Podstawy Internetu Rzeczy
Instrukcja do zajęć laboratoryjnych

Laboratorium 9

Urządzenia peryferyjne i komunikacja z nimi
Część 1

Opracował opiekun przedmiotu dr inż. Krzysztof Chudzik

Wydział Informatyki i Telekomunikacji
Politechnika Wrocławska

Wrocław, 2025.09.30 18:50:06

Spis treści

1 Porty I/O w Raspberry Pi **1**
1.1 Napięciowe poziomy logiczne i konwerter poziomów logicznych . . . 2
1.2 Piny portów I/O . . . 2
1.3 Pomocniczy plik konfiguracji zestawu `config.py` . . . 3
1.4 Konfiguracja trybów pracy portów I/O i sterowanie portami . . . 3
1.4.1 Porty w trybie wyjścia - sterowanie diodami LED i buzzerem. . . . 4
1.4.2 Porty w trybie wejścia - odczyt stanów przełączników przyciskanych i enkodera . . . 4
1.4.3 Sygnał PWM . . . 5

2 Programowalne diody LED WS2812 **6**

3 Czujniki DS18B20 oraz BME280 **7**

Lista zadań

1 Regulacja jasności świecenia diody enkoderem . . . 6
2 Odczyt parametrów środowiskowych z czujników DS18B20 oraz BME280 i ich wizualizacja poprzez diody WS2812 . . . 8

Zanim przystąpimy do konfigurowania i programowania, pamiętajmy, że z tych samych zestawów korzysta wielu Studentów. Kolejne instrukcje laboratoryjne przygotowane są z założeniem, że zestawy posiadają oryginalną konfigurację laboratoryjną. **Zabrania się wprowadzania trwałych zmian konfiguracyjnych systemu operacyjnego oraz jakiegokolwiek oprogramowania w sposób zmieniający jego działanie, w tym działanie interfejsów graficznych. Zmiany takie mogą uniemożliwić prawidłowe przeprowadzenie kolejnych zajęć.** Proszę umożliwić innym Studentom odbycie zajęć, tak jak zostały one zaplanowane. Kto nie zastosuje się do tego i będzie dezorganizował zajęcia, będzie traktowany jako uszkadzający zestawy i zostanie odsunięty od zajęć.

1 Porty I/O w Raspberry Pi

Porty I/O, podobnie jak w zestawach Arduino, umożliwiają interakcję z układami zewnętrznymi, reagowanie na sygnały wysyłane przez te układy oraz sterowanie nimi z wykorzystaniem sygnałów logicznych.

1.1 Napięciowe poziomy logiczne i konwerter poziomów logicznych

Raspberry Pi korzysta z logiki o napięciu 3,3V. Płytka Arduino korzysta z logiki o napięciu 5V. Istnieje wiele popularnych modułów elektronicznych z logiką o napięciu 5V, które chcielibyśmy wykorzystać w naszych układach. Musimy pamiętać, że nie wolno łączyć bezpośrednio logik o różnych napięciach, gdyż może to prowadzić do nieodwracalnego uszkodzenia układów elektronicznych. Aby połączyć logiki o różnych napięciach korzysta się z konwerterów poziomów logicznych.

W zestawie laboratoryjnym z Raspberry Pi wykorzystywane sa dwa układy o logice napięciowej 5V. Są to: sygnalizator dźwiękowy (*buzzer*) oraz linijka diod programowalnych RGB WS2812. Ich wykorzystanie wymagało zamontowania w zestawie laboratoryjnym konwertera stanów logicznych, który umożliwia bezpieczne połączenie układów o różnych logikach napięciowych, to znaczy 3,3V i 5V. Posiada on cztery dwukierunkowe kanały. Opis można znaleźć na stronie producenta poświęconej konwerterowi (link). Podano tam informacje o jego budowie, sposobie wykorzystania, i jako przykład użycia podano realizację połączenia łączem szeregowym pomiędzy Raspberry Pi i Arduino UNO.

