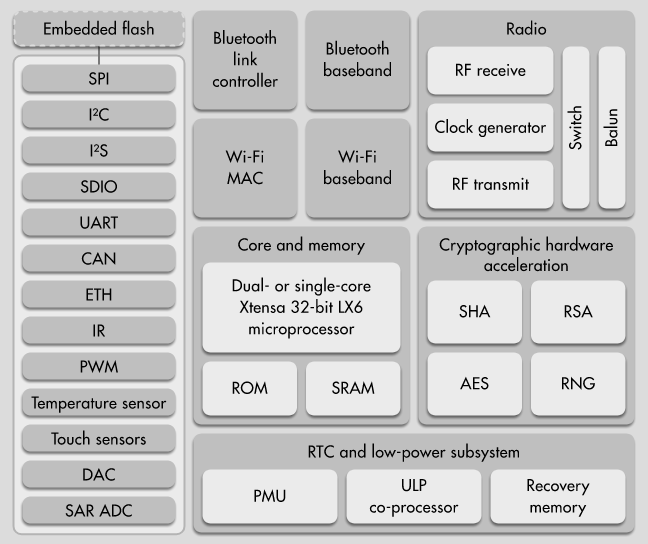
**ESP32**



**Séria ESP32-SOLO**

Modul ESP32-SOLO-1 je založený na ESP32-S0WD a má integrovanú pamäť FLASH, čo predstavuje cenovo výhodné riešenie pre jednoduché aplikácie pripojenia založené na Wi-Fi a Bluetooth/Bluetooth LE.

**Séria ESP32-WROOM**

Jedná sa o moduly založené na ESP32-D0WD s integrovanou pamäťou FLASH. Tieto moduly sú vhodné pre aplikácie konektivity založené na Wi-Fi a Bluetooth/Bluetooth LE. Na dvoch jadrách poskytujú solídny výkon.

**Séria ESP32-WROVER**

Séria ESP32-WROVER je založená na SoC ESP32-D0WD a má tiež integrovanú pamäť FLASH a SPIRAM. Majú dve jadrá a dobrý výkon. Sú vhodné pre aplikácie vyžadujúce viac pamäte, ako sú aplikácie AIoT a GW.

Modul ESP32 má predinštalovaný binárny obraz ROM so sadou príkazov AT, rovnako ako to bolo v ESP8266. Ak chcete minúť 95% potenciálu tohto čipu tým, že ho použijete ako modem sériový-WiFi, ak máte všetko!

**Rýchly sprievodca micropythonom ESP32**

<https://docs.micropython.org/en/latest/esp32/quickref.html>

<https://www.dfrobot.com/blog-1569.html>

Odkazy:

* [General information about the ESP32 port](https://docs.micropython.org/en/latest/esp32/general.html)
* [MicroPython tutorial for ESP32](https://docs.micropython.org/en/latest/esp32/tutorial/index.html)

Inštalácia programu MicroPython

Pozrite si príslušnú časť tutoriálu: [Getting started with MicroPython on the ESP32](https://docs.micropython.org/en/latest/esp32/tutorial/intro.html#esp32-intro). Obsahuje tiež podsekciu na riešenie problémov.

**Všeobecné ovládanie dosky**

MicroPython REPL (skratka REPL je skratka pre read-eval-print loop a v zásade poskytuje programátorovi interaktívne programovacie prostredie) využíva UART0 (GPIO1 = TX, GPIO3 = RX) s modulačnou rýchlosťou 115200 Bd. Tab-completion (dokončenie príkazového riadka (tiež dopĺňanie záložiek) je bežnou vlastnosťou interpreterov príkazového riadku – program automaticky dopĺňa čiastočne zadané príkazy) ak chceme zistiť aké metódy má objekt. Režim prilepenia (ctrl-E) je užitočný na vloženie veľkej časti kódu Python do REPL.

