# Podstawy Systemów Mikroprocesorowych Laboratorium

Laboratorium 5: $I^2C$ — magistrala wielu urządzeń

Autorzy: Dariusz Tefelski, Angelika Tefelska

# Spis treści

**1 Wprowadzenie** **1**

**2 Wyposażenie stanowiska** **1**

**3 Zadanie 1: Uruchomienie magistrali i skaner $I^2C$** **2**

**4 Zadanie 2: Co dzieje się na magistrali?** **4**

**5 Zadanie 3: Pamięć EEPROM** **6**
5.1 Pierwszy zapis i odczyt z pamięci EEPROM . . . . . 6
5.2 Test nie ulotności pamięci EEPROM . . . . . 7

**6 Zadanie 4: Zegar czasu rzeczywistego RTC** **7**
6.1 Odczyt czasu z RTC . . . . . 8
6.2 Ustawienie daty i godziny . . . . . 9

**7 Zadanie 5: Kilka urządzeń na jednej magistrali** **10**

**8 Zadanie końcowe: Prosty monitor środowiskowy** **13**

**9 Podsumowanie** **14**

# 1 Wprowadzenie

**Cel ćwiczenia**

Celem ćwiczenia jest poznanie magistrali I²C i wykorzystanie jej do komunikacji z kilkoma urządzeniami peryferyjnymi podłączonymi równolegle do tych samych linii. W czasie laboratorium wykorzystasz pamięć EEPROM, zegar czasu rzeczywistego RTC oraz moduł środowiskowy ENS160/BME280.

Na Laboratorium 4 UART łączył dwa urządzenia głównie w układzie punkt–punkt. I²C działa inaczej: wiele urządzeń może współdzielić te same dwie linie sygnałowe. O tym, z którym urządzeniem prowadzona jest komunikacja, decyduje **adres przesyłany na początku transakcji**.

**Dwie linie magistrali I²C**

- **SDA** (*Serial Data*) — dwukierunkowa linia danych,
- **SCL** (*Serial Clock*) — linia zegarowa generowana przez urządzenie sterujące.

Na tych zajęciach Raspberry Pi Pico 2 W jest urządzeniem **sterującym** (*controller*), a RTC, EEPROM i czujniki są urządzeniami **peryferyjnymi** (*target/peripheral*). W starszej dokumentacji można spotkać nazwy *master* i *slave*.

**Dlaczego linie SDA i SCL wymagają rezystorów podciągających?**

Urządzenia I²C nie wymuszają aktywnie stanu wysokiego na liniach SDA i SCL. Mogą natomiast zwierać linię do masy, wymuszając stan niski. Stan wysoki zapewniają rezystory podciągające do napięcia zasilania.
Dzięki temu wiele urządzeń może być bezpiecznie połączonych równolegle do tej samej magistrali. Na używanej płytce oraz na module SparkFun znajdują się rezystory podciągające, dlatego w tym laboratorium nie dodajemy osobnych rezystorów.

# 2 Wyposażenie stanowiska

Do wykonania ćwiczenia potrzebne są:

- płytka dydaktyczna z Raspberry Pi Pico 2 W,
- przewód USB i przewody połączeniowe,
- zegar czasu rzeczywistego PCF8563 znajdujący się na płytce,
- pamięć EEPROM M24C02/AT24C02 znajdująca się na płytce,
- moduł SparkFun Environmental Combo Breakout ENS160/BME280 (SEN-22858),
- przewód Qwiic z końcówkami do podłączenia do płytki dydaktycznej,

- jedna z diod LED na płytce,
- Analog Discovery II z programem WaveForms.

