EXPLORER SET

RB-P-XPLR-SET

TABLE OF CONTENTS

1. GENERAL INFORMATION

Dear customer, thank you for choosing our product. In the following, we will show you what you need to bear in mind during commissioning and use.

Should you encounter any unexpected problems during use, please do not hesitate to contact us.

2. DEVICE OVERVIEW & PIN ASSIGNMENT

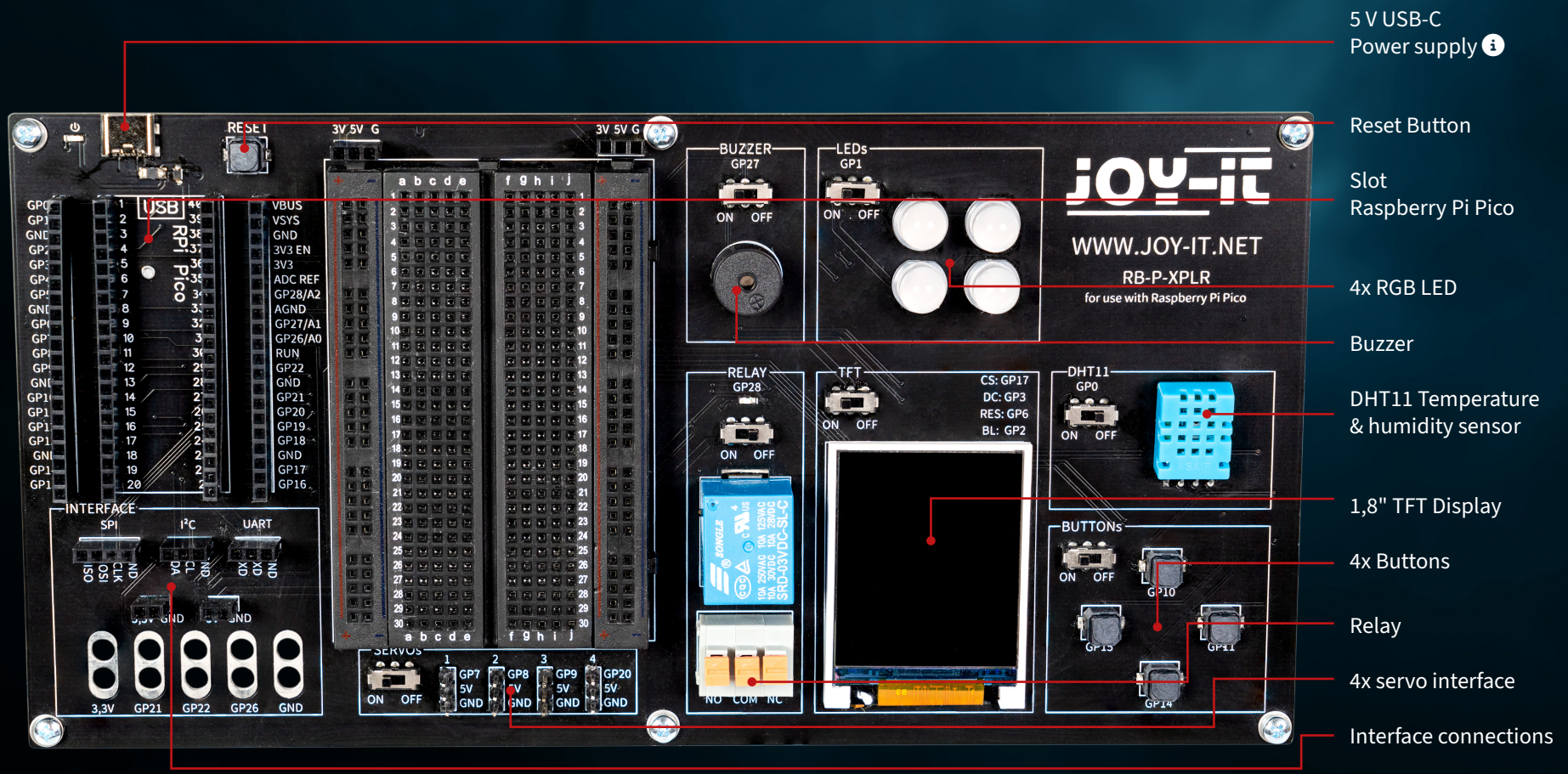
Our Explorer Board is the simple and efficient way to develop your Raspberry Pi Pico projects.

With the most important components already integrated, you save time and effort when wiring. The Explorer Board has a wide range of interface connectors so you can connect your projects to a variety of modules and devices. With the integrated breadboard, you can quickly build and realize your own projects.

Thanks to the option of switching all modules on or off individually, you can use your pins, which are also routed separately to the outside, for other projects or experiment on the integrated breadboard at any time.

All built-in components can be switched off via the respective switch if they are not required. This means that the associated pins can also be used for other components if necessary.

To the left and right of the Raspberry Pi Pico, all pins are additionally designed. Components can be connected directly here or routed to the integrated breadboard via additional cables.

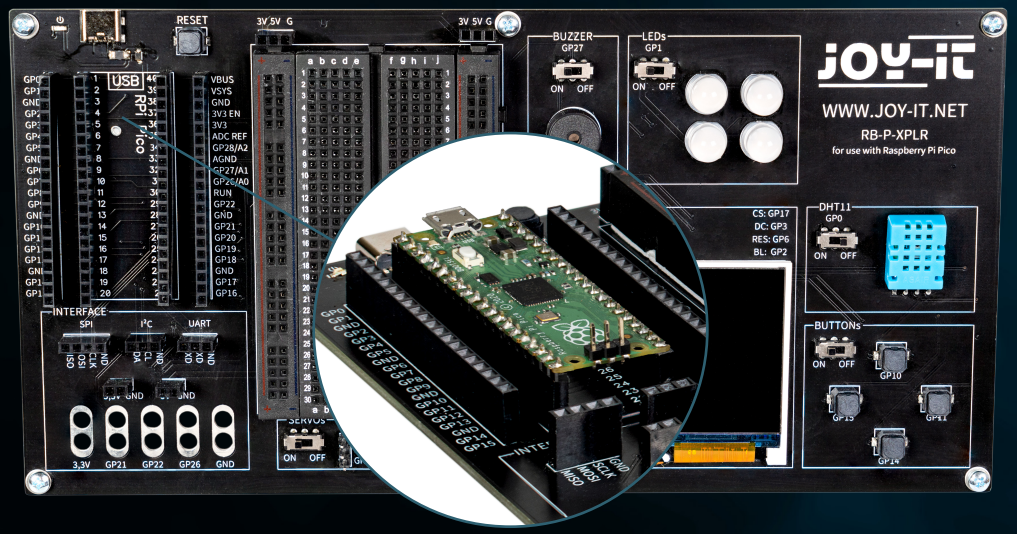


Please note that the USB-C connection must always be connected for use. A power supply via the micro USB connection of the Raspberry Pi Pico is not possible.

PIN ASSIGNMENT	
Buzzer	GP27
LEDs	GP1
Relay	GP28
1,8" TFT Display	CS: GP17, DC: GP3, RES: GP6, BL: GP2
DHT11	GP0
Buttons	GP10, GP11, GP14 & GP15
Servos	GP7, GP8, GP9 & GP20
UART	RXD: GP13, TXD: GP12
I2C	SDA: GP4, SCL: GP5
SPI	MISO: GP16, MOSI: GP19, SCLK: GP18

3. RASPBERRY PI PICO

First plug your Raspberry Pi Pico into the slot on your board.



Now connect a micro USB cable to your computer and to the Raspberry Pi Pico for programming.

ATTENTION! The USB-C port on the Explorer board is used exclusively for power supply. It is not used to transfer data to the Raspberry Pi.
You can use a suitable development program of your choice to transfer our example program. We recommend the **Thonny Python IDE**.

ATTENTION! If you are new to the world of microcontrollers and electronics, don't worry! We have prepared a special beginner's guide for you. This guide is specially tailored to the needs of beginners and explains how to use the Raspberry Pi Pico step by step.

From basic configuration to running projects, in this guide we'll walk you through the entire process. Our guide includes easy-to-understand explanations and useful tips to help you quickly and effectively develop your skills at scale with the Raspberry Pi Pico. You can download our guide **here**.

4. MODULES IN DETAIL

In the following, all modules available on the Explorer Board are explained individually with sample codes. Here you can download all sample codes and libraries, as well as a sample code that links all modules together.

For the use of some modules, external libraries and a font file are used. Download the libraries and load them into the lib folder of your Raspberry Pi Pico. Place the font file in the root directory of your Raspberry Pi Pico.

