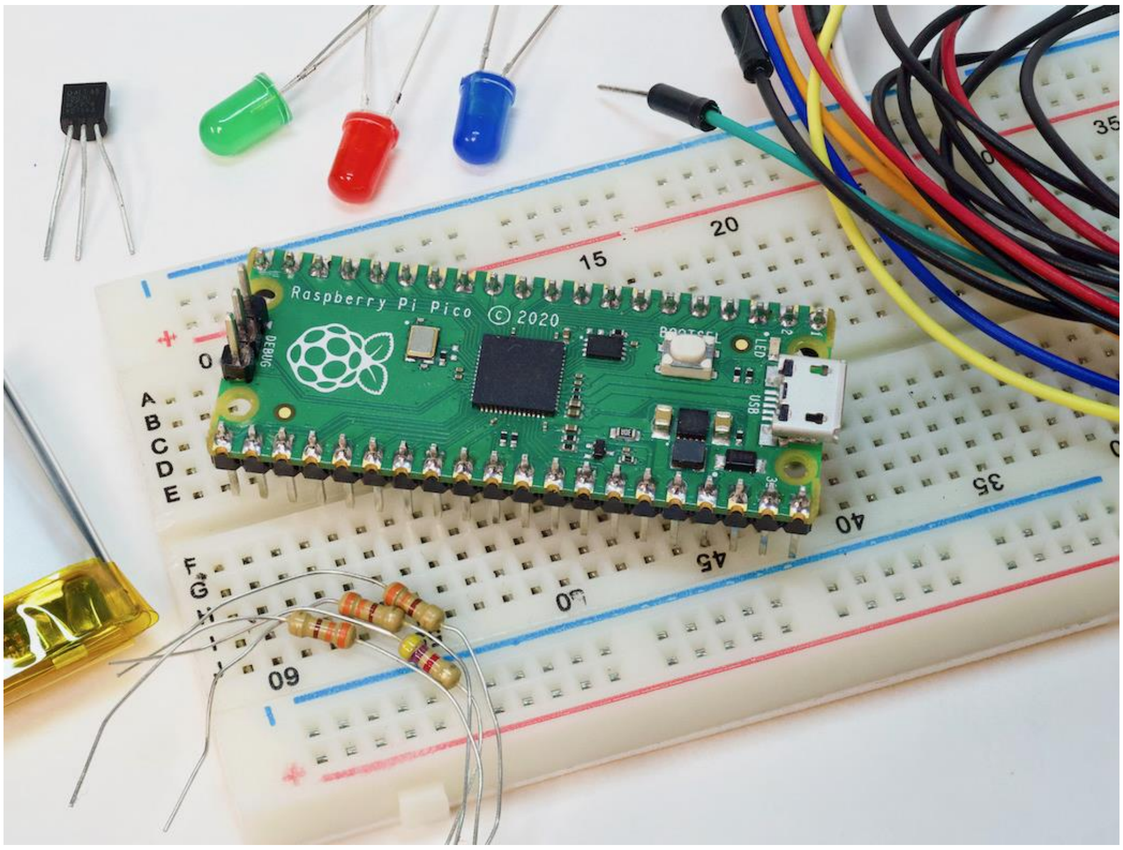
NISB Hardware Focus Group-21

“The World of Microcontrollers”