W tym laboratorium wykorzystamy sprzętowy interfejs $I^2C0$:

| Sygnał | Raspberry Pi Pico 2 W |
|---|---|
| SDA | `GP16` |
| SCL | `GP17` |

**Co należy zapamiętać?**

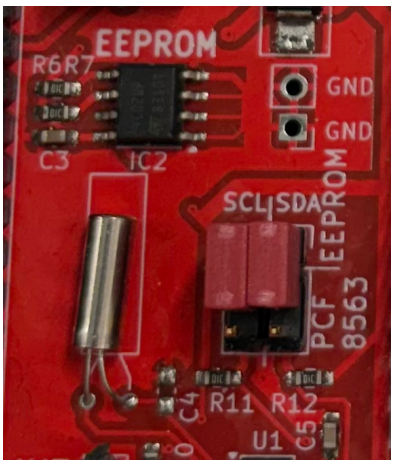
Wszystkie urządzenia podłączone do magistrali muszą mieć wspólną masę. Moduł SparkFun ENS160/BME280 zasilaj napięciem **3,3 V**. Nie podawaj napięcia 5 V na linie SDA lub SCL Raspberry Pi Pico 2 W.

# 3 Zadanie 1: Uruchomienie magistrali i skaner $I^2C$

Na początek podłącz do Raspberry Pi Pico zegar RTC oraz pamięć EEPROM:

- SDA zegara RTC i SDA pamięci EEPROM do `GP16` (piny pod zworkami z rys. 1),
- SCL zegara RTC i SCL pamięci EEPROM do `GP17`.

Oba urządzenia korzystają z tych samych linii. Nie oznacza to jednak konfliktu, ponieważ mają różne adresy.




Rysunek 1: Wyprowadzenia magistrali przy EEPROM i RTC na płytce dydaktycznej. W orientacji pokazanej na zdjęciu SCL znajduje się po lewej, a SDA po prawej. Kieruj się opisami nadrukowanymi na płytce.

### Zadanie 1.1 — Skanowanie magistrali

Uruchom program:

```
from machine import Pin, I2C

i2c = I2C(0, sda=Pin(16), scl=Pin(17), freq=100_000)

devices = i2c.scan()

print("Znaleziono:", len(devices))
for address in devices:
    print(hex(address))
```

Na płytce dydaktycznej powinny zostać wykryte dwa urządzenia:

- pamięć EEPROM — adres `0x50`,
- RTC PCF8563 — adres `0x51`.

Odłącz kolejno linię SDA jednego z urządzeń i ponownie wykonaj skanowanie. Sprawdź, który adres znika z listy.

### Dlaczego zapisujemy `100_000` zamiast `100000`?

W Pythonie znak podkreślenia `_` może być używany wewnątrz liczby jako **separator poprawiający czytelność**. Nie zmienia on wartości liczby.
Na przykład:

`100_000 = 100000`, `400_000 = 400000`.

W tym przypadku zapis `freq=100_000` oznacza więc częstotliwość 100 000 Hz, czyli 100 kHz.

### Adres 7-bitowy i adresy spotykane w dokumentacji

W MicroPythonie podajemy **7-bitowy adres urządzenia**. Dlatego RTC PCF8563 używamy pod adresem `0x51`.
Niektóre dokumentacje pokazują osobne wartości dla zapisu i odczytu, np. `0xA2` i `0xA3`. Są to wartości zawierające już dodatkowy bit R/W. Nie należy ich bezpośrednio wpisywać jako adresu do funkcji MicroPythona.

**Podstawowe funkcje I²C w MicroPythonie**

- `i2c.scan()` — zwraca listę 7-bitowych adresów urządzeń odpowiadających na magistrali,
- `i2c.readfrom(addr, n)` — odczytuje `n` bajtów z urządzenia,
- `i2c.writeto(addr, data)` — wysyła bajty do urządzenia,
- `i2c.readfrom_mem(addr, reg, n)` — odczytuje dane od wskazanego rejestru lub komórki pamięci,
- `i2c.writeto_mem(addr, reg, data)` — zapisuje dane od wskazanego rejestru lub komórki pamięci.