4.1 BUZZER

A buzzer produces a signal tone, similar to a loudspeaker. Unlike a loudspeaker, however, it is only suitable for a limited frequency range, so it does not produce a good sound for reproducing music or speech. However, it is ideal for generating loud warning tones in the form of beeps. Whenever an electrical device generates a warning tone, it is almost always a buzzer. For example, in alarm clocks, smoke detectors or the seatbelt reminder in cars.

The buzzer is connected to GPIO pin GP27.

```
# Load libraries
from machine import Pin, PWM

buzzerPin = Pin(27)
buzzer = PWM(buzzerPin)

while True:
    # Activate buzzer for 1 sec
    buzzer.freq(1000)
    buzzer.duty_u16(1000)
    sleep(1)
    buzzer.duty_u16(0)
    sleep(1)
```

4.2 RGB LEDS

RGB LEDs are a type of light-emitting diode that combines red, green and blue to produce a variety of colors. Much like a buzzer only produces simple tones, RGB LEDs cannot display complex images, but they are excellent at mixing and varying colors. Each LED in an RGB unit can be varied in intensity to produce different hues, from soft pastels to bright, saturated colors. This makes them ideal for mood lighting, decorative lighting and in applications where visual signals are required, such as in gaming setups or as status indicators in electronic devices. Their versatility and energy efficiency have made them a popular choice in modern lighting systems, although, like the buzzer, their simple operation means they cannot create complex images or patterns without additional control units.

The GPIO LEDs are connected to the GPIO pin GP1.

```
# Load libraries
from machine import Pin, PWM
from utime import sleep
from neopixel import NeoPixel

ledPin = 1
ledCount = 4

# Initialize GPIOs
led = Pin(ledPin, Pin.OUT)
led = NeoPixel(Pin(ledPin, Pin.OUT), ledCount)

while True:
    # Turn LEDs white
    for i in range (ledCount):
        led[i] = (255, 255, 255)
    led.write()
    sleep(1)
    # Turn LEDs red
    for i in range (ledCount):
        led[i] = (255, 0, 0)
    led.write()
    sleep(1)
    # Turn LEDs blue
    for i in range (ledCount):
        led[i] = (0, 0, 255)
    led.write()
    sleep(1)
    # Turn LEDs green
    for i in range (ledCount):
        led[i] = (0, 255, 0)
    led.write()
    sleep(1)
```

4.3 RELAY

Relays are some of the oldest electromechanical components and function as electrically controlled switches. With a small input voltage and low current, a large electrical load can be switched on and off at the output. When the relay switches through, the red LED also lights up. You can insert stripped cable ends into the terminal socket (by pressing down the orange lever) to use the three connections.

The relay is connected to GPIO pin GP28.

```
# Load libraries
from machine import Pin, PWM
from utime import sleep

relayPin = 28
# Initialize GPIOs
relay = Pin(relayPin, Pin.OUT)

while True:
    # Toggle Relay
    relay.on()
    sleep(1)
    relay.off()
    sleep(1)
```

4.4 TFT

The liquid crystal display (LCD TFT) with around 65,000 colors and a diagonal of 1.8 inches has a resolution of 128×160 pixels and can be controlled via SPI. It is suitable for displaying colorful graphics and images. Letters and other characters are displayed as graphics made up of many individual dots.

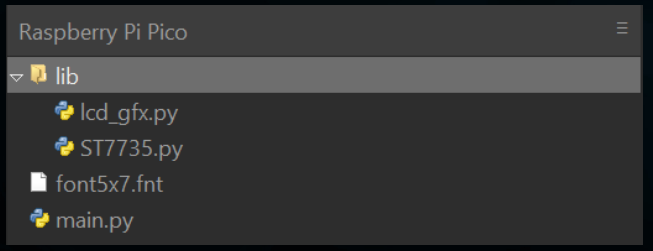
The TFT is connected to the GPIO pins GP17 (CS), GP3 (DC), GP6 (RES) and GP2 (BL).

```
from machine import Pin, SPI
import ST7735

# Initialize LCD
spi = SPI(0, baudrate=8000000, polarity=0, phase=0, sck=Pin(18), mosi=Pin(19),
miso=Pin(16))
lcd = ST7735.ST7735(spi, rst=6, ce=17, dc=3)
backlight = Pin(2, Pin.OUT)

# Turn backlight on
backlight.high()
lcd.reset()
lcd.begin()

# Display content on the LCD
lcd.fill_screen(lcd.rgb_to_565(0, 255, 0))  # Fills the screen with a green color

# Display text
lcd.p_string(20, 50, ‚Hello, World!‘)
```

In addition to texts, rectangles can also be displayed, for example:

```
# Draw red rectangle
lcd.draw_block(10, 10, 50, 50, lcd.rgb_to_565(255, 0, 0))
```

ATTENTION! Two separate library files and a font file are required for the TFT display; you can download the required files here. Then transfer all files from the Libraries folder to the root directory of your Raspberry Pi Pico so that the folder structure looks like this:



4.5 DHT 11

The DHT11 sensor can detect temperatures from 0 °C to 50 °C (±2 °C accuracy) and relative humidity from 20 % to 80 % (±5 %) (at most once per second). Weather stations are probably the primary area of application for a sensor such as the DHT11. To test the functionality, it is sufficient to hold your mouth close to the sensor and exhale slowly. The breathing air differs from the environment in terms of temperature and humidity, which should lead to a significant change in the values.

The DHT11 is connected to the GPIO pin GP0.

```
from machine import Pin
from dht import DHT11
from utime import sleep

# Initialize DHT11 Sensor
dhtPin = 0
dht = DHT11(Pin(dhtPin, Pin.IN))

while True:
    # Measure DHT11 values
    dht.measure()
    temp = dht.temperature()  # Temperature in Celsius
    humid = dht.humidity()    # Relative Humidity in %

    # Print the measurements
    print('Temperature:', temp, '°C')
    print('Humidity:', humid, '%')

    sleep(2)  # Wait for 2 seconds before the next reading
```

4.6 BUTTONS

Buttons are interactive elements in user interfaces that fulfill a simple but essential function: user input. Similar to how RGB LEDs can display a variety of colors, buttons are used to initiate a wide range of commands and actions in digital environments.

The buttons are connected to the GPIO pins GP10 (top), GP11 (right), GP14 (bottom) and GP15 (left).

```
from machine import Pin

# Define button pins
buttons = [10, 11, 14, 15]

# Initialize buttons
buttonOne = Pin(buttons[0], Pin.IN, Pin.PULL_DOWN)
buttonTwo = Pin(buttons[1], Pin.IN, Pin.PULL_DOWN)
buttonThree = Pin(buttons[2], Pin.IN, Pin.PULL_DOWN)
buttonFour = Pin(buttons[3], Pin.IN, Pin.PULL_DOWN)

# Define button handler functions
def buttonUp(pin):
    print("Button Up Pressed")

def buttonRight(pin):
    print("Button Right Pressed")

def buttonDown(pin):
    print("Button Down Pressed")

def buttonLeft(pin):
    print("Button Left Pressed")

# Attach interrupt handlers to buttons
buttonOne.irq(trigger=Pin.IRQ_RISING, handler=buttonUp)
buttonTwo.irq(trigger=Pin.IRQ_RISING, handler=buttonRight)
buttonThree.irq(trigger=Pin.IRQ_RISING, handler=buttonDown)
buttonFour.irq(trigger=Pin.IRQ_RISING, handler=buttonLeft)
```

4.7 SERVOS

A servo consists of an electric motor with gearbox and control electronics. On the output side of the gearbox there is a gear wheel on which the servo horn is mounted. Servos are used in model making, for example to control the wing or rudder position of an airplane or ship. More and more servos are also being used in automotive engineering to automatically close doors, for window regulators, mirrors and other adjustable elements.

The servo connections are the GPIO pins GP7, GP8, GP9 and GP20.