**Modul „machine“:**

import machine

machine.freq() # get the current frequency of the CPU

machine.freq(240000000) # set the CPU frequency to 240 MHz

**Modul „esp“:**

import esp

esp.osdebug(None) # turn off vendor O/S debugging messages

esp.osdebug(0) # redirect vendor O/S debugging messages to UART(0)

# low level methods to interact with flash storage

esp.flash\_size()

esp.flash\_user\_start()

esp.flash\_erase(sector\_no)

esp.flash\_write(byte\_offset, buffer)

esp.flash\_read(byte\_offset, buffer)

**Modul „esp32“:**

import esp32

esp32.hall\_sensor() # read the internal hall sensor

esp32.raw\_temperature() # read the internal temperature of the MCU, in Fahrenheit

esp32.ULP() # access to the Ultra-Low-Power Co-processor

*Všimnite si, že snímač teploty v ESP32 bude zvyčajne čítať vyššie ako okolité teploty, pretože sa IC počas prevádzky zahrieva. Tento efekt je možné minimalizovať odčítaním teplotného senzora bezprostredne po prebudení zo spánku.*

**Modul „Network“:**

import network

wlan = network.WLAN(network.STA\_IF) # create station interface

wlan.active(True) # activate the interface

wlan.scan() # scan for access points

wlan.isconnected() # check if the station is connected to an AP

wlan.connect('essid', 'password') # connect to an AP

wlan.config('mac') # get the interface's MAC address

wlan.ifconfig() # get the interface's IP/netmask/gw/DNS addresses

ap = network.WLAN(network.AP\_IF) # create access-point interface

ap.config(essid='ESP-AP') # set the ESSID of the access point

ap.config(max\_clients=10) # set how many clients can connect to the network

ap.active(True) # activate the interface

Užitočné funkcie na pripojenie k miestnej sieti WiFi je:

def do\_connect():

import network

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

if not wlan.isconnected():

print('connecting to network...')

wlan.connect('essid', 'password')

while not wlan.isconnected():

pass

print('network config:', wlan.ifconfig())

Akonáhle je sieť vytvorená, modul „socket“ možno použiť na vytváranie a používanie soketov TCP/UDP ako obvykle a modul „urequests“ na požiadavky HTTP.

Po zavolaní na wlan.connect () sa zariadenie predvolene pokúsi trvale pripojiť, aj keď autentifikácia zlyhala alebo nie je v dosahu žiadny prístupový bod. wlan.status () vráti v tomto stave network.STAT\_CONNECTING, kým sa pripojenie nepodarí alebo kým sa rozhranie nevypne. Toto je možné zmeniť zavolaním na wlan.config (reconnects = n), kde n je počet požadovaných pokusov o opätovné pripojenie (0 znamená, že sa to nebude opakovať, -1 obnoví predvolené správanie pri pokuse o trvalé pripojenie).

**Oneskorenie a časovanie – Modul „time“**

import time

time.sleep(1) # sleep for 1 second

time.sleep\_ms(500) # sleep for 500 milliseconds

time.sleep\_us(10) # sleep for 10 microseconds

start = time.ticks\_ms() # get millisecond counter

delta = time.ticks\_diff(time.ticks\_ms(), start) # compute time difference

ESP32 má štyri hardvérové časovače. Používajte [machine.Timer](https://docs.micropython.org/en/latest/library/machine.Timer.html#machine-timer) triedu s časovačom s ID od 0 do 3 (vrátane):

from machine import Timer

tim0 = Timer(0)

tim0.init(period=5000, mode=Timer.ONE\_SHOT, callback=lambda t:print(0))

tim1 = Timer(1)

tim1.init(period=2000, mode=Timer.PERIODIC, callback=lambda t:print(1))

Perióda je v milisekundách.

Virtuálne časovače nie sú momentálne podporované.

## Vývody a GPIO

Použi triedu „[machine.Pin](https://docs.micropython.org/en/latest/library/machine.Pin.html#machine-pin)“

from machine import Pin

p0 = Pin(0, Pin.OUT) # create output pin on GPIO0

p0.on() # set pin to "on" (high) level

p0.off() # set pin to "off" (low) level

p0.value(1) # set pin to on/high

p2 = Pin(2, Pin.IN) # create input pin on GPIO2

print(p2.value()) # get value, 0 or 1

p4 = Pin(4, Pin.IN, Pin.PULL\_UP) # enable internal pull-up resistor

p5 = Pin(5, Pin.OUT, value=1) # set pin high on creation

Dostupné vývody sú z nasledujúcich rozsahov (vrátane): 0-19, 21-23, 25-27, 32-39. Tieto zodpovedajú skutočnému počtu vývodov GPIO na čipe ESP32. Všimnime si, že mnohé dosky koncových používateľov používajú svoje vlastné číslovanie vývodov adhoc (označené napr. D0, D1, ...). Informácie o mapovaní medzi logickými vývodmi na doske a fyzickými vývodmi nájdete v dokumentácii ku konkrétnej doske.

Poznámky:

Vývody 1 a 3 sú REPL UART TX a RX

Vývody 6, 7, 8, 11, 16 a 17 slúžia na pripojenie vstavanej Flash pamäte a neodporúčajú sa na iné použitie.

Vývody 34-39 sú len vstupné a nemajú vnútorné pull-up odpory

Hodnotu záťaže niektorých vývodov je možné nastaviť na Pin.PULL\_HOLD, aby sa znížila spotreba energie počas hlbokého spánku.

Existuje [machine.Signal](https://docs.micropython.org/en/latest/library/machine.Signal.html#machine-signal) vyššej úrovne. Signál, ktorý je možné použiť na invertovanie vývodu. To je užitočné na ovládanie aktívnych LED diód on () alebo hodnotou (1).

## UART (sériové rozhranie)

Použi triedu „[machine.UART](https://docs.micropython.org/en/latest/library/machine.Pin.html#machine-pin)“

from machine import UART

uart1 = UART(1, baudrate=9600, tx=33, rx=32)

uart1.write('hello') # write 5 bytes

uart1.read(5) # read up to 5 bytes

ESP32 má tri hardvérové UART: UART0, UART1 a UART2. Každý z nich má priradené predvolené GPIO, avšak v závislosti od variantu a dosky ESP32 môžu byť tieto vývody v konflikte so vstavanou FLASH, vstavanou PSRAM alebo inými perifériami.

Na hardvérové UARTy pomocou GPIO matice je možné použiť akékoľvek GPIO, aby sa predišlo konfliktom, pri návrhu jednoducho priraďte vývody tx a rx. Predvolené vývody sú uvedené nižšie.

|  | UART0 | UART1 | UART2 |
| --- | --- | --- | --- |
| tx | 1 | 10 | 17 |
| rx | 3 | 9 | 16 |

**PWM**

PWM je možné povoliť na všetkých vývodoch s povoleným výstupom. Základná frekvencia sa môže pohybovať od 1 Hz do 40 MHz. Zo zvyšujúcou základnou frekvenciou sa znižuje rozlíšenie PWM. Ďalšie podrobnosti nájdete v časti Ovládanie LED. V súčasnosti sa musí pracovný cyklus pohybovať v rozmedzí 0-1023.

**Trieda machine.PWM**

from machine import Pin, PWM

pwm0 = PWM(Pin(0)) # create PWM object from a pin

pwm0.freq() # get current frequency (default 5kHz)

pwm0.freq(1000) # set frequency

pwm0.duty() # get current duty cycle (default 512, 50%)

pwm0.duty(200) # set duty cycle

pwm0.deinit() # turn off PWM on the pin

pwm2 = PWM(Pin(2), freq=20000, duty=512) # create and configure in one go

Čipy ESP majú rôzne hardvérové periférie:

|  |  |  |  |
| --- | --- | --- | --- |
| Hardware specification | ESP32 | ESP32-S2 | ESP32-C3 |
| Number of groups (speed modes) | 2 | 1 | 1 |
| Number of timers per group | 4 | 4 | 4 |
| Number of channels per group | 8 | 8 | 6 |
| Different of PWM frequencies (groups \* timers) | 8 | 4 | 4 |
| Total PWM channels (Pins, duties) (groups \* channels) | 16 | 8 | 6 |

Na ESP32 je k dispozícii maximálny počet PWM kanálov (vývodov) - 16, pričom je k dispozícii iba 8 rôznych frekvencií PWM, zvyšných 8 kanálov musí mať rovnakú frekvenciu. Na druhej strane je možné mať 16 nezávislých pracovných cyklov PWM s rovnakou frekvenciou.

**ADC (AD prevodník)**

Na ESP32 je funkcia ADC dostupná na vývodoch 32-39. Upozorňujeme, že pri použití predvolenej konfigurácie musí byť vstupné napätie na vývode ADC medzi 0,0 V a 1,0 V (čokoľvek nad 1,0 V sa bude čítať len ako 4095). Aby sa zvýšil použiteľný rozsah napätia je potrebné použiť útlm adc.atten( ) .

Použite triedy **machine.ADC**:

from machine import ADC

adc = ADC(Pin(32)) # create ADC object on ADC pin

adc.read() # read value, 0-4095 across voltage range 0.0v-1.0v

adc.atten(ADC.ATTN\_11DB) # set 11dB input attenuation (voltage range roughly 0.0v - 3.6v)

adc.width(ADC.WIDTH\_9BIT) # set 9 bit return values (returned range 0-511)

adc.read() # read value using the newly configured attenuation and width

Odkaz na metódu triedy ADC špecifický pre ESP32:

**ADC.atten(*attenuation*)**

Táto metóda umožňuje nastavenie veľkosti útlmu na vstupe ADC. To umožňuje rozšíriť rozsah vstupného napätia za cenu presnosti (rovnaký počet bitov teraz predstavuje širší rozsah). Možné možnosti útlmu sú:

* ADC.ATTN\_0DB: 0dB útlm, maximálne vstupné napätie je približne of 1.00V, pôvodná, prednastavená konfigurácia
* ADC.ATTN\_2\_5DB: 2.5dB maximálne vstupné napätie je približne 1.34V
* ADC.ATTN\_6DB: maximálne vstupné napätie je približne 2.0V
* ADC.ATTN\_11DB: 11dB, maximálne vstupné napätie je približne 3.6V

**Pozor**

Napriek útlmu 11 dB, ktorý umožňuje rozsah až 3,6 V, si všimnite, že absolútne maximálne napätie pre vstupné vývody je 3,6 V, takže prekročenie tejto hranice môže poškodiť IC!

**ADC.width(*width*)**

Táto metóda umožňuje nastavenie počtu bitov, ktoré sa majú použiť a vrátiť počas čítania ADC. Použiteľné možnosti sú:

* ADC.WIDTH\_9BIT: 9 bit data
* ADC.WIDTH\_10BIT: 10 bit data
* ADC.WIDTH\_11BIT: 11 bit data
* ADC.WIDTH\_12BIT: 12 bit data – toto je prednastavená konfigurácia

**Softvérová zbernica SPI**

Softvérové SPI (pomocou bit-bangingu) funguje na všetkých vývodoch a je prístupná cez triedu **machine.SoftSPI:**

from machine import Pin, SoftSPI

# construct a SoftSPI bus on the given pins

# polarity is the idle state of SCK

# phase=0 means sample on the first edge of SCK, phase=1 means the second

spi = SoftSPI(baudrate=100000, polarity=1, phase=0, sck=Pin(0), mosi=Pin(2), miso=Pin(4))

spi.init(baudrate=200000) # set the baudrate

spi.read(10) # read 10 bytes on MISO

spi.read(10, 0xff) # read 10 bytes while outputting 0xff on MOSI

buf = bytearray(50) # create a buffer

spi.readinto(buf) # read into the given buffer (reads 50 bytes in this case)

spi.readinto(buf, 0xff) # read into the given buffer and output 0xff on MOSI

spi.write(b'12345') # write 5 bytes on MOSI

buf = bytearray(4) # create a buffer

spi.write\_readinto(b'1234', buf) # write to MOSI and read from MISO into the buffer

spi.write\_readinto(buf, buf) # write buf to MOSI and read MISO back into buf

Pri inicializácii Software SPI musia byť špecifikované všetky vývody pre sck, mosi a miso.