# 4 Zadanie 2: Co dzieje się na magistrali?

W I²C każda transakcja jest synchronizowana zegarem SCL. Urządzenie sterujące rozpoczyna transmisję warunkiem START, wysyła adres urządzenia i informację o kierunku transmisji, a odbiornik potwierdza odebrane bajty bitem ACK.

**Pomiar 2.1 — Dekodowanie I²C w Analog Discovery II**

Podłącz analizator do działającej magistrali, pozostawiając EEPROM i RTC połączone z Pico:

| Analog Discovery II | Magistrala / Pico |
|---|---|
| DIO 0 | SCL / `GP17` |
| DIO 1 | SDA / `GP16` |
| GND | GND płytki |

W programie WaveForms:

1. uruchom narzędzie **Logic** i dodaj dekoder **I2C**,
2. ustaw **SCL = DIO 0**, **SDA = DIO 1**,
3. wybierz **Mode: Record** oraz **Protocol: I2C**.

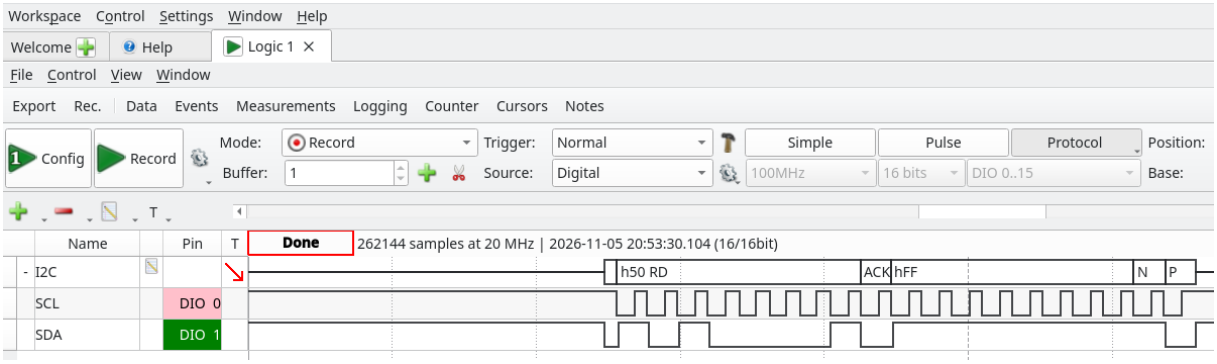
**Zamiast skanowania wszystkich adresów odczytuj jeden bajt z urządzenia `0x50` co sekundę:**

```
from machine import Pin, I2C
from time import sleep

i2c = I2C(0, sda=Pin(16), scl=Pin(17), freq=100_000)

while True:
    try:
        data = i2c.readfrom(0x50, 1)
        print("Odebrano:", hex(data[0]))
    except OSError:
        print("Blad odczytu lub brak odpowiedzi z 0x50")
    sleep(1)
```

Argument `1` oznacza **liczbę odczytywanych bajtów**, a nie wartość wysyłanych danych. Wywołanie `sleep(1)` wprowadza jednosekundową przerwę. Operacja $I^2C$ znajduje się wewnątrz pętli, więc transmisja powtarza się podczas pomiaru.




Rysunek 2: Odczyt jednego bajtu z EEPROM pod adresem `0x50`. Górny przebieg to SCL, dolny to SDA. W tym przykładzie odebrano `0xFF`.

**Jak odczytać zarejestrowaną ramkę?**

1. **START**: SDA opada, gdy SCL jest w stanie wysokim.
2. **h50 RD**: Pico wysyła 7-bitowy adres `1010000` i bit odczytu `1`; na magistrali daje to `10100001` (`0xA1`). W kodzie nadal podajemy adres `0x50`.
3. **ACK**: EEPROM ustawia SDA na 0 podczas dziewiątego impulsu zegara, potwierdzając adres.
4. **hFF**: EEPROM wysyła bajt `11111111`, dlatego SDA pozostaje wysoