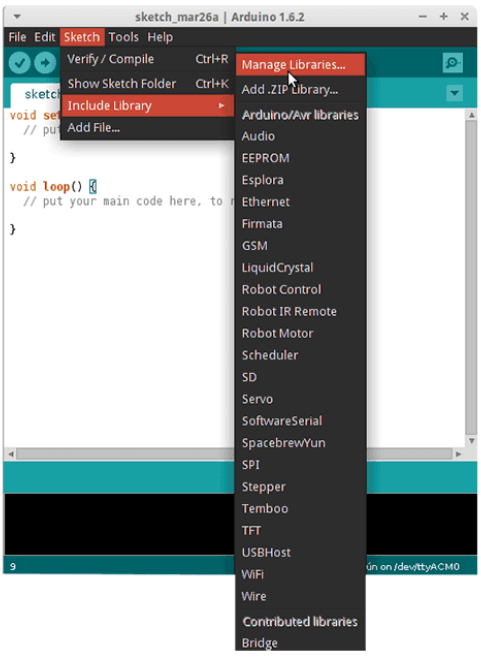
Then connect the device to AntiX ARQCP Virtual Machine. In AntiX ARQCP Virtual Machine open a Terminal and get the messages sent by the device.



5 Installing Libraries

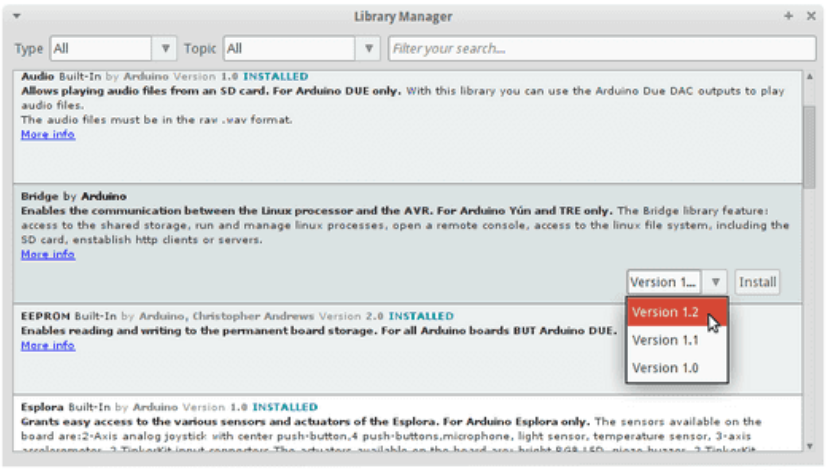
To install a new library into your Arduino IDE you can use the Library Manager[1].

1. In the Arduino IDE, click on `Sketch >Include Library > Manage Libraries`.

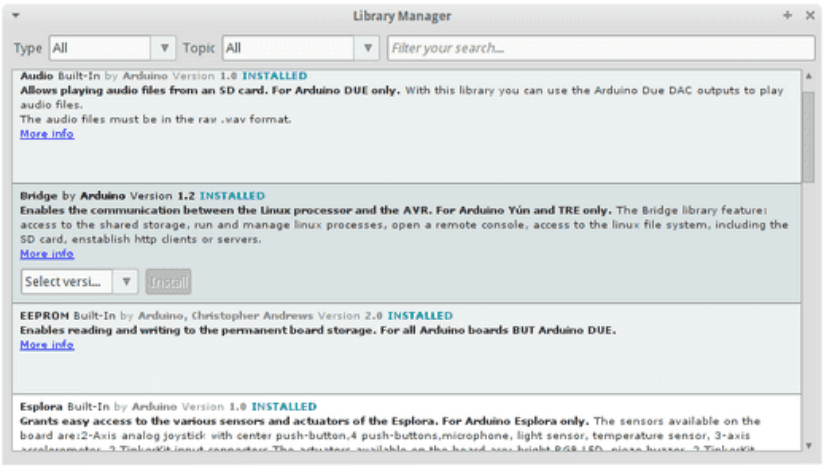


2. Then the Library Manager will open and you will find a list of libraries that are already installed or ready for installation. In this example we will install the Bridge library. Scroll the list to find it, click on it, then select the version of the library you want to install. Sometimes only one version of the library is available. If the version selection menu does not appear, don't worry: it is normal.