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

# Servo pin numbers
servoOnePin = 7
servoTwoPin = 8
servoThreePin = 9
servoFourPin = 20

# Initialize servos
servoOne = PWM(Pin(servoOnePin))
servoTwo = PWM(Pin(servoTwoPin))
servoThree = PWM(Pin(servoThreePin))
servoFour = PWM(Pin(servoFourPin))

# Servo degree positions in nanoseconds
deg0 = 500000
deg45 = 1000000
deg90 = 1500000
deg135 = 2000000
deg180 = 2500000

while True:
    # Move each servo through a range of angles
    for servo in [servoOne, servoTwo, servoThree, servoFour]:
        servo.duty_ns(deg0)
        sleep(1)
        servo.duty_ns(deg45)
        sleep(1)
        servo.duty_ns(deg90)
        sleep(1)
        servo.duty_ns(deg135)
        sleep(1)
        servo.duty_ns(deg180)
        sleep(1)
```

4.8 INTERFACES

Interface connections play a crucial role in the world of electronics, similar to buttons in user interfaces. They enable communication and power supply between different electronic components. The following connections can therefore be found in the interface area on our Explorer Board:

SPI (Serial Perhipheral Interface): This connection is used for fast serial data transmission. It typically consists of four lines: MISO (Master In, Slave Out), MOSI (Master Out, Slave In), SCK (Serial Clock) and SS (Slave Select). SPI is ideal for situations where a high data transfer rate is required, such as when controlling LCD displays or SD cards.

I2C (Inter-Integrated Circuit): I2C is a two-wire interface consisting of a data line (SDA) and a clock line (SCL). It is commonly used in microcontroller applications for communication between different integrated circuits. Its simplicity makes it ideal for applications where not many GPIO pins are available.

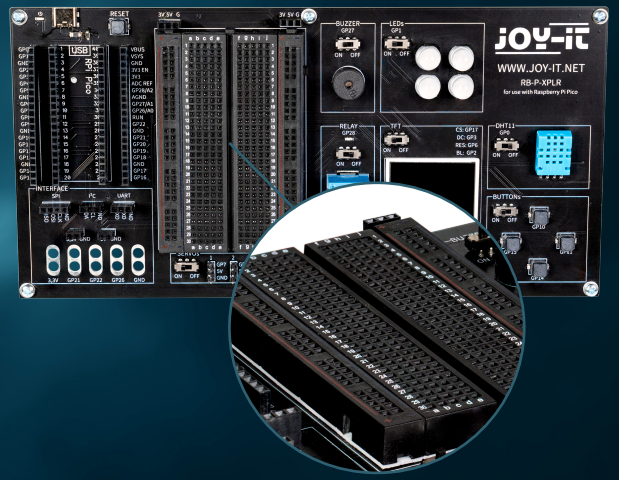
UART (Universal Asynchronous Receiver/Transmitter): This interface enables asynchronous serial communication via two lines: TX (Transmit) and RX (Receive). UART is often used for communication between microcontrollers and computers or for connecting modules such as GPS receivers or Bluetooth modules.

3.3 V and 5 V connections: These connections provide the power supply for electronic components. 3.3 V is often used for modern microcontrollers and sensors, while 5 V is often found in older or more power-hungry devices.

Connections for crocodile clips: These connectors are ideal for temporary connections or for test purposes. They enable quick and easy connection to various components or measuring devices without soldering. There are a total of five such connectors on the Explorer Board, which can be used flexibly for a variety of applications.

Each of these connections has its specific application and meaning in electronics, similar to how different types of buttons in a user interface have different functions. They provide the necessary flexibility and functionality for setting up and expanding electronic systems.

4.9 BREADBOARD



Breadboards are an indispensable tool in the world of electronics, much like interface connectors are crucial for connecting different components. They allow electronic circuits to be built and tested quickly and without soldering, making them particularly popular for prototyping and educational purposes.

A breadboard typically consists of a rectangular plastic block with a large number of embedded holes arranged in rows. These holes are internally connected by metallic traces that allow components and wires to be easily plugged in and connected. The standard layout of a breadboard includes two main areas:

The main areas: These consist of a series of parallel rows of holes, usually separated by a central groove. The holes within a row are electrically connected to each other. This arrangement is ideal for inserting integrated circuits (ICs) and other components.

The power strips: At the edge of the breadboard there are usually one or two rows of holes that serve as power strips. These are connected vertically along the entire length of the breadboard and offer a convenient way to provide power and ground at various points on the circuit.

The flexibility of a breadboard lies in its reusability and the ability to build circuits without permanent changes. This makes it ideal for experimentation as errors can be easily corrected and components removed without damage. It is also an excellent learning tool as it promotes understanding of circuit logic and component functions in a practical, visual way.

In addition, breadboards are available in different sizes and with different numbers of connection points to suit different requirements. Smaller breadboards are good for simple projects and experiments, while larger ones are suitable for more complex circuits.

Despite their versatility, breadboards also have limitations. They are not well suited for very high frequencies or for circuits that require high power. Also, the connections can sometimes be less reliable than soldered connections, especially if the breadboard wears out over time.

Overall, breadboards are an essential tool for anyone working with electronics - from beginners learning the basics to experienced developers looking to prototype quickly and efficiently. They are the electronic equivalent of an artist's sketchbook: a place to explore ideas and experiment before the final work is created.

5. PROJECTS

Welcome to the chapter on innovative electronics projects with the Raspberry Pi Pico! In this section you will be introduced to a wide range of applications applications ranging from the simple control of LEDs to the development of more complex systems such as automated weather stations and dynamic lighting systems. Each project is carefully designed to give you hands-on experience with a variety of hardware components.

Start your journey of discovery with basic projects that teach you how to use GPIOs (General Purpose Input/Output) on the Raspberry Pi Pico, and increase your skills with more advanced topics such as controlling servo motors or using sensors for environmental monitoring. Using components such as rotary encoders, ultrasonic sensors, buzzers and Neopixel LEDs, you will learn how to design interactive and reactive systems.

Each project provides a detailed introduction to the components required, step-by-step instructions on hardware configuration and clear examples of program code to help you understand the principles of electronics and computer programming. It also explains how to integrate external sensors and actuators to collect and respond to real-time data.

These projects are not only educational, but also fun, with plenty of opportunities for customization and expansion so you can develop your own creative solutions. Whether you're a beginner just starting to explore the world of digital electronics or an experienced developer looking to expand your skills, this chapter provides the resources and inspiration you need to improve your technical skills and have fun learning. Prepare to expand your programming and electronics skills as you are guided through each project while having fun creating and experimenting.

You will find the respective example codes at the end of each project. You can also download the files **here**.

5.1 DISTANCE DISPLAY

In our first project, our goal is to build an ultrasonic rangefinder that visualizes distances on our TFT display. This project is a great introduction to sensing and visualizing data with your Raspberry Pi Pico.

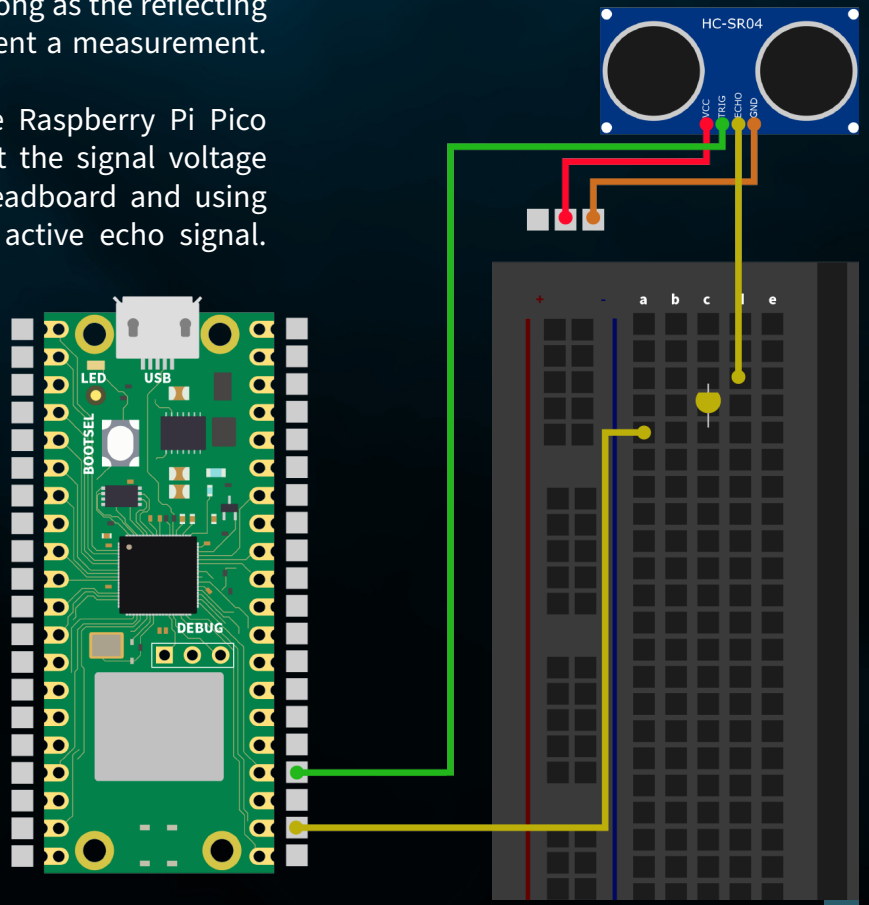
ULTRASONIC SENSOR: A transmitter emits an ultrasonic wave and measures the time until it is reflected and arrives back at the transmitter. As the speed of sound is known in various media such as air (343 m/s at 20 °C) and water (1,484 m/s), the distance to the reflecting surface can be calculated (halving the transit time, as the distance there and back was measured). A pulse from the microcontroller at the trigger input triggers a sequence of eight short ultrasonic pulses. As soon as the signal is received again, the echo output briefly goes high. The time between the trigger and the echo signal corresponds to the transit time. Distance measurement is possible in the range from about 2 cm to 400 cm and is quite precise as long as the reflecting surface is as hard and even as possible. Soft materials such as carpeting can even prevent a measurement.

Since the ultrasonic sensor is a sensor that requires a power supply of 5 V, but the Raspberry Pi Pico can only process 3.3 V signals without any problems, it is necessary to down-convert the signal voltage to avoid damage. We achieve this by connecting our yellow LED in series on the breadboard and using it to down-convert our signal. The LED also functions as a signal indicator for the active echo signal.

Further details on LEDs can also be found in chapter 5.5.

RASPBERRY PI PICO	ULTRASONIC SENSOR
3V3	VCC
GP17	Trig
GP16	Echo
GND	GND

ATTENTION! For this project it is necessary to set the switches for the **RELAY** and the **DHT11** to **OFF** and the switch for the **TFT DISPLAY** to **ON**.



SUMMARY: In our first project, we measure distances with the ultrasonic sensor and visualize the measured distance by filling the graphic on the TFT display to a greater or lesser extent. In our example, we fill the display completely from a measured distance of 100 cm.