1.2 Piny portów I/O

Piny, które umożliwiają przyłączanie urządzeń peryferyjnych, udostępnione są w na płytce Raspberry Pi w postaci 40 pinowego złącza. Wiele przydatnych informacji na jego temat znajduje się na stronie Raspberry Pi Pinout (link).

Najprostszą metodą uzyskania mapy pinów jest wydanie w konsoli tekstowej polecenia `pinout`, tak jak na widoku ekranu 1.

```
pi@raspberrypi:~ $ pinout
,--------------------------------.
| oo ooo oo oooo o ooo J8   +======
| looo ooooooo ooooo   PoE |   Net
|  Wi                      lo +======
|  Fi  Pi Model 4B  V1.1 oo    |
|       ,----.  +---+           +====
|  |D|  |SoC |  |RAM|           |USB3
|  |S|  |    |  |   |           +====
|  |I|  `----'  +---+             |
|                  |C|           +====
|                  |S|           |USB2
|  pwr   |hd|   |hd| |I||A|     +====
`-| |----|m0|---|m1|-----|V|-------'

Revision          : c03111
SoC               : BCM2711
RAM               : 4GB
Storage           : MicroSD
USB ports         : 4 (of which 2 USB3)
Ethernet ports    : 1 (1000Mbps max. speed)
Wi-fi             : True
Bluetooth         : True
Camera ports (CSI) : 1
Display ports (DSI): 1

J8:
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21

POE:
TR01 (1) (2) TR00
TR03 (3) (4) TR02

For further information, please refer to https://pinout.xyz/
pi@raspberrypi:~ $
```

Ekran 1: Mapa pinów Raspberry Pi uzyskana poleceniem `pinout`.

Pod schematycznym rysunkiem płytki Raspberry Pi, z zaznaczonymi podstawowymi elementami funkcjonalnymi, znajduje się podstawowa charakterystyka płytki i mapa pinów.

W mapie pinów, w nawiasach okrągłych, podano numer pinu w złączu, a obok jego podstawową funkcjonalność.

GND oznacza masę układu (potencjał zerowy, względem, którego funkcjonują pozostałe napięcia i logika układu). *3V3* oznacza wyprowadzenie napięcia 3,3V, które można wykorzystać do zasilania układów peryferyjnych. *5V* oznacza napięcie 5V, dostarczane przez zasilacz, które można wykorzystać do zasilania układów o logice napięciowej 5V, podłączonych z użyciem konwertera stanów logicznych. Należy pamiętać, że wyprowadzenia 5V, a szczególnie 3,3V mają niewielką obciążalność prądową i należy rozważnie planować przyłączanie urządzeń peryferyjnych, aby nie uszkodzić płytki Raspberry Pi.

Porty cyfrowe oznaczono jako GPIO*nn*, gdzie *nn* oznacza numer logiczny pinu GPIO. Skrót GPIO pochodzi od angielskiego terminu *general purpose input/output*, czyli wejście/wyjście ogólnego przeznaczenia (zastosowania). Numery logiczne pinów

GPIO nie pokrywają się z numerami złącza płytki Raspberry Pi. Ponieważ w użyciu są oba sposoby identyfikacji numerycznej (numer logiczny GPIO i numer pinu złącza płytki Raspberry Pi), to jeśli przy programowaniu będziemy posługiwać się numerami, musimy dokładnie widzieć kiedy możemy, lub musimy, użyć numeru GPIO, a kiedy numeru pinu złącza płytki.

Zwróćmy też uwagę na fakt, że piny w złączu Raspberry Pi mogą pełnić różne funkcje, a nie tylko być pinami wejścia/wyjścia ogólnego przeznaczenia. Mogą być skonfigurowane, na przykład, jako porty komunikacyjne, ze wsparciem sprzętowym dla UART, SPI, I2C, itp. Dobry wgląd w mapowanie funkcjonalności pinów daje wspomniana już witryna Raspberry Pi Pinout (link).