**Hardvérová zbernica SPI**

Existujú dva hardvérové kanály SPI, ktoré umožňujú rýchlejšie prenosové rýchlosti (až 80 MHz). Môžu byť použité na akýchkoľvek IO vývodoch, ktoré podporujú požadovaný smer a inak sa nepoužívajú (pozri Piny a GPIO), ale ak nie sú nakonfigurované na svoje predvolené vývody, musia prejsť ďalšou vrstvou multiplexovania GPIO, čo môže ovplyvniť ich spoľahlivosť pri vysokých rýchlostiach. Hardvérové kanály SPI sú obmedzené na 40 MHz, ak sa používajú na iných vývodoch, ako sú predvolené nižšie uvedené.

HSPI (id=1) VSPI (id=2)

Sck 14 18

Mosi 13 23

Miso 12 19

Hardvérové SPI je prístupné cez triedu machine.SPI a má rovnaké metódy ako softvérové SPI vyššie:

from machine import Pin, SPI

hspi = SPI(1, 10000000)

hspi = SPI(1, 10000000, sck=Pin(14), mosi=Pin(13), miso=Pin(12))

vspi = SPI(2, baudrate=80000000, polarity=0, phase=0, bits=8, firstbit=0, sck=Pin(18), mosi=Pin(23), miso=Pin(19))

**Softvérová zbernica I2C**

Softvérová I2C (používajúci bit-banging) funguje na všetkých vývodoch s výstupom a je prístupná cez triedu **machine.SoftI2C:**

from machine import Pin, SoftI2C

i2c = SoftI2C(scl=Pin(5), sda=Pin(4), freq=100000)

i2c.scan() # scan for devices

i2c.readfrom(0x3a, 4) # read 4 bytes from device with address 0x3a

i2c.writeto(0x3a, '12') # write '12' to device with address 0x3a

buf = bytearray(10) # create a buffer with 10 bytes

i2c.writeto(0x3a, buf) # write the given buffer to the peripheral

**Hardvérová I2C zbernica**

Existujú dve hardvérové periférie I2C s identifikátormi 0 a 1. Pre SCL a SDA možno použiť akékoľvek dostupné vývody s výstupom, ale predvolené hodnoty sú uvedené nižšie.

**I2C(0) I2C(1)**

scl 18 25

sda 19 26

Ovládač je prístupný cez triedu machine.I2C a má rovnaké metódy ako softvérová I2C vyššie:

from machine import Pin, I2C

i2c = I2C(0)

i2c = I2C(1, scl=Pin(5), sda=Pin(4), freq=400000)

**Zbernica I2S**

I2S je synchrónny sériový protokol používaný na pripojenie digitálnych audio zariadení. Na fyzickej úrovni zbernica pozostáva z 3 liniek: SCK, WS, SD. Trieda I2S podporuje činnosť ovládača. Operácie periférnych zariadení nie sú podporované.

Trieda I2S je momentálne k dispozícii ako beta verzia. Počas tohto obdobia sa očakáva spätná väzba od používateľov. Na základe tejto spätnej väzby sa API triedy I2S a ich implementácia môže zmeniť.

Objekty I2S je možné vytvoriť a inicializovať pomocou:

from machine import I2S

from machine import Pin

# ESP32

sck\_pin = Pin(14) # Serial clock output

ws\_pin = Pin(13) # Word clock output

sd\_pin = Pin(12) # Serial data output

or

# PyBoards

sck\_pin = Pin("Y6") # Serial clock output

ws\_pin = Pin("Y5") # Word clock output

sd\_pin = Pin("Y8") # Serial data output

audio\_out = I2S(2,

sck=sck\_pin, ws=ws\_pin, sd=sd\_pin,

mode=I2S.TX,

bits=16,

format=I2S.MONO,

rate=44100,

ibuf=20000)

audio\_in = I2S(2,

sck=sck\_pin, ws=ws\_pin, sd=sd\_pin,

mode=I2S.RX,

bits=32,

format=I2S.STEREO,

rate=22050,

ibuf=20000)