```
# Load libraries
from machine import Pin, SPI
import ST7735
import time
import lcd_gfx

# Initialization of GPIO16 as input and GPIO17 as output
trig = Pin(17, Pin.OUT)
echo = Pin(16, Pin.IN, Pin.PULL_DOWN)

# Initialize LCD
spi = SPI(0, baudrate=8000000, polarity=0, phase=0, sck=Pin(18), mosi=Pin(19),
miso=Pin(16))
lcd = ST7735.ST7735(spi, rst=6, ce=17, dc=3)
backlight = Pin(2, Pin.OUT)
backlight.high()
lcd.reset()
lcd.begin()
lcd.fill_screen(lcd.rgb_to_565(255, 255, 255))

def translate(value, leftMin, leftMax, rightMin, rightMax):
    # Figure out how 'wide' each range is
    leftSpan = leftMax - leftMin
    rightSpan = rightMax - rightMin

    # Convert the left range into a 0-1 range (float)
    valueScaled = float(value - leftMin) / float(leftSpan)

    # Convert the 0-1 range into a value in the right range.
    return rightMin + (valueScaled * rightSpan)

# Endless loop for measuring the distance
while True:
     # Distance measurement is started using the 10us trigger signal
     trig.value(0)
     time.sleep(0.1)
     trig.value(1)

     # Now wait at the echo input until the signal has been activated
     # Then the time is measured for how long it remains activated
     time.sleep_us(2)
     trig.value(0)
     while echo.value()==0:
          pulse_start = time.ticks_us()
     while echo.value()==1:
          pulse_end = time.ticks_us()
     pulse_duration = pulse_end - pulse_start
```

Initialization of the ultrasonic sensor and the TFT display

Auxiliary function for adjusting the value range

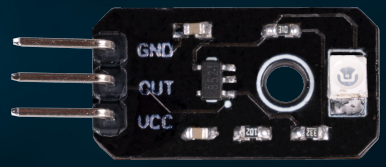
Distance measurement

```
# Now the distance is calculated using the recorded time
distance = pulse_duration * 17165 / 1000000
distance = round(distance, 0)

# Serial output
print ('Distance:',"{:.0f}".format(distance),'cm')
time.sleep(1)

# Adjust measured value to LCD height
if(distance > 100):
    distance = 100
drawHeight = round(translate(distance, 0, 100, 0, 160))

# Fill the TFT display
lcd.fill_screen(lcd.rgb_to_565(255, 255, 255))
lcd.draw_block(0, 0, 128, drawHeight, lcd.rgb_to_565(0, 255, 0))
```

Calculation of the value range and description of the TFT display

5.2 WEATHER STATION

Dive into the second project of our electronics adventure where you create your own weather station! This project combines the use of a UV sensor with the DHT11 temperature and humidity sensor to not only give you insight into the current weather conditions, but also measure the UV radiation at your location. All this information is clearly displayed on the colorful TFT display, so you can see the weather and UV radiation at a glance.



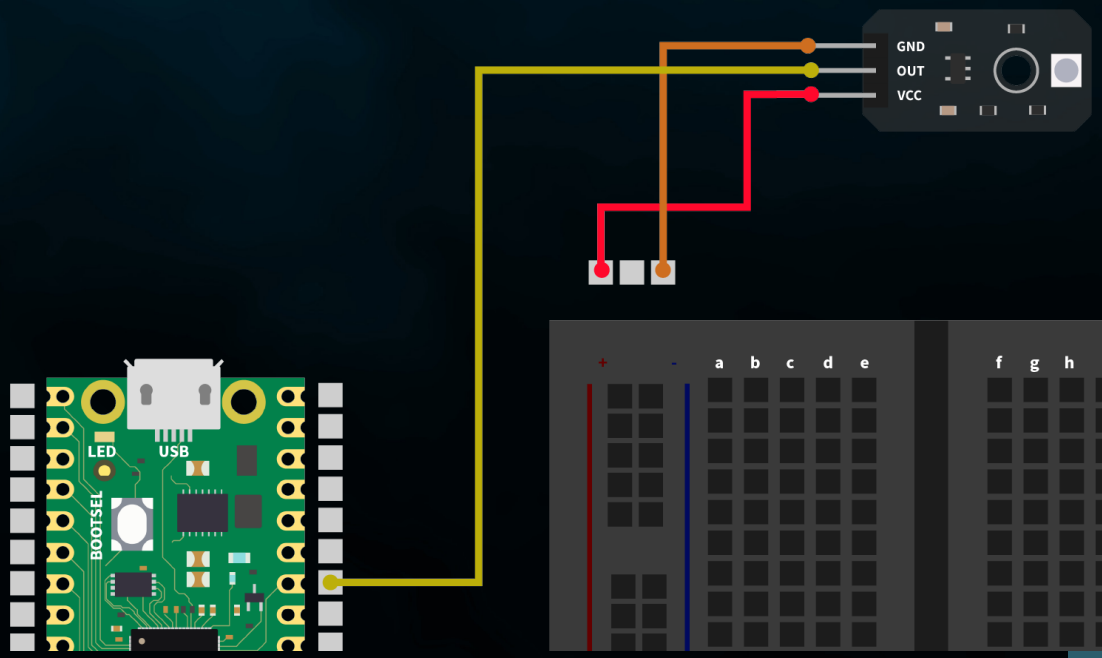
UV-SENSOR: The UV sensor is a small component that helps us to measure the invisible ultraviolet (UV) rays of the sun. UV rays are the part of sunlight that is invisible but can have an impact on our skin and health. Think of sunburn or tanning of the skin - both are caused by UV rays.

At its core, the sensor contains a material that reacts to UV radiation. When UV rays hit this material, the sensor changes its electrical resistance. This change is converted by the sensor into a signal that we can measure and read out. With the help of the Raspberry Pi Pico, we can then convert this signal into a value that indicates how strong the UV radiation currently is.

First connect the UV sensor to your Raspberry Pi Pico using the enclosed cables. Although you can of course also plug it onto the breadboard, you are much more flexible here with a direct cable connection.

RASPBERRY PI PICO	UV-SENSOR
GND	GND
GP28	OUT
3V3	VCC

ATTENTION! For this project, it is necessary to set the switch for the **RELAY** to **OFF** and the switches for the **TFT DISPLAY** and the **DHT11** sensor to **ON**.



SUMMARY: Our weather station reads the DHT11 and UV sensors and outputs the data on the TFT display.

```
from machine import ADC, Pin, SPI
import utime
import dht
import ST7735  # Assuming this is the library for your TFT display

# Initialize DHT11 sensor
sensor_dht11 = dht.DHT11(Pin(0))

# Initialize UV sensor
uv_sensor = ADC(2)  # Assuming GP28 is ADC pin number 1 in your configuration

# Initialize LCD
spi = SPI(0, baudrate=8000000, polarity=0, phase=0, sck=Pin(18), mosi=Pin(19),
miso=Pin(16))
lcd = ST7735.ST7735(spi, rst=Pin(6), ce=Pin(17), dc=Pin(3))
backlight = Pin(2, Pin.OUT)
backlight.high()
lcd.reset()
lcd.begin()
lcd.fill_screen(lcd.rgb_to_565(255, 255, 255))

while True:
    lcd.fill_screen(lcd.rgb_to_565(255, 255, 255))

    # Read UV value
    uv_value = uv_sensor.read_u16()
    # Conversion in percent
    uv_percent = (uv_value / 65000) * 100
    print("UV Intensity (percent):", uv_percent)

    # DHT11 Read values
    sensor_dht11.measure()
    temp = sensor_dht11.temperature()
    humid = sensor_dht11.humidity()

    # Display values on LCD
    lcd.p_string(20, 20, "Temp: {}C".format(temp))
    lcd.p_string(20, 40, "Humid: {}%".format(humid))
    lcd.p_string(20, 60, "UV: {:.2f}%".format(uv_percent))  # Display of UV
intensity in percent with two decimal places

    utime.sleep(10)
```

Initialization of the TFT display

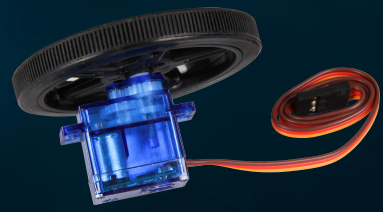
Measuring the sensor values

Output on the display

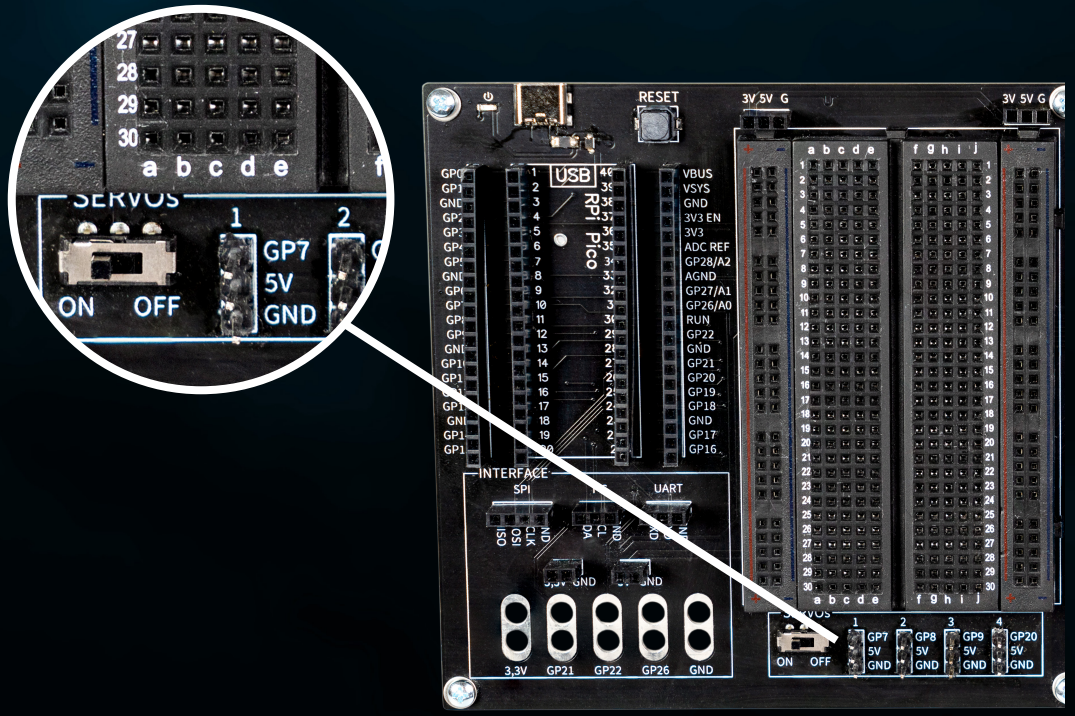
5.3 SERVO CONTROL

Welcome to the third project in our series of exciting electronics adventures with the Explorer Set! This time it's all about movement and control. Our aim is to program and control a servo motor so that its direction of rotation can be controlled by simply pressing a button. This project not only provides an excellent introduction to the world of motor control, but also shows how to reinforce interactions through visual feedback on a TFT display.

SERVOMOTOR: A servo consists of an electric motor with gearbox and control electronics. On the output side of the gearbox there is a gear wheel on which the servo wheel is mounted. Servos are used in model making, for example to control the wing or rudder position of an airplane or ship. More and more servos are also being used in automotive engineering to automatically close doors, for window regulators, mirrors and other adjustable elements.



First connect the servo motor to the **SERVO INTERFACE** with the number **1** on your Explorer Board.



ATTENTION! For this project it is necessary to set the switch for the **TFT DISPLAY,** the **BUTTONS** and the **SERVOS** to **ON**.

SUMMARY: We control our servomotor and let it switch between left and right rotation, controlled by our buttons.

Initialization of the servomotor and buttons

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

# Servo pin numbers
servoOnePin = 7

# Key pin numbers
buttonLeftPin = 15
buttonRightPin = 11

# Initialization of the servo
servoOne = PWM(Pin(servoOnePin))

# Initialize the buttons with PULL_UP to use the default HIGH state
buttonLeft = Pin(buttonLeftPin, Pin.IN, Pin.PULL_UP)
buttonRight = Pin(buttonRightPin, Pin.IN, Pin.PULL_UP)

# Servo speeds in nanoseconds
leftSpeed = 1300000  # Moves the servo to the left
rightSpeed = 1700000  # Moves the servo to the right

# Servo frequency
servoOne.freq(50)  # Typical servo frequency of 50Hz

# Status of the servo
servoState = 'left'  # Starts with counterclockwise rotation

# Last state of the buttons to recognize edges
lastButtonLeft = buttonLeft.value()
lastButtonRight = buttonRight.value()

while True:
    # Read out the current status of the buttons
    currentButtonLeft = buttonLeft.value()
    currentButtonRight = buttonRight.value()

    # Check whether an edge from HIGH to LOW was detected (button was pressed)
    if lastButtonLeft == 1 and currentButtonLeft == 0:
        servoState = 'right'  # Changes the direction to the right when the left
button is pressed
    elif lastButtonRight == 1 and currentButtonRight == 0:
        servoState = 'left'  # Changes the direction to the left when the right
button is pressed
```

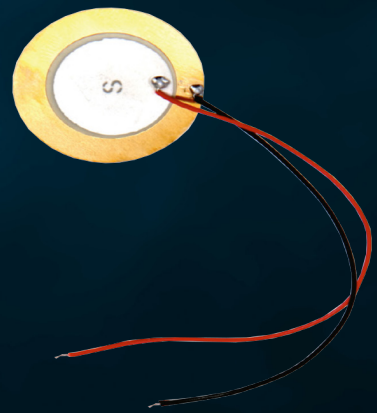
Checking the buttons

```
    # Update the states for the next run
    lastButtonLeft = currentButtonLeft
    lastButtonRight = currentButtonRight

    # Controls the servo based on the current status
    if servoState == 'left':
        servoOne.duty_ns(leftSpeed)  # Moves the servo to the left
    else:
        servoOne.duty_ns(rightSpeed)  # Moves the servo to the right

    sleep(0.1)  # Short break for the control cycle
```

Servo control

5.4 SELF-MADE BUZZER

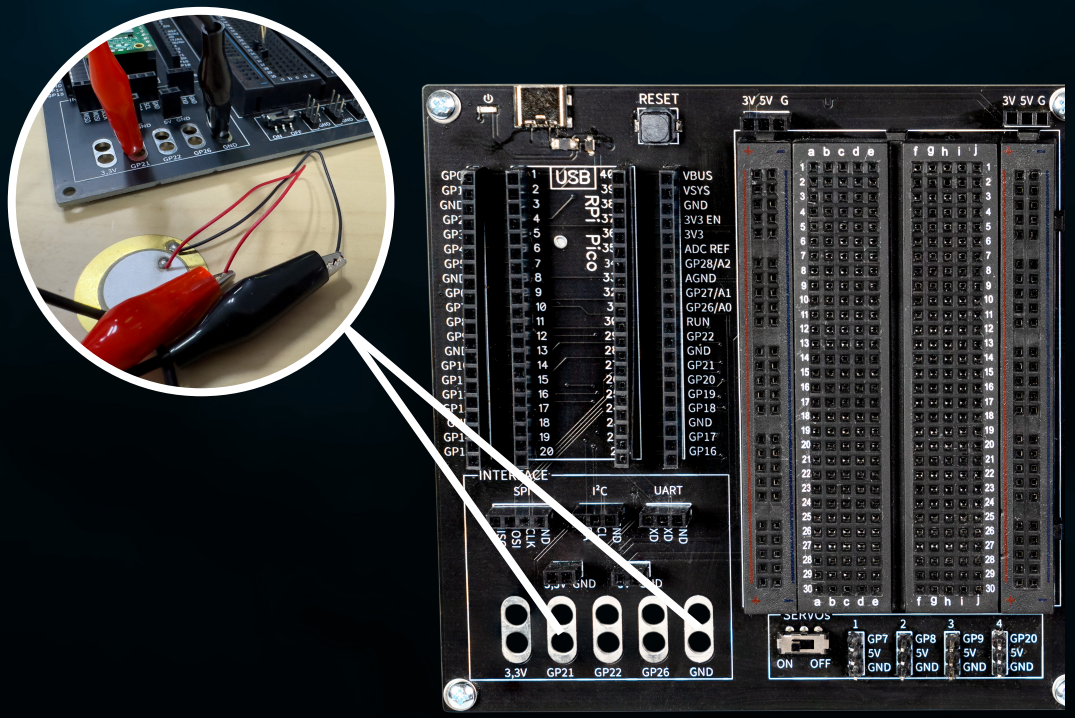
The fourth project in our electronics adventure with the Explorer Set is all about sound! We delve into the world of acoustic signals by creating our own buzzer circuit. This project not only gives you the opportunity to understand the basics of circuit creation, but also to learn how to create audible signals using simple tools. The use of alligator clips makes the build particularly user-friendly and accessible, even for those who are still at the beginning of their electronics journey.