1.3 Pomocniczy plik konfiguracji zestawu `config.py`

W celu ułatwienia programowania w czasie zajęć laboratoryjnych przygotowano plik konfiguracyjny, który zawiera podstawowe informacje o mapowaniu portów I/O do obsługi urządzeń korzystających z funkcjonalności portów cyfrowych. Dokonuje on też inicjalizacji protów GPIO urządzeń obsługiwanych przez te porty. Zawartość tego pliku przedstawiono jako kod 1.

Kod 1: Plik `config.py`.

```
#!/usr/bin/env python3
# pylint: disable=no-member

import RPi.GPIO as GPIO

# pin numbers in BCM
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

led1 = 13
led2 = 12
led3 = 19
led4 = 26
GPIO.setup(led1, GPIO.OUT)
GPIO.setup(led2, GPIO.OUT)
GPIO.setup(led3, GPIO.OUT)
GPIO.setup(led4, GPIO.OUT)

buttonRed = 5
buttonGreen = 6
encoderLeft = 17
encoderRight = 27
GPIO.setup(buttonRed, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(buttonGreen, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(encoderLeft, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(encoderRight, GPIO.IN, pull_up_down=GPIO.PUD_UP)

buzzerPin = 23
GPIO.setup(buzzerPin, GPIO.OUT)
GPIO.output(buzzerPin, 1)

ws2812pin = 8


def configInfo():
    print('This is only configuration file.\n')


if __name__ == "__main__":
    configInfo()
```

Plik ten jest częścią zestawu skryptów języka Python testujących zestaw Raspberry Pi. W skryptach testujących poszczególne peryferia można znaleźć przykłady jego wykorzystania. Przypomnijmy, że program testowy znajduje się w katalogu `/home/pi/tests`. Należy wykonać kopię pliku `config.py` do katalogu z własnym projektem i z niego korzystać, aby uniknąć pomyłek, czy zwykłych literówek, przy własnoręcznym wpisywaniu identyfikatorów numerycznych i inicjalizacji portów GPIO. Plik ten należy wykorzystywać w zadaniach.

1.4 Konfiguracja trybów pracy portów I/O i sterowanie portami

Porty cyfrowe Raspberry Pi, podobnie jak w Arduino, mogą pracować w dwóch podstawowych trybach: wejścia i wyjścia. Do obsługi portów I/O w języku Python wykorzystamy bibliotekę RPi.GPIO (link). Biblioteka ta wykorzystuje różne sposoby identyfikacji portów. Zgodnie ze schematem elektrycznym naszego zestawu (patrz dokument *Zestawy laboratoryjne z Raspberry Pi 4 - Wprowadzenie* z laboratorium 7), dioda święcąca LED 1 w module niebieskich diod sygnalizacyjnych podłączona jest do pinu 33 płytki Raspberry Pi. Pin ten oznaczany jest też jako GPIO13. Zestawienie dla pozostałych pinów, jest na kolejnej stronie za schematem, we wspomnianym dokumencie. Zatem, mamy numerację fizyczną pinu złącza (33) oraz logiczną (13) do sterowania diodą LED 1.

W programie możemy posłużyć się jedną z nich, ale musimy jawnie zadeklarować, którą będziemy posługiwać się. W dalszej części zajęć będziemy posługiwać się numeracją logiczną pinów, zatem taką deklarację wykonamy komendą:

```
GPIO.setmode(GPIO.BCM)
```

Operacja to została zapisana w pliku `config.py` przedstawionym jako kod 1 i będzie wykonana, w wraz z pozostałymi komendami tego pliku, podczas jego importu do własnego skryptu języka Python komendą:

```
from config import *
```

W pliku `config.py` są też zdefiniowane identyfikatory (zmienne), które mapują łatwe do skojarzenia nazwy z numerami logicznymi pinów. Przykładowo, dla LED 1 zdefiniowano zmienną:

```
led1 = 13
```

Mając te identyfikatory możemy teraz łatwo skonfigurować funkcje pinów i sterować ich stanem.