ESP32 má dve zbernice I2S s id=0 a id=1

Podporované sú 3 režimy prevádzky:

blokovanie

neblokovanie

uasyncio

**blocking:**

num\_written = audio\_out.write(buf) # blocks until buf emptied

num\_read = audio\_in.readinto(buf) # blocks until buf filled

**non-blocking**:

audio\_out.irq(i2s\_callback) # i2s\_callback is called when buf is emptied

num\_written = audio\_out.write(buf) # returns immediately

audio\_in.irq(i2s\_callback) # i2s\_callback is called when buf is filled

num\_read = audio\_in.readinto(buf) # returns immediately

**uasyncio:**

swriter = uasyncio.StreamWriter(audio\_out)

swriter.write(buf)

await swriter.drain()

sreader = uasyncio.StreamReader(audio\_in)

num\_read = await sreader.readinto(buf)

**Konštruktor**

class machine.I2S(id, \*, sck, ws, sd, režim, bity, formát, rýchlosť, ibuf)

**Zostavte I2S objekt daného id:**

id identifikuje konkrétnu zbernicu I2S.

id je špecifické pre dosku a port:

PYBv1.0/v1.1: má jednu zbernicu I2S s id=2.

PYBD-SFxW: má dve zbernice I2S s id=1 a id=2.

ESP32: má dve zbernice I2S s id=0 a id=1.

**Parametre iba pre kľúčové slová, ktoré sú podporované na všetkých portoch:**

sck je objekt pin pre sériovú hodinovú linku

ws je objekt špendlíka pre riadok výberu slova

sd je objekt pin pre sériovú dátovú linku

režim určuje príjem alebo vysielanie

bitov určuje veľkosť vzorky (bity), 16 alebo 32

format určuje formát kanálu, STEREO alebo MONO

rýchlosť určuje vzorkovaciu frekvenciu zvuku (vzorky/s)

ibuf určuje dĺžku vnútornej vyrovnávacej pamäte (bajty)

Pre všetky porty DMA beží nepretržite na pozadí a umožňuje užívateľským aplikáciám vykonávať iné operácie, zatiaľ čo sa dáta prenášajú medzi internou vyrovnávacou pamäťou a periférnou jednotkou I2S. Zväčšenie veľkosti internej vyrovnávacej pamäte umožňuje predĺžiť čas, počas ktorého môžu používateľské aplikácie vykonávať operácie iné ako I2S pred „podtečením“ (napr. metóda zápisu) alebo „pretečením“ (napr. metódou readinto).

## Metódy

I2S.init(sck, ...)

popis argumentov nájdete v časti Konštruktor

I2S.deinit()

Deinicializácia zbernice I2S

I2S.readinto(*buf*)

Číta zvukové vzorky do vyrovnávacej pamäte špecifikovanej v buf. buf musí podporovať protokol vyrovnávacej pamäte, ako napríklad bytearray alebo pole. Usporiadanie bajtov „buf“ je little-endian. Pre stereo formát vzorka ľavého kanála predchádza vzorke pravého kanála. Pre mono formát sú použité vzorky ľavého kanála. Vráti počet prečítaných bajtov

I2S.write(*buf*)

Zapíše zvukové vzorky obsiahnuté v buf. buf musí podporovať protokol vyrovnávacej pamäte, ako napríklad bytearray alebo pole. Usporiadanie bajtov „buf“ je little-endian. Pre stereo formát vzorka ľavého kanála predchádza vzorke pravého kanála. V prípade formátu Mono sa vzorové údaje zapisujú do pravého aj ľavého kanála. Vráti počet zapísaných bajtov.

I2S.irq(*handler*)

Nastaví callback. handler sa volá, keď sa buf vyprázdni (metóda zápisu) alebo sa zaplní (metóda readinto). Nastavenie callback zmení metódy zápisu a čítania na neblokujúcu operáciu. handler sa volá v kontexte plánovača MicroPythonu.

static I2S.shift(*\**, *buf*, *bits*, *shift*)

bitový posun všetkých vzoriek obsiahnutých v buf. bits určuje veľkosť vzorky v bitoch. shift určuje počet bitov na posunutie každej vzorky. Kladný pre posun vľavo, záporný pre posun vpravo. Zvyčajne sa používa na ovládanie hlasitosti. Každý bitový posun zmení hlasitosť vzorky o 6 dB.