First, we carefully connect the buzzer to the Explorer Board using crocodile clips. These terminals are ideal for quick and flexible connections without the need for soldering. The buzzer, a small component capable of producing sound, becomes the centerpiece of our project. When current is applied, the buzzer vibrates and produces a sound.

First, connect an alligator clip to the GP21 alligator clip connector on your Explorer board. Connect the other end of the crocodile clip to the red buzzer cable. Connect another crocodile clip to the GND crocodile clip connection on your Explorer board. Connect the other end of the terminal to the black buzzer cable.

RASPBERRY PI PICO	BUZZER
GP21	Red cable
GND	Black cable



ATTENTION! For this project, it is necessary to set the switch for the **BUTTONS** to **ON**.

SUMMARY: We connect an external buzzer to our Raspberry Pi Pico to play a simple, fun tune.

```
from machine import Pin, PWM
import utime
# Note frequencies (in Hz)
notes = {
    'C4': 262,
    'D4': 294,
    'E4': 330,
    'F4': 349,
    'G4': 392,
    'A4': 440,
    'B4': 494,
    'C5': 523
}
# Melody and duration (in ms)
melody = ['C4', 'D4', 'E4', 'C4', 'C4', 'D4', 'E4', 'C4', 'E4', 'F4', 'G4', 'E4',
'F4', 'G4']
durations = [500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 1000, 500, 500,
1000]
# Initialization of the buzzer
buzzer = PWM(Pin(21))
buzzer.freq(440) # Set a start frequency
# Function to play a note
def play_note(note, duration):
    if note in notes:
        buzzer.freq(notes[note]) # Set the frequency based on the note
        buzzer.duty_u16(32767) # Start the PWM signal
        utime.sleep_ms(duration) # Hold the note for the duration
    buzzer.duty_u16(0) # Stop the PWM signal (switch off note)
    utime.sleep_ms(50) # Short pause between the notes
# Play the melody
for note, duration in zip(melody, durations):
    play_note(note, duration)
buzzer.deinit() # Deactivate the PWM channel when finished
```

List of notes

Melody memory

Function for playing the notes

5.5 YOUR OWN CIRCUIT

In the fifth project of our electronics adventure with the Explorer Set, we explore the fascinating world of light control. This time we are building a circuit that you can use to control LEDs. This provides a fantastic opportunity to understand the principles of electronic circuits while gaining control over lighting effects.

LEDS: LEDs, or light-emitting diodes, are small but powerful light sources that are used in many electronic projects. They have many advantages over traditional light bulbs, such as a longer lifespan, lower energy consumption and the ability to light up in different colors. An LED consists of a semiconductor material that emits light when an electric current flows through it.

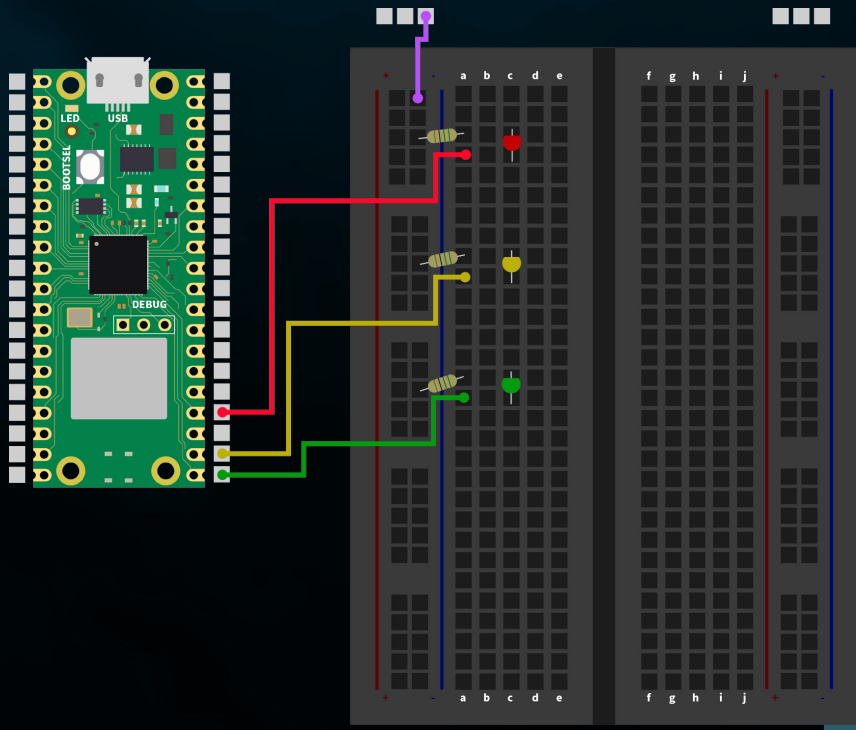
It is important to recognize the correct polarity of LEDs, as they only work if the current flows through them in the right direction. This means that the positive pole of the power source must be connected to the positive end of the LED and the negative pole of the power source must be connected to the negative end of the LED.

This is how you recognize the polarity of an LED:

Longer leg: for most LEDs, the longer leg is the anode (+), i.e. the positive connection. The shorter leg is the cathode (-), i.e. the negative connection.

Flat edge: There may be a flat edge on the side of the LED housing. This side usually marks the cathode, i.e. the negative pole.

First reconstruct the circuit as shown in the following diagram. But make sure that the polarity of the LEDs is correct. Also use a 56 Ω (green-blue-black) series resistor for each LED.

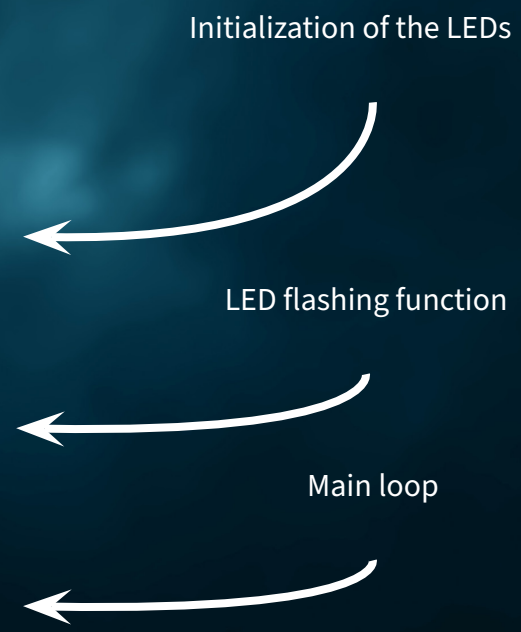


RASPBERRY PI PICO	LEDS
GP18	Red LED
GP17	Yellow LED
GP16	Green LED

ATTENTION! For this project, it is necessary to set the switch for the **TFT DISPLAY** to **OFF**.