1.4.1 Porty w trybie wyjścia - sterowanie diodami LED i buzzerem.

Konfiguracja pinu diody LED 1 jako wyjścia będzie wykonywana komendą:

```
GPIO.setup(led1, GPIO.OUT)
```

Stan wyjścia wysoki, aby zaświecić diodę, ustawiać możemy komendą:

```
GPIO.output(led1, GPIO.HIGH)
```

natomiast, aby ją zgasić:

```
GPIO.output(led1, GPIO.LOW)
```

Prosty przykładowy program, wykorzystujący plik `config.py`, który 5 razy zaświeci LED1 będzie miał postać kodu 2. Przypomnijmy, że podstawowa konfiguracja pinów (portów) odbywa się w pliku `config.py`.

Kod 2: Plik `led1blink.py`.

```
#!/usr/bin/env python3

from config import *
import RPi.GPIO as GPIO
import time

def blink():
    GPIO.output(led1, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(led1, GPIO.LOW)
    time.sleep(1)

def blinkTest():
    for i in range (5):
        blink()
    GPIO.cleanup()


if __name__ == "__main__":
    print("\nProgram started")
    blinkTest()
    print("\nProgram finished")
```

W podobny sposób można sterować pozostałymi **diodami LED** oraz **buzzerem**. W przypadku sterowania buzzerem, należy zwrócić uwagę, że niski stan pinu sterującego powoduje włączenie dźwięku. Dlatego, w pliku `config.py`, stan pinu buzzera ustawiany jest na wysoki, aby nie włączyć dźwięku (komenda: `GPIO.output(buzzerPin, 1)`).

1.4.2 Porty w trybie wejścia - odczyt stanów przełączników przyciskanych i enkodera

Konfigurację portu omówimy na przykładzie czerwonego przełącznika przyciskanego. Przyłączony jest od portu GPIO 5, co zapisano w pliku `config.py` definicją identyfikatora (zmiennej):

```
buttonRed = 5
```

Konfiguracja pinu wejściowego realizowana jest komendą:

```
GPIO.setup(buttonRed, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

w której oprócz wskazania trybu wejściowego, włączono rezystor podciągający do dodatniego napięcia zasilającego. Powoduje to, że przełącznik przyciskany może być podłączony bez zewnętrznego rezystora podciągającego i tak jest podłączony w zestawie laboratoryjnym.

Bezpośredni odczyt stanu portu można zrealizować komendą:

```
GPIO.input(buttonRed)
```

Podobnie jak dla niebieskich diod świecących i buzzera, konfiguracja przełączników przyciskanych jest zawarta w pliku `config.py`.

Prosty program, który drukuje znak „*” i kontynuuje swoje działanie, aż do naciśnięcia **przycisku czerwonego** może zatem wyglądać następująco (kod 3):

Kod 3: Plik `redbutton.py`.

```
#!/usr/bin/env python3

from config import *
import RPi.GPIO as GPIO
import time

def redButtonTest():
    while GPIO.input(buttonRed) == GPIO.HIGH :
        print('*', end='', flush=True)
        time.sleep(0.1)


if __name__ == "__main__":
    print("\nProgram started")
    redButtonTest()
    print("\nProgram finished")
```

Warto przypomnieć, że przycisk podłączony jest do masy układu, z rezystorem podciągającym podłączonym do dodatniej szyny zasilania, podaje w stanie spoczynkowym (nie jest przyciśnięty) na pin wejścia stan wysoki, a po przyciśnięciu podaje stan niski.

Takie odpytywanie o stan portu powoduje jednak, że musimy co chwilę odpytywać o stan przełączników przyciskanych. Jeśli nie będziemy robić tego odpowiednio często, program może nie zarejestrować naciśnięcia przycisku. Innym rozwiązaniem jest reakcja na zdarzenia i stworzenie osobnych funkcji obsługi zdarzeń.

Popatrzmy na wersję programu wykorzystującą zdarzenia (kod 4).

Kod 4: Plik `redbuttonwithcallback.py`.