**Konštanty**

I2S.RX

Inicializácia I2S bus mode príjem

I2S.TX

Inicializácia I2S bus mode vysielanie

I2S.STEREO

Inicializácia I2S bus format stereo

I2S.MONO

Inicializácia I2S bus format mono

## Real time clock (RTC)

See [machine.RTC](https://docs.micropython.org/en/latest/library/machine.RTC.html#machine-rtc)

from machine import RTC

rtc = RTC()

rtc.datetime((2017, 8, 23, 1, 12, 48, 0, 0)) # set a specific date and time

rtc.datetime() # get date and time

## WDT (Watchdog timer)

See [machine.WDT](https://docs.micropython.org/en/latest/library/machine.WDT.html#machine-wdt).

from machine import WDT

# enable the WDT with a timeout of 5s (1s is the minimum)

wdt = WDT(timeout=5000)

wdt.feed()

## Deep-sleep mode

The following code can be used to sleep, wake and check the reset cause:

import machine

# check if the device woke from a deep sleep

if machine.reset\_cause() == machine.DEEPSLEEP\_RESET:

print('woke from a deep sleep')

# put the device to sleep for 10 seconds

machine.deepsleep(10000)

Poznámky:

Volanie *deepsleep(*) bez argumentu uspí zariadenie na neurčito

Softvérový reset nezmení príčinu resetovania

Cez povolené vnútorné pull-up odpory môže pretekať nejaký zvodový prúd. Pre ďalšie zníženie spotreby energie je možné deaktivovať interné pull-up:

p1 = Pin(4, Pin.IN, Pin.PULL\_HOLD)

Po opustení hlbokého spánku môže byť potrebné explicitne uvoľniť vývod (napr. ak ide o výstupný kolík) pomocou:

p1 = Pin(4, Pin.OUT, None)

## SD card

See [machine.SDCard](https://docs.micropython.org/en/latest/library/machine.SDCard.html#machine-sdcard).

import machine, os

# Slot 2 uses pins sck=18, cs=5, miso=19, mosi=23

sd = machine.SDCard(slot=2)

os.mount(sd, "/sd") # mount

os.listdir('/sd') # list directory contents

os.umount('/sd') # eject

RMT

RMT je špecifický pre ESP32 a umožňuje generovanie presných digitálnych impulzov s rozlíšením 12,5 ns. Podrobnosti nájdete v esp32.RMT. Použitie je:

import esp32

from machine import Pin

r = esp32.RMT(0, pin=Pin(18), clock\_div=8)

r # RMT(channel=0, pin=18, source\_freq=80000000, clock\_div=8)

# The channel resolution is 100ns (1/(source\_freq/clock\_div)).

r.write\_pulses((1, 20, 2, 40), start=0) # Send 0 for 100ns, 1 for 2000ns, 0 for 200ns, 1 for 4000ns

**Ovládač OneWire**

Ovládač OneWire je implementovaný v softvéri a funguje na všetkých vývodoch:

from machine import Pin

import onewire

ow = onewire.OneWire(Pin(12)) # create a OneWire bus on GPIO12

ow.scan() # return a list of devices on the bus

ow.reset() # reset the bus

ow.readbyte() # read a byte

ow.writebyte(0x12) # write a byte on the bus

ow.write('123') # write bytes on the bus

ow.select\_rom(b'12345678') # select a specific device by its ROM code

Pre zariadenia DS18S20 a DS18B20 existuje špecifický ovládač:

import time, ds18x20

ds = ds18x20.DS18X20(ow)

roms = ds.scan()

ds.convert\_temp()

time.sleep\_ms(750)

for rom in roms:

print(ds.read\_temp(rom))

Uistite sa, že ste dátovú linku ošetrili 4,7k pull-up rezistorom. Všimnite si, že metóda convert\_temp() sa musí volať zakaždým, keď chcete vzorkovať teplotu.