on

Raspberry Pi Pico



Session1: Getting started with the Pico

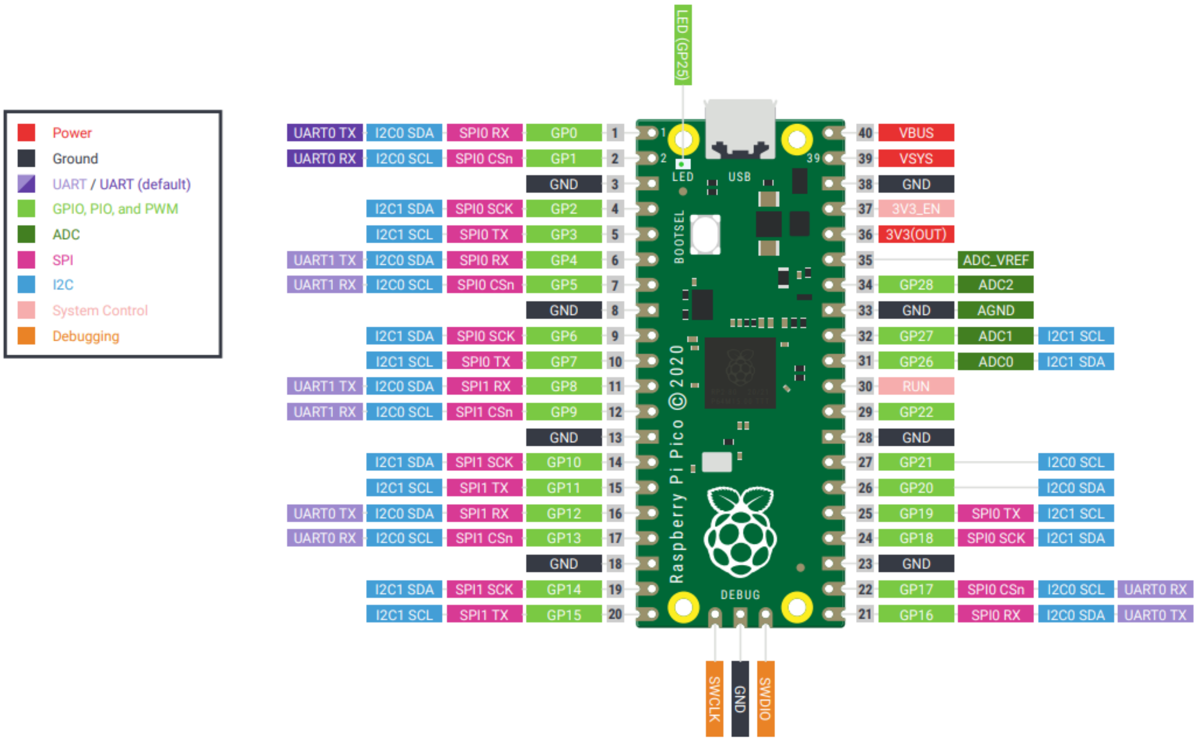
Raspberry Pi Pico is a microcontroller board built on silicon designed by Raspberry Pi. Microcontrollers are computers stripped back to their bare essentials. You don’t use monitors or keyboards, but program them to take their input from, and send their output to the input/output pins. Using these programmable connections, you can light lights, make noises, send text to screens, and much more.

The ultra-light, ultra-small Raspberry Pi Pico microcontroller board is ideal for embedding inside digital projects. Raspberry Pi Pico represents two major firsts for Raspberry Pi: it’s the first microcontroller development board from Raspberry Pi; it’s also the first device to use a silicon chip (RP2040) designed by Raspberry Pi’s in-house Application-Specific Integrated Circuit (ASIC) team.

Raspberry Pi Pico specifications:

- RP2040 microcontroller chip designed by Raspberry Pi in the United Kingdom.
- Dual-core ARM Cortex-M0+ processor, flexible clock running up to 133MHz
- 264kB of SRAM, and 2MB of on-board flash storage
- USB 1.1 Host and Device support
- 26 multifunction GPIO pins include 2× SPI, 2× I2C, 2× UART, 3× 12-bit ADC, 16× controllable PWM channels
- 8× Programmable IO (PIO) state machines for custom peripheral support
- Drag & drop programming using mass storage over USB
- Accurate clock and timer on-chip
- Temperature sensor
- Low-power sleep and dormant modes