```
#!/usr/bin/env python3

from config import *
import RPi.GPIO as GPIO
import time

execute = True

def buttonPressedCallback(channel):
    global execute
    execute = False
    print("\nButton connected to GPIO " + str(channel) + " pressed.")

def redButtonTest():
    GPIO.add_event_detect(buttonRed, GPIO.FALLING, callback=buttonPressedCallback, bouncetime=200)

    while execute:
        print('*', end='', flush=True)
        time.sleep(0.1)


if __name__ == "__main__":
    print("\nProgram started\n")
    redButtonTest()
    print("\nProgram finished")
```

Program wykonuje to samo zadanie, to znaczy, drukuje gwiazdki aż do przyciśnięcia czerwonego przełącznika przyciskanego. Jednak tutaj funkcja `GPIO.add_event_detect()` pozwala dla określonego pinu zdefiniować, na które zbocze zmiany stanu ma reagować, która funkcja obsługi (*callback*) zostanie wywołana i jak długi ma być czas braku reakcji na kolejne zdarzenia. Ten ostatni parametr wykorzystywany jest do eliminacji wpływu drgania styków przełącznika przyciskanego (ang. *debouncing*).

Więcej informacji na ten temat na stronie RPi.GPIO - Inputs (link).

Podobnie można odczytywać stan **przełącznika zielonego i enkodera**. Enkoder w zestawie z Raspberry Pi jest taki sam jak w zestawie z płytką Arduino, więc zasada działania i odczyt stanów są takie same jak enkodera z zestawu z płytką Arduino.

1.4.3 Sygnał PWM

Sygnał PWM nie jest dostępny na każdym porcie. W zestawie laboratoryjnym jest on dostępny dla niebieskich diod święcących LED 1 i LED 2.

Aby utworzyć instancję wyjścia PWM dla LED 1 wydajemy komendę:

```
diode1 = GPIO.PWM(led1, 50)
```

Drugi parametr definiuje częstotliwość cyklu w jakim zmienia się współczynnik wypełnienia. W przykładzie jest to 50Hz. Aby wystartować generowanie sygnału PWM wydajemy komendę:

```
diode1.start(30)
```

przy czym parametr określa współczynnik wypełnienia przebiegu PWM i jest z zakresu od liczbowego 0,0 do 100,0.

Aby zmienić współczynnik wypełnienia możemy wydać komendę:

```
diode1.ChangeDutyCycle(70)
```

przy czym, ponownie, parametr określa współczynnik wypełnienia przebiegu PWM i jest z zakresu od liczbowego 0,0 do 100,0.

Generowanie przebiegu zatrzymujemy komendą:

```
diode1.stop()
```

Przykładowy program, który stopniowo rozjaśnia diodę LED 1, został przedstawiony jako kod 5.

Kod 5: Plik `pwm.py`.

```
#!/usr/bin/env python3

from config import *
import RPi.GPIO as GPIO
import time

def pwmTest():
    diode1 = GPIO.PWM(led1, 50)

    dutyCycle = 0
    diode1.start(dutyCycle)
    while dutyCycle < 100:
        time.sleep(0.2)
        dutyCycle += 10
        diode1.ChangeDutyCycle(dutyCycle)
    time.sleep(0.2)

    diode1.stop()
    GPIO.cleanup()