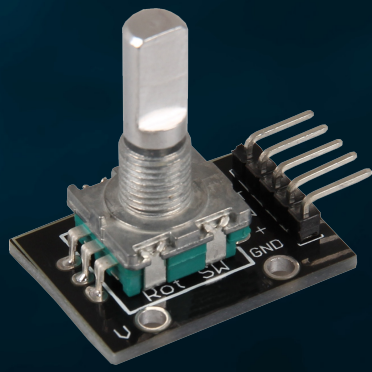
SUMMARY: We control three different LEDs via the pins of the Raspberry Pi Pico, with each LED flashing alternately.

```
from machine import Pin
import utime

# Initialize the LEDs
red_led = Pin(18, Pin.OUT)
yellow_led = Pin(17, Pin.OUT)
green_led = Pin(16, Pin.OUT)

# Flashing function for one LED
def blink_led(led, duration):
    led.value(1)  # Switch on LED
    utime.sleep(duration)
    led.value(0)  # Switch off LED
    utime.sleep(duration)

# Hauptschleife
while True:
    blink_led(red_led, 0.5)     # Red LED flashes for 0.5 seconds
    blink_led(yellow_led, 0.5) # Yellow LED flashes for 0.5 seconds
    blink_led(green_led, 0.5)   # Green LED flashes for 0.5 seconds
```



5.6 LED CONTROL

In the sixth project of our electronics adventure, we use our rotary encoder to control the brightness and color of LEDs - a simple yet fascinating way to immerse yourself in electronics. The rotary encoder, our central control element, enables playful interaction thanks to its dual function. Changes in brightness are controlled by rotation, while pressing the encoder cycles through the LED colors - ideal for illustrating the basics of electronics and color mixing.

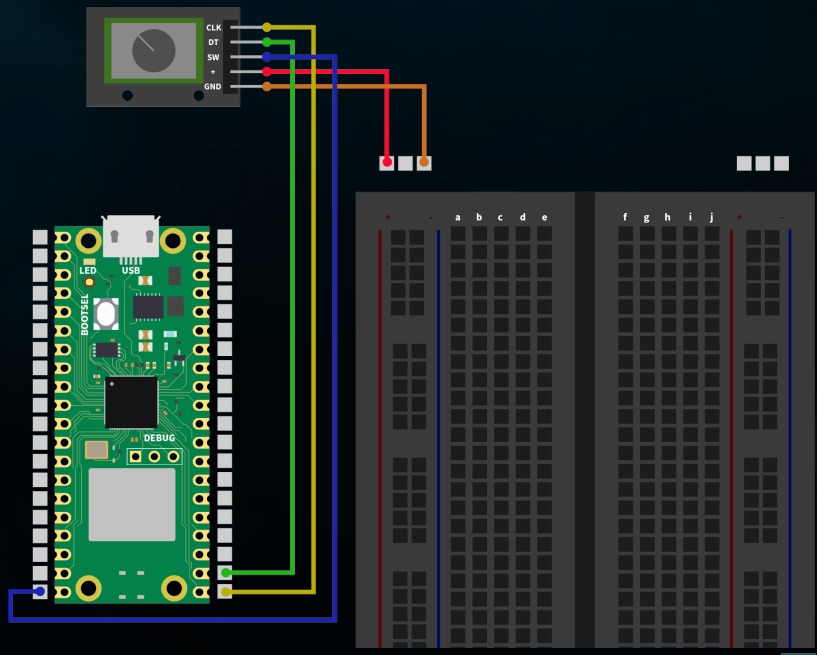


ROTARY ENCODER: The rotary encoder is a clever little device that converts your rotary movements into electronic signals. Imagine a rotary knob like the one you know from a radio. When you turn this knob, the rotary encoder can measure how far and in which direction you have turned it. This information can then be used, for example, to change the volume, navigate through menus or, in our projects, adjust the brightness of LEDs.

Rotary encoders often also have a built-in button - this means they can also function as a pressure switch. When you press the rotary knob, the rotary encoder recognizes this pressure as a separate signal. This can be used for various functions, such as switching a device on and off or changing operating modes.

TO SUMMARIZE: A rotary encoder allows you to send various commands to your electronics projects by turning and pressing. It is an intuitive and versatile tool that makes interactions with your projects easy and fun.

First connect the rotary encoder to your Raspberry Pi Pico as follows:



RASPBERRY PI PICO	ROTARY ENCODER
GP16	CLK
GP17	DT
GP15	SW
3V3	+
GND	GND

ATTENTION! For this project, it is necessary to set the switch for the **TFT DISPLAY** to **OFF** and the switch for the **LEDS** to **ON**.

SUMMARY: We use the rotary encoder to control the color and brightness of our four LEDs. Turning the encoder changes the brightness, while pressing the encoder adjusts the color of the LEDs.

```
from machine import Pin
import utime
import neopixel

# Neopixel setup
NUM_LEDS = 4
PIXEL_PIN = 1
np = neopixel.NeoPixel(Pin(PIXEL_PIN), NUM_LEDS)

# Rotate encoder setup
PIN_CLK = Pin(16, Pin.IN, Pin.PULL_UP)
PIN_DT = Pin(17, Pin.IN, Pin.PULL_UP)
BUTTON_PIN = Pin(15, Pin.IN, Pin.PULL_UP)

# Global variables
counter = 0
PIN_CLK_LAST = PIN_CLK.value()
delayTime = 0.001
debounce_time_encoder = 0
debounce_time_button = 0

# Initialize colors
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 255)]  # Rot, Grün,
Blau, Weiß
color_index = 0

# Initialize brightness
brightness_levels = [0.2, 0.4, 0.6, 0.8, 1.0]
brightness_index = 0

def update_leds(color, brightness):
    dimmed_color = tuple([int(c * brightness) for c in color])
    for i in range(NUM_LEDS):
        np[i] = dimmed_color
    np.write()

def rotaryFunction(null):
    global counter, brightness_index, debounce_time_encoder
    PIN_CLK_CURRENT = PIN_CLK.value()
    if PIN_CLK_CURRENT != PIN_CLK_LAST and (utime.ticks_ms() - debounce_time_
encoder) > 300:
        if PIN_DT.value() != PIN_CLK_CURRENT:
            brightness_index = (brightness_index + 1) % len(brightness_levels)
        else:
            brightness_index = (brightness_index - 1) % len(brightness_levels)
        update_leds(colors[color_index], brightness_levels[brightness_index])
        debounce_time_encoder = utime.ticks_ms()
```

Initialization of the LEDs and the rotary encoder

Function for the rotary encoder

```
def counterReset(null):
    global color_index, debounce_time_button
    if (utime.ticks_ms() - debounce_time_button) > 300:
        color_index = (color_index + 1) % len(colors)
        update_leds(colors[color_index], brightness_levels[brightness_index])
        debounce_time_button = utime.ticks_ms()



PIN_CLK.irq(trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING, handler=rotaryFunction)
BUTTON_PIN.irq(trigger=Pin.IRQ_FALLING, handler=counterReset)

update_leds(colors[color_index], brightness_levels[brightness_index])

while True:
    utime.sleep(delayTime)
```

List of notes

5.7 AUTOMATIC BRIGHTNESS CONTROL

In the seventh project of our electronics adventure, we use a photodiode to automatically control the brightness of LEDs. The photodiode converts light into an electrical signal so that the LEDs light up brighter when it is dark and dim when there is more ambient light.

By connecting the photodiode to the Explorer Board and programming it on the Raspberry Pi Pico, the LEDs adapt intelligently to the ambient brightness. This project shows how reactive and energy-saving electronic systems can be built with simple components.

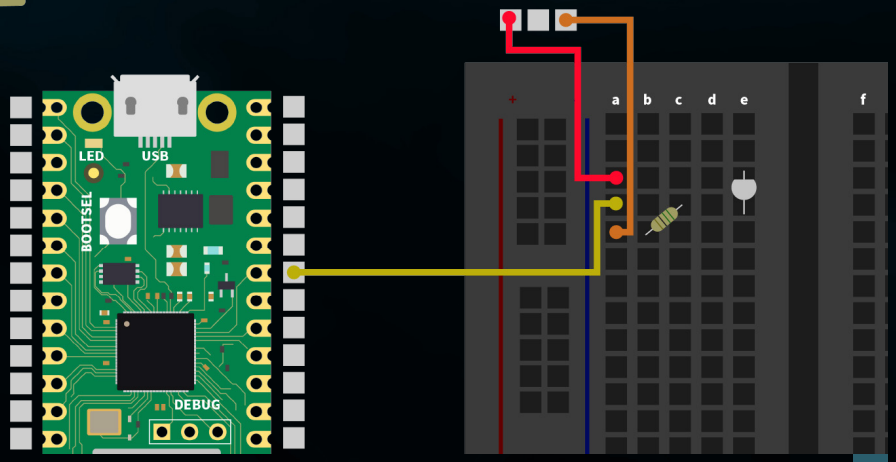
PHOTODIODE: A photodiode is a special type of semiconductor that responds to light hitting it by generating an electric current. Think of a photodiode like a small solar panel: when light falls on it, it converts this light into an electrical signal. The more light that reaches the photodiode, the stronger the signal becomes.

Photodiodes are very sensitive and can detect even small amounts of light, which makes them ideal for projects where it is important to measure the brightness or presence of light. For example, they can be used in automatic brightness controllers, light sensors or as part of a lighting control system.