Raspberry Pi Pico Pinout



3V3	3.3 volts power	A source of 3.3 V power, the same voltage your Pico runs at internally, generated from the VSYS input. This power supply can be switched on and off using the 3V3_EN pin above it, which also switches your Pico off.
VSYS	~2-5 volts power	A pin directly connected to your Pico's internal power supply, which cannot be switched off without also switching Pico off.
VBUS	5 volts power	A source of 5 V power taken from your Pico's micro USB port, and used to power hardware which needs more than 3.3 V.
GND	0 volts ground	A ground connection, used to complete a circuit connected to a power source. Several of these pins are dotted around your Pico to make wiring easier.
GP*xx*	General-purpose input/output pin number '*xx*'	The GPIO pins available for your program, labelled 'GP0' through to 'GP28'.
GP*xx*_ADC*x*	General-purpose input/output pin number '*xx*', with analogue input number '*x*'	A GPIO pin which ends in 'ADC' and a number can be used as an analogue input as well as a digital input or output – but not both at the same time.
ADC_VREF	Analogue-to-digital converter (ADC) voltage reference	A special input pin which sets a *reference voltage* for any analogue inputs.
AGND	Analogue-to-digital converter (ADC) 0 volts ground	A special ground connection for use with the ADC_VREF pin.
RUN	Enables or disables your Pico	The RUN header is used to start and stop your Pico from another microcontroller.

Programming with Pico- Using Thonny IDE

MicroPython adds hardware-specific modules, such as *machine*, that is used to program Pico.

Some of the classes in *machine*:

- class Pin – control I/O pins
- class Signal – control and sense external I/O devices
- class ADC – analog to digital conversion
- class PWM – allows to give analogue behaviours to digital devices like LEDs
- class UART – duplex serial communication bus
- class I2C – a two-wire serial protocol
- class RTC – real time clock
- class Timer – control hardware timers

Code1: Blinking onboard LED

- Method1- Using 'sleep' function (software timers)

```
#LED will switch on for 4 seconds and switch off for next 4
#seconds, which will be repeating in an infinite loop
from machine import Pin
from utime import sleep

led_onboard = Pin(25, Pin.OUT) #declaring Pin 25 as output

while True:
    led_onboard.value(1)
    sleep(4)
    led_onboard.value(0)
    sleep(4)
```

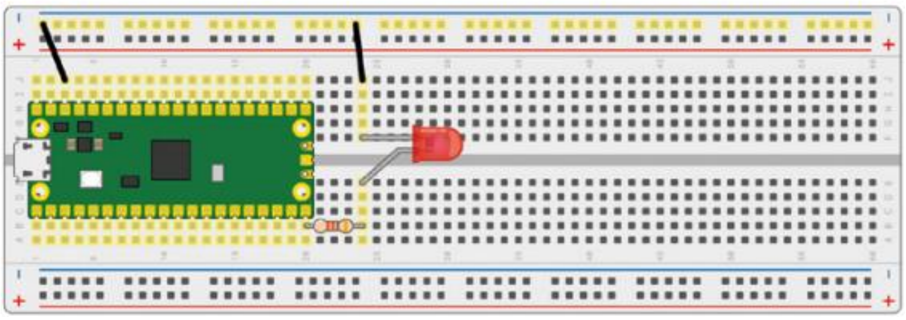
- Method2- Using Timer

Hardware timers deal with timing of periods and events, which are more accurate when compared to software delays.

```
#Initialize timer with 2.5ms which runs periodically,
#along with execution of "blink" function
from machine import Pin, Timer
led = Pin(25, Pin.OUT)
timer = Timer()

def blink(timer):
    led.toggle()#using toggle function to blink LED

timer.init(freq=2.5, mode=Timer.PERIODIC, callback=blink)
```

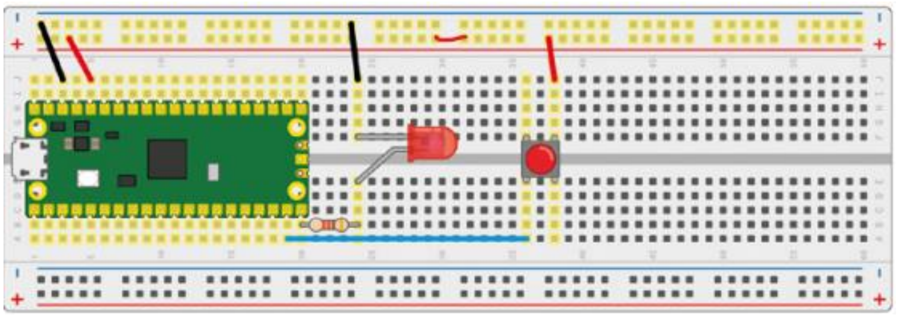
Session2: Physical Computing

Code2: Blinking external LED



```
#Connect LED to GPIO 15 of Pico along with a resistor(around 330Ω)
from machine import Pin
from utime import sleep

led_external = Pin(15,Pin.OUT) #declaring Pin 15 as output

while True:
    led_external.toggle()#LED will toggle for every 0.2 seconds
    sleep(0.2)
```

Code3: Connecting a push-button with LED



```
from machine import Pin
from utime import sleep
led_external = Pin(15, Pin.OUT) #Declaring LED as output on Pin 15
button = Pin(14, Pin.IN, Pin.PULL_DOWN) #Declaring button connected to
#Pin 14 as input, and using Pico's pull-down resistor
while True:
    if button.value() == 1: #checks for the value of button (Pin 14) is high
        led_external.toggle() #Toggle the LED when button is pressed for
0.5 seconds
        sleep(0.5)
    led_external.value(0) #Switching off LED when the button isn't being
pressed
```

Code4: Reaction Game (using interrupt)
Circuit: Refer the circuit of Code3

```
from machine import Pin
import utime
import urandom #To generate random numbers for switching off the LED

led=Pin(15,Pin.OUT) #LED as output on Pin15
button=Pin(14,Pin.IN,Pin.PULL_DOWN) #Button as input on Pin14

#starting handler by turning interrupt off, so that it triggers only once,
#and measuring the time taken to push button when LED goes off with
#printing the response time
def button_handler(pin):
    button.irq(handler=None)
    timer_reaction = utime.ticks_ms() - timer_start
    print("Your reaction time is "+ str(timer_reaction) + " milliseconds")

led.value(1)
utime.sleep(urandom.uniform(5,10)) #Generate random numbers between 5-10 to
#turn off the LED after it completes the sleep duration
led.value(0)
timer_start=utime.ticks_ms()# initialize timer to measure the duration once LED
turns off
button.irq(trigger=Pin.IRQ_RISING, handler=button_handler)#Raise an interrupt
request to Pico during the rising edge of Pico's clock, and executing the
function mentioned as handler
```

Session3: Sensors and Actuators

Code5: LED brightness control (using PWM class)

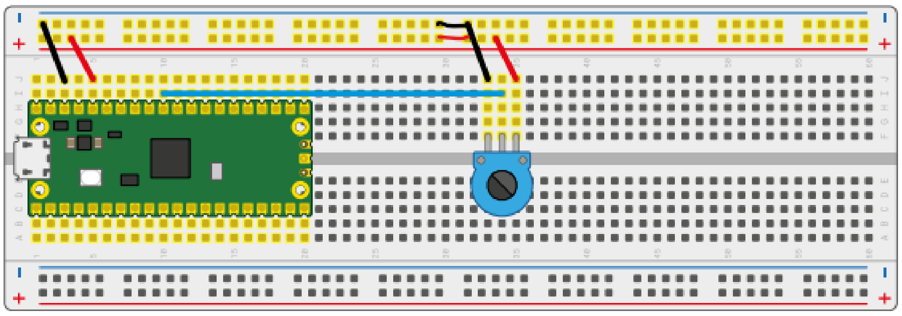
```
from machine import Pin,PWM
from utime import sleep
led = Pin(25, mode=Pin.OUT)
ledPWM = PWM(led)

ledPWM.freq(50)

while 1:
    ledPWM.duty_u16(0) #unsigned 16-bit integer 0-65535(Setting duty 0%)
    print("0%")
    sleep(1)
    ledPWM.duty_u16(65535//2) #50% duty cycle
    print("50%")
    sleep(1)
    ledPWM.duty_u16(65535)   # 100% duty cycle
    print("100%")
    sleep(1)
```