if __name__ == "__main__":
    print("\nProgram started")
    pwmTest()
    print("\nProgram finished")
```

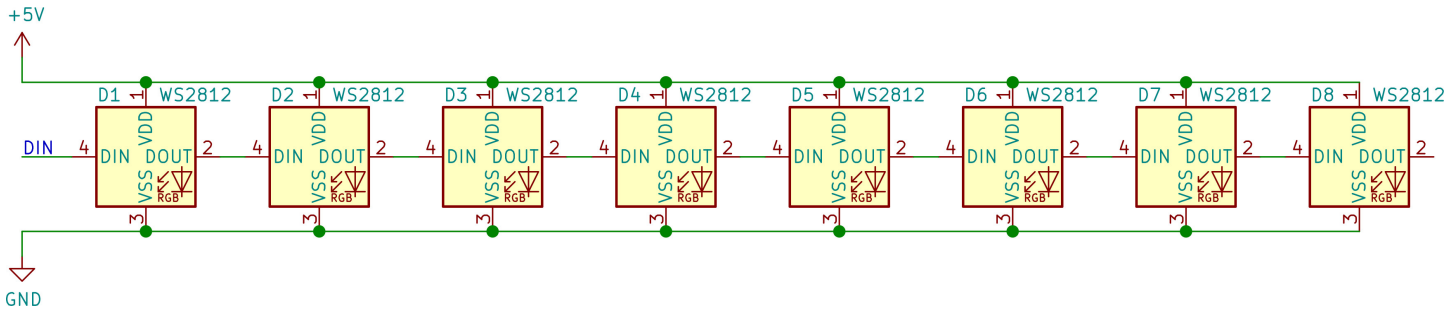
Więcej informacji na temat generowania sygnału PWM na stronie Using PWM in RPi.GPIO (link).

Zadanie 1: Regulacja jasności świecenia diody enkoderem

Napisz program, który pozwala regulować przy pomocy enkodera jasność świecenia diody LED 1 w module niebieskich diod świecących. Do obsługi enkodera wykorzystaj zdarzenia (*events*).
Jest to przykładowe zadanie, które może być zmienione przez Wykładowcę i połączone z innym.

2 Programowalne diody LED WS2812

Linijka diod LED WS2812 pozwala sterować kolorem i jasnością poszczególnych diod w sposób programowy. Diody połączone są zgodnie ze schematem 1.



Schemat 1: Schemat połączeń elektrycznych diod programowalnych WS2812.

Sygnał programujący diody podawany jest na pierwszą diodę i przekazywany dalej w łańcuchu połączonych diod. W ten sposób można programować dowolny kolor i jasność każdej z diod w łańcuchu diodowym osobno. Diody te są wykorzystywane do budowy ciekawych efektów świetlnych i ekranów LED. Więcej szczegółowych informacji na temat diod WS2812 można znaleźć w nocie katalogowej (link).

Do programowania wykorzystana zostanie biblioteka adafruit-circuitpython-neopixel (link). Przygotowanie systemu operacyjnego i instalację biblioteki opisano w materiale do laboratorium 7. Nieco dokładniejszy opis samej biblioteki można znaleźć w serii artykułów NeoPixels on Raspberry Pi (link).

Przykładową aplikację z wykorzystaniem diod WS2812 realizującą proste efekty świetlne prezentuje kod 6. Skrypt ten musi być uruchamiany z podniesionymi uprawnienia, czyli komendą `sudo`.

Kod 6: Plik `ws2812.py`.

```
#!/usr/bin/env python3
# This program must be executed with root privileges.
# Enter the command:
# sudo ./ws2812.py
import time
import os
import board
import neopixel
from config import *

def test():
    pixels = neopixel.NeoPixel(board.D18, 8, brightness=1.0/32, auto_write=False)

    pixels.fill((255, 0, 0))
    pixels.show()
    time.sleep(0.5)

    pixels.fill((0, 255, 0))
    pixels.show()
    time.sleep(0.5)

    pixels.fill((0, 0, 255))
    pixels.show()
    time.sleep(0.5)

    pixels[0] = (255, 0, 0)
    pixels[1] = (0, 255, 0)
    pixels[2] = (0, 0, 255)
    pixels[3] = (255, 255, 0)
    pixels[4] = (0, 255, 255)
    pixels[5] = (255, 0, 255)
    pixels[6] = (255, 255, 255)
    pixels[7] = (63, 63, 63)
    pixels.show()
    time.sleep(1)

    pixels.fill((0, 0, 0))
    pixels.show()