In short, photodiodes are effective light detectors that allow us to make electronic devices react intelligently to changes in ambient lighting.

First connect the photodiode via the breadboard as follows. Please note that the use of a resistor is also required here. Use the 100 kΩ (brown-black-yellow) resistor here.

RASPBERRY PI PICO	PHOTODIODE
3V3	Positive cathode
GP28/A2	Negative anode



ATTENTION! For this project it is necessary to set the switch for the **RELAY** to **OFF** and the switch for the **LEDS** to **ON**.

SUMMARY: We use our photodiode to measure the ambient brightness and adjust the brightness of four LEDs. The intensity of the LEDs changes according to the light detected by the photodiode, with darker environments leading to brighter LEDs and vice versa. It is best to use a flashlight to achieve the best possible result.

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

# Neopixel setup
NUM_LEDS = 4
PIXEL_PIN = 1
np = neopixel.NeoPixel(Pin(PIXEL_PIN), NUM_LEDS)

# Photodiode setup on ADC pin GP28 (A2)
fotodiode = ADC(2)

# Conversion function for brightness values of the photodiode into a suitable
brightness for the LEDs
def brightness_from_light(sensor_value):
    # Minimum and maximum sensor value
    min_sensor_value = 400
    max_sensor_value = 10000

    # Invert the sensor value within the actual range
    normalized_value = max_sensor_value - sensor_value + min_sensor_value

    # Scale the inverted value to a brightness range (0.05 to 0.5)
    # Adjust the scaling: Divide by (max_sensor_value - min_sensor_value)
    return max(0.05, min(0.5, normalized_value / (max_sensor_value - min_sensor_
value) * 0.45 + 0.05))

def update_leds(brightness):
    color = (255, 255, 255)  # White
    dimmed_color = tuple([int(c * brightness) for c in color])
    for i in range(NUM_LEDS):
        np[i] = dimmed_color
    np.write()


while True:
    # Read the sensor value from the photodiode
    light_value = fotodiode.read_u16()
    print(light_value)

    # Calculate the brightness based on the sensor value
    brightness = brightness_from_light(light_value)

    # Update the LEDs with the new brightness
    update_leds(brightness)

    # Waiting time to reduce the load on the CPU and for smoother brightness
transitions
    utime.sleep(0.5)
```

Initialization of the LEDs and the photodiode

Measurement of the photodiode and controlling the brightness

5.8 RGB LED CONTROL

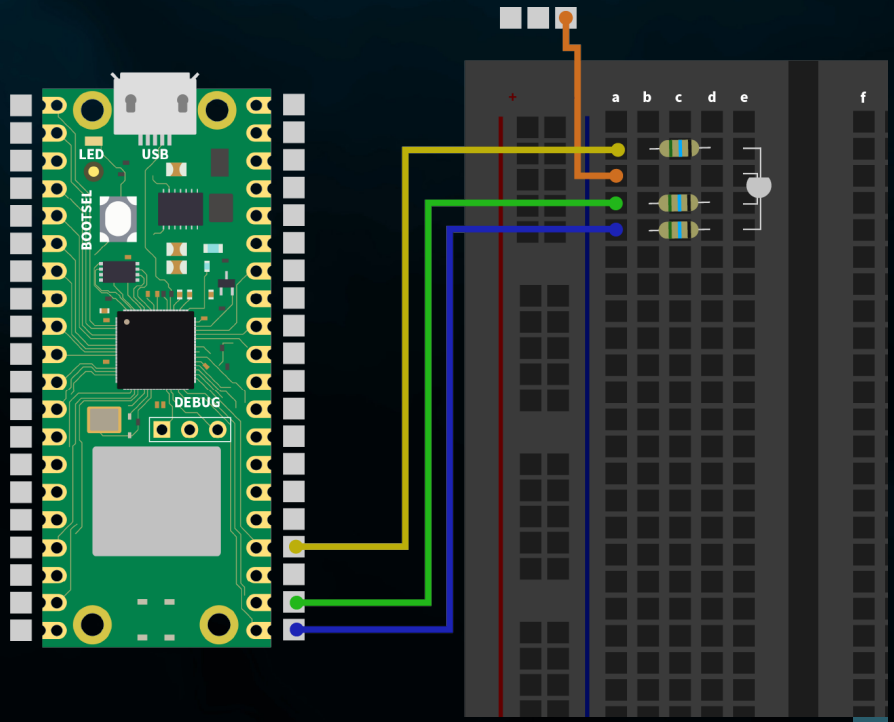
In the eighth and final project in our electronics series, we will focus on the color control of RGB LEDs using the integrated buttons on the Explorer Board. RGB LEDs are special light-emitting diodes that combine red, green and blue (RGB) light to display a wide range of colors. By adjusting the intensity of each color component individually, we can create almost any color.

In this project, we connect the RGB LED to the breadboard and use the existing buttons to control the colors of the LED. Each button is assigned to a color (red, green, blue).

RGB LED: An RGB LED combines red, green and blue in a single point of light. By changing the brightness of each of the three colors, almost any color can be created. This is done by pulse width modulation (PWM), which controls the intensity of each color. Thus, RGB LEDs with only three colors enable a wide color spectrum, ideal for colorful lighting projects.

First connect the RGB LED to the breadboard as follows. Please note that each of the three color channels also requires a series resistor here. You should use the 56 Ω resistor (green-blue-black) here.

RASPBERRY PI PICO	RGB-LED
GP18	First pin
GND	Second pin
GP17	Third pin
GP16	Fourth pin



ATTENTION! For this project it is necessary to set the switch for the **TFT** to **OFF** and for the **BUTTONS** to **ON**.

SUMMARY: The three color channels of the RGB LED (red, green & blue) are switched on and off using the buttons (left, top & right).

```
from machine import Pin
import utime

# Initialize the LED pins
red_led = Pin(18, Pin.OUT)
green_led = Pin(17, Pin.OUT)
blue_led = Pin(16, Pin.OUT)

# Initialize the button pins
button_red = Pin(15, Pin.IN, Pin.PULL_UP)
button_green = Pin(10, Pin.IN, Pin.PULL_UP)
button_blue = Pin(11, Pin.IN, Pin.PULL_UP)

# Save states of the LEDs
red_state = False
green_state = False
blue_state = False

def toggle_led(led, state):
    led.value(state)

while True:
    # Check the status of the red button
    if button_red.value() == 0:
        red_state = not red_state
        toggle_led(red_led, red_state)
        utime.sleep(0.2)  # Entprellung

    # Check the status of the green button
    if button_green.value() == 0:
        green_state = not green_state
        toggle_led(green_led, green_state)
        utime.sleep(0.2)  # Entprellung

    # Check the status of the blue button
    if button_blue.value() == 0:
        blue_state = not blue_state
        toggle_led(blue_led, blue_state)
        utime.sleep(0.2)  # Debouncing
```

Initialization of the LED and the buttons

Testing the buttons and controlling the LED

6. INFORMATION & TAKE-BACK OBLIGATIONS

OUR INFORMATION AND TAKE-BACK OBLIGATIONS UNDER THE GERMAN ELECTRICAL AND ELECTRONIC EQUIPMENT ACT (ELEKTROG)

SYMBOL ON ELECTRICAL AND ELECTRONIC EQUIPMENT: 
This crossed-out garbage can means that electrical and electronic appliances do not belong in household waste. You must hand in the old appliances at a collection point. Before handing them in, you must separate used batteries and accumulators that are not enclosed by the old appliance.

RETURN OPTIONS:
As an end user, you can return your old appliance (which essentially fulfills the same function as the new appliance purchased from us) for disposal free of charge when you purchase a new appliance. Small appliances with no external dimensions greater than 25 cm can be disposed of in normal household quantities regardless of whether you have purchased a new appliance.

POSSIBILITY OF RETURN AT OUR COMPANY LOCATION DURING OPENING HOURS:
SIMAC Electronics GmbH, Pascalstr. 8, D-47506 Neukirchen-Vluyn

RETURN OPTION IN YOUR AREA:
We will send you a parcel stamp with which you can return the device to us free of charge. To do so, please contact us by e-mail at service@joy-it.net or by telephone.

PACKAGING INFORMATION:
Please pack your old appliance securely for transportation. If you do not have suitable packaging material or do not wish to use your own, please contact us and we will send you suitable packaging.

7. SUPPORT

We are also there for you after your purchase. If any questions remain unanswered or problems arise, we are also available by e-mail, telephone and ticket support system.

E-Mail: service@joy-it.net
Ticket-System: http://support.joy-it.net
Phone: +49 (0)2845 9360 – 50 (Mon. - Thur.: 09:00 - 17:00 ó clock, Fri: 09:00 - 14:30 ó clock)

For further information, please visit our website:

WWW.JOY-IT.NET