Code6: LED brightness control with smooth transition (for large values of N)

```
from machine import Pin,PWM
from utime import sleep
led = Pin(25, mode=Pin.OUT)
ledPWM = PWM(led)

ledPWM.freq(100)
N = 100
while 1:
    for i in range(N+1):
        ledPWM.duty_u16(65535*i//N) #unsigned 16-bit integer 0-
65535(Setting duty 0%)
        print(i*100/N,"%")
        sleep(1)
```

Note: Even external LED can be used for codes 5, 6, 7 and 9

Code7: Using Potentiometer for controlling LED Brightness with Pico’s in-built ADC



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

pot_pin = ADC(Pin(26))
onboard_led = PWM(Pin(25))
onboard_led.freq(1000)
while 1:
    voltage_u16 = pot_pin.read_u16()
    onboard_led.duty_u16(voltage_u16)
```

Code8: Using Pico’s in-built temperature sensor

```
import machine
import utime
sensor_temp = machine.ADC(4) #ADC Channel 4 is connected to Pico's temperature
sensor
conversion_factor = 3.3 / (65535)
while True:
    reading = sensor_temp.read_u16() * conversion_factor
    temperature = 27 - (reading - 0.706)/0.001721 #to convert voltage of
ADC into degrees Celsius
    print(temperature)
    utime.sleep(2)
```

Session4: Multi-core and PIO

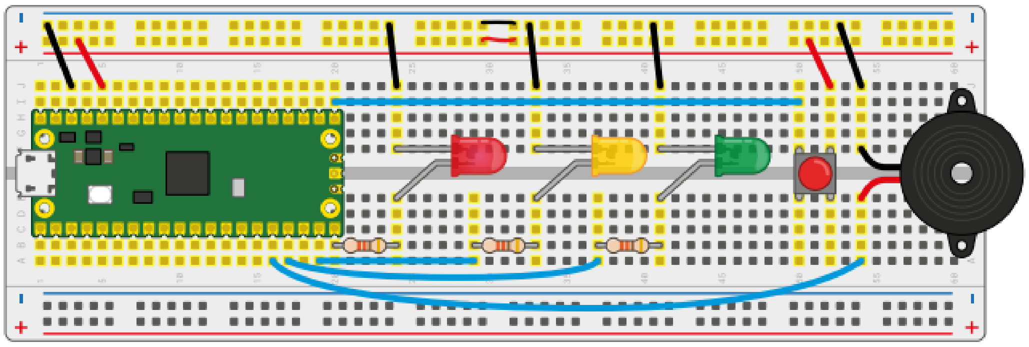
Code9: Programmable I/O (PIO) to set Pico’s onboard LED to quarter, half and full brightness

Working: Each PIO instruction takes exactly one clock cycle to run (by default)

- led_quarter_brightness – 2 set instructions + 2 delays = 4 clock cycles (Number of instructions indicating ‘off ’ = 3 and Number of instructions indicating ‘on’ = 1, therefore quarter brightness)
- led_half_brightness – 2 set instructions = 2 clock cycles (1 instruction indicates ‘off ’ and the other ‘on ’, therefore half brightness)
- led_full_brightness – 1 set instruction = 1 clock cycle, indicating ‘on’

```python
from rp2 import PIO, StateMachine, asm_pio
from machine import Pin
import time
led=Pin(25)

@asm_pio(set_init=PIO.OUT_LOW)
def led_quarter_brightness():
    set(pins, 0) [2]
    set(pins, 1)

@asm_pio(set_init=PIO.OUT_LOW)
def led_half_brightness():
    set(pins, 0)
    set(pins, 1)