    GPIO.cleanup()


if __name__ == "__main__":
    print("\nProgram started")
    if os.getuid() == 0:
        test()
    else:
        print("\nWS2812 test ommited - root/sudo privileges demanded.")
    print("\nProgram finished")
```

3 Czujniki DS18B20 oraz BME280

Czujnik temperatury DS18B20 wykorzystany jest w zestawie z laboratoryjnym z Arduino i był przedmiotem ćwiczenia w ramach laboratorium 6. Jest to czujnik wykorzystujący magistralę komunikacyjną *1-Wire*. Szczegółowe informacje na temat czujnika zawiera jego nota katalogowa (link).

Do obsługi czujnika DS18B20 wykorzystana zostanie biblioteka W1ThermSensor (link). Przygotowanie systemu operacyjnego i instalację biblioteki opisano w materiale do laboratorium 7. Sposób jej wykorzystania można znaleźć na stronie projektu W1ThermSensor (link)

Czujnik BME280 przez producenta opisany jest następująco na stronie produktu (link):

„BME280 to czujnik wilgotności opracowany specjalnie do zastosowań mobilnych i noszonych na ubraniach, gdzie rozmiar i niski pobór mocy są kluczowymi parametrami projektowymi. Urządzenie łączy w sobie czujniki o wysokiej liniowości i dokładności oraz jest doskonale dostosowane do aplikacji, gdzie wymagany jest niski pobór prądu, długoterminowa stabilność oraz wysoka odporność na zakłócenia elektromagnetyczne. Czujnik wilgotności oferuje niezwykle szybki czas reakcji, dzięki czemu spełnia wymagania wydajnościowe dla nowych aplikacji, takich jak świadomość kontekstowa, oraz posiada wysoką dokładność w szerokim zakresie temperatur."

Szczegółowy opis BME280, jego parametrów i wykorzystania można znaleźć w bardzo rozbudowanej nocie katalogowej układu (link).

W zestawie z laboratoryjnym z Raspberry Pi, czujnik BME280 wykorzystuje do komunikacji magistralę I2C. Do obsługi czujnika podczas zajęć wykorzystana zostanie biblioteka adafruit-circuitpython-bme280. Podobnie jak w przypadku poprzednich bibliotek, przygotowanie systemu operacyjnego i instalację biblioteki opisano w materiale do laboratorium 7.

Przykładową aplikację odczytującą parametry środowiskowe mierzone przez układy DS18B20 i BME280 prezentuje kod 7.

Kod 7: Plik `thermometers.py`.

```
#!/usr/bin/env python3

from config import *
import w1thermsensor
import board
import busio
import adafruit_bme280.advanced as adafruit_bme280

def ds18b20():
    sensor = w1thermsensor.W1ThermSensor()
    temp = sensor.get_temperature()
    print(f'\nDS18B200 Temp : {temp} '+chr(176)+'C')

def bme280():
    i2c = busio.I2C(board.SCL, board.SDA)
    bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x76)

    bme280.sea_level_pressure = 1013.25
    bme280.standby_period = adafruit_bme280.STANDBY_TC_500
    bme280.iir_filter = adafruit_bme280.IIR_FILTER_X16
    bme280.overscan_pressure = adafruit_bme280.OVERSCAN_X16
    bme280.overscan_humidity = adafruit_bme280.OVERSCAN_X1
    bme280.overscan_temperature = adafruit_bme280.OVERSCAN_X2

    print('\nBME280:')
    print(f'Temperature: {bme280.temperature:0.1f} '+chr(176)+'C')
    print(f'Humidity: {bme280.humidity:0.1f} %')
    print(f'Pressure: {bme280.pressure:0.1f} hPa')
    print(f'Altitude: {bme280.altitude:0.2f} meters')

def test():
    print('\nThermometers test.')
    ds18b20()
    bme280()
    GPIO.cleanup()


if __name__ == "__main__":
    print("\nProgram started")
    test()
    print("\nProgram finished")
```

Zadanie 2: Odczyt parametrów środowiskowych z czujników DS18B20 oraz BME280 i ich wizualizacja poprzez diody WS2812

Napisz program, który wykorzystuje linijkę diod WS2812 do wizualizacji parametrów środowiskowych, odczytanych z czujników DS18B20 oraz BME280, według schematu podanego przez Wykładowcę. Program może wykorzystywać interakcje przez konsolę tekstową i być sterowany poprzez przełączniki przyciskane i enkoder.
Jest to przykładowe zadanie, które może być zmienione przez Wykładowcę i połączone z innym.