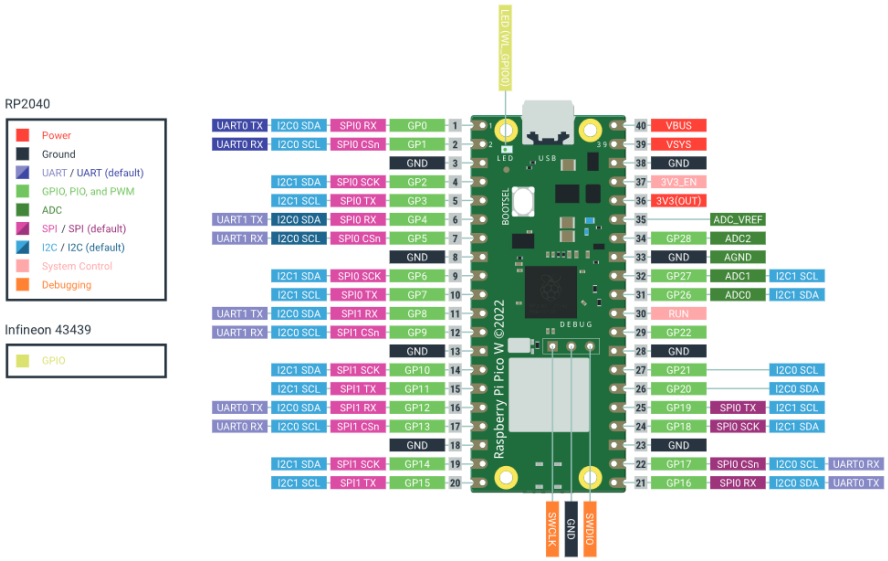
[1] `https://docs.arduino.cc/software/ide-v1/tutorials/installing-libraries`



3. Finally click on install and wait for the IDE to install the new library. Downloading may take time depending on your connection speed. Once it has finished, an Installed tag should appear next to the Bridge library. You can close the library manager.



6 Raspberry Pi Pico W Pinout



The Raspberry Pi[2]. company expanded the Pico lineup with the introduction of the Raspberry Pi Pico W in June 2022.

The "W" stands for wireless, and this new iteration incorporates Infineon's CYW43439 chip, enabling the board to provide built-in 2.4 GHz Wi-Fi connectivity through an onboard antenna. It also supports Bluetooth connectivity. Although the pinout diagram may appear intricate at first glance, it can actually be simplified into distinct and easy-to-remember blocks. We have power, PWM, ADC, GPIO, communication, and debugging pins.

6.1 Power Pins

The Raspberry Pi Pico has several power pins, including the VBUS, VSYS, and 3V3. The VBUS pin is used for powering the Pico via USB and is connected to micro-USB port pin 1, while the VSYS pin allows for an external power supply to be connected to provide power to the board.

The 3V3 pin provides a regulated 3.3V power output, which can be used to power external components.

There are other power pins present on the board that can be used for special cases, as specified below:

Pin	Description
ADC_VREF	ADC pin power supply voltage, filtered from the 3.3V supply on the board. (Pin 35)
AGND	Ground reference for GPIO26-29, connected to a separate analog ground plane. Can be linked to digital ground. (Pin 33)
3V3_EN	Connects to the onboard SMPS enable pin. High (to VSYS) with a 100kΩ resistor. Short it to disable 3.3V.
GND	Ground pins.
RUN	RP2040 enable pin with an internal pull-up resistor ($\sim$ 50kΩ) to 3.3V. Short this pin low to reset RP2040.

[2] `https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html` and `https://www.makeuseof.com/raspberry-pi-pico-pinout-explained-everything-you-need-to-know/`

6.2 GPIO Pins

Out of the 40 pins, 26 of them are GPIO (General-Purpose Input/Output) pins. Labeled from GP0 to GP28, these pins can handle both digital input and output operations. Four of these GPIO pins, GP23, GP24, GP25, and GP29, are not exposed on the header. Instead, they are dedicated to internal board functions.