@asm_pio(set_init=PIO.OUT_HIGH)
def led_full_brightness():
    set(pins, 1)

# parameters of StateMachine:
# 1. The state machine number(0-7)
# 2. The PIO program to load
# 3. The frequency (which must be between 2000 and 125000000)
# 4. The GPIO pin that the state machine manipulates
sm1 = StateMachine(1, led_quarter_brightness, freq=10000, set_base=led)
sm2 = StateMachine(2, led_half_brightness, freq=10000, set_base=led)
sm3 = StateMachine(3, led_full_brightness, freq=10000, set_base=led)

while(True):
    sm1.active(1)
    time.sleep(1)
    sm1.active(0)

    sm2.active(1)
    time.sleep(1)
    sm2.active(0)

    sm3.active(1)
    time.sleep(1)
    sm3.active(0)
```

Code10: Traffic light controller using RP2040's 2 cores



```
import machine
import utime
import _thread

led_red = machine.Pin(15,machine.Pin.OUT)
led_yellow = machine.Pin(14,machine.Pin.OUT)
led_green = machine.Pin(13,machine.Pin.OUT)
button = machine.Pin(16,machine.Pin.IN,machine.Pin.PULL_DOWN)
buzzer = machine.Pin(12,machine.Pin.OUT)

global button_press #global variable is used so that it's accessible throughout
the program
button_press=False #initializing that the button hasn't yet been presssed

def button_reader_thread():
    global button_press
    while True:
        if button.value()==1:
            button_press=True #identifying the button's input and modifying
global variable

_thread.start_new_thread(button_reader_thread,()) #the thread defined will
check the button's input, without disturbing the execution of main part of the
program

while True:
    if button_press==True:
        led_red.value(1) # when button_press is True, red LED is turned on with
 toggling of buzzer
```

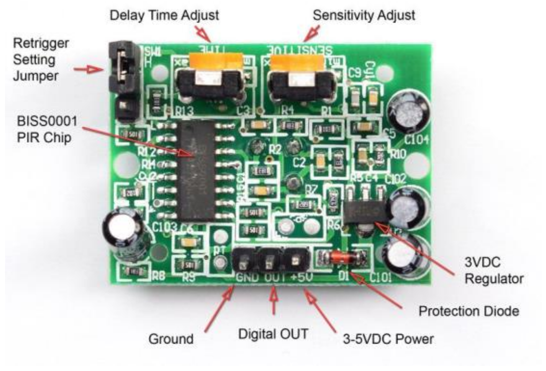
```
        for i in range(5):
            buzzer.value(1)
            utime.sleep(0.2)
            buzzer.value(0)
            utime.sleep(0.2)
        global button_press
        button_press=False # button_press is reset to False until next input
from button is received

    #execution of traffic lights using red, yellow, green LEDs
    led_red.value(1)
    utime.sleep(5)
    led_red.value(0)
    led_yellow.value(1)
    utime.sleep(2)
    led_yellow.value(0)
    led_green.value(1)
    utime.sleep(5)
    led_green.value(0)
    led_yellow.value(1)
    utime.sleep(2)
    led_yellow.value(0)
```

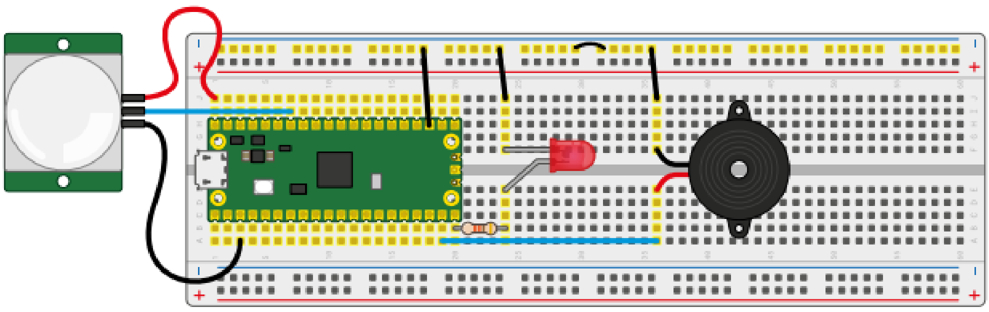
Additional code:

<u>Code11</u>: Burglar alarm using Passive Infrared (PIR) Sensor – Based on Interrupts

<u>PIR sensor</u>

- PIR Sensor recognizes infrared light emitted from nearby objects.
- The actual sensor is buried underneath a plastic lens, which provides wider field of vision (FOV).
- Specifications of PIR sensor
    - I. BISS0001 PIR motion detector IC
    - II. Input voltage: 4.5-20V DC
    - III. Sensing range: 3-10 meters
    - IV. Delay time: 5-200 seconds
    - V. Operating temperature: -20^o to 80^o C
    - VI. 3-pin ground/output/power pads



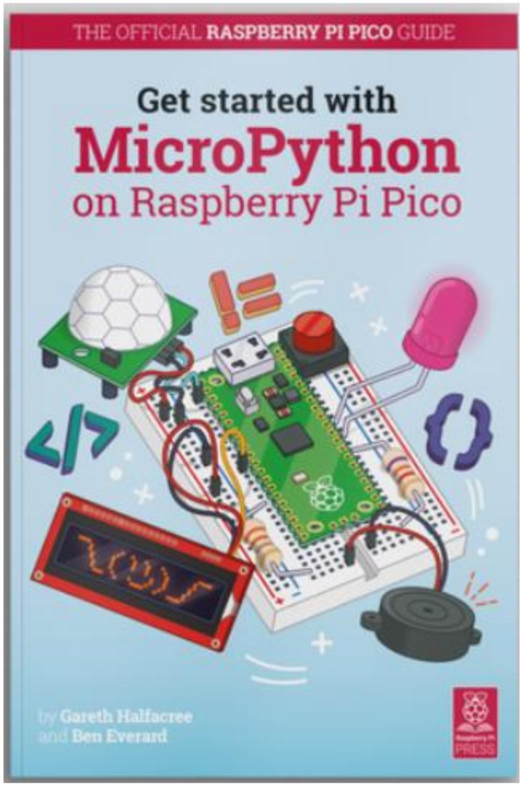


```
# Toggling rate of LED increases along with toggling of
# buzzer(alarm) when PIR detects motion
from machine import Pin
import utime

sensor_pir = Pin(28,Pin.IN,Pin.PULL_DOWN)
led = Pin(15,Pin.OUT)
buzzer = Pin(14,Pin.OUT)

def pir_handler(pin):
    print("Motion detected!")
    for i in range(20):
        led.toggle()
        buzzer.toggle()
        utime.sleep_ms(100)

sensor_pir.irq(trigger=Pin.IRQ_RISING,handler=pir_handler)

while True:
    led.toggle()
    utime.sleep(5)
```

Pico references:

- RP2040 Datasheet
- Raspberry Pi Pico Python SDK
- For further references, you can visit Raspberry Pi Documentation
- For MicroPython (machine module), you can visit here
- For project ideas, you can refer Magpi and Hackspace magazines

YOUR OFFICIAL RASPBERRY PI MAGAZINE
The MagPi
Issue 109 | September 2021 | magpi.cc
The official Raspberry Pi magazine
Raspberry Pi 4 SID Synth
Upcycling a Game Boy
JOYSTICK
RASPBERRY PI 20 PICO PROJECTS
Make incredible objects with microcontroller boards!
MIDI FIGHTER
PICO ROBOT
Raspberry Pi Student Guide
42 PAGES OF PROJECTS & TUTORIALS


THE OFFICIAL RASPBERRY PI PICO GUIDE
Get started with MicroPython on Raspberry Pi Pico
by Gareth Halfacree and Ben Everard
Raspberry Pi PRESS