Pin	Functionality	Description
GPIO29	ADC mode (ADC3) for measuring VSYS/3	Monitors voltage levels
GPIO25	Connected to user LED	Allows control over LED output
GPIO24	Indicator for VBUS presence	Goes high when VBUS is present, low otherwise
GPIO23	Controls on-board SMPS Power Save functionality	Acts as a convenient power switch

6.3 Analog Pins

The Pico board has four dedicated analog pins boasting a 12-bit ADC (analog-to-digital converter).

- ADC0: Mapped to GP26.
- ADC1: Mapped to GP27.
- ADC2: Mapped to GP28.

The board also has eight PWM (pulse-width modulation) blocks numbered from 1 to 8, each having two PWM outputs that it can drive simultaneously. In short, you have access to 16 PWM output channels that can be used at any given time.

It's important to note that two GPIO pins sharing the same PWM designation cannot be used simultaneously.

6.4 Communication Pins

For communication with devices, the Pi Pico board relies on specific pins. Now, what's noteworthy is that Raspberry Pi Pico generously offers all 26 general-purpose pins for SCL, SDA, TX, and RX.

6.4.1 SPI

There are two SPI interfaces available for communication: SPI0 and SPI1.

SPI Controller	RX (GPIO Pins)	TX (GPIO Pins)	CLK (GPIO Pins)	CSn (GPIO Pins)
SPI0	GP0/GP4/GP16 (Pin 1/6/24)	GP3/GP7/GP19 (Pin 4/9/37)	GP2/GP6/GP18 (Pin 3/8/35)	GP1/GP5/GP17 (Pin 2/7/37)
SPI1	GP8/GP12 (Pin 12/16)	GP11/GP15 (Pin 15/19)	GP10/GP14 (Pin 14/18)	GP9/GP13 (Pin 13/17)

6.4.2 I2C

There are two SPI interfaces available for communication: SPI0 and SPI1.

I2C Controller	SDA (GPIO Pins)	SCL (GPIO Pins)
I2C0	GP0/GP4/GP8/GP12/GP16/GP20 (Pin 1/6/12/16/24/38)	GP1/GP5/GP9/GP13/GP17/GP21 (Pin 2/7/13/17/25/40)
I2C1	GP2/GP6/GP10/GP14/GP18/GP26 (Pin 3/8/14/18/35/37)	GP3/GP7/GP11/GP15/GP19/GP27 (Pin 4/9/15/19/37/39)

6.4.3 UART

The Pi Pico board has two UART interfaces with pins, as shown in the table below:

UART	TX (GPIO Pins)	RX (GPIO Pins)
UART0	GP0/GP12/GP16 (Pin 1/12/24)	GP1/GP13/GP17 (Pin 2/13/25)
UART1	GP4/GP8 (Pin 6/12)	GP5/GP9 (Pin 7/13)

6.5 Debugging Pins

The Raspberry Pi Pico board has three dedicated debugging pins that can be used for troubleshooting and debugging purposes.

- SWD GND (Serial Wire Debug): This pin acts as the ground pin for the two-wire interface.
- SWCLK (Serial Wire Clock): This pin is associated with the SWD interface and provides the clock signal for synchronized communication during debugging.
- SWDIO (Serial Wire Debug I/O): This bidirectional pin is also part of the SWD interface and carries both control and data signals during debugging.

These pins provide direct access to important signals and interfaces on the Pico board, allowing you to monitor and analyze the system's behavior during the debugging process—this can be made easier by using a Raspberry Pi Debug Probe.

6.6 The PIO Feature

The PIO (Programmable Input/Output) feature in the Pi Pico is a special hardware block that allows the Pi Pico to perform custom digital signal processing and control tasks. It's like having an extra dedicated processor inside the Pi Pico that can handle complex tasks quickly and efficiently, freeing up the main CPU.

The PIO can be programmed to handle various tasks such as generating precise timing signals, reading and writing data to external devices, and even implementing simple algorithms. It can also be used to create custom interfaces for connecting devices (in addition to the standard I2C, SPI, and UART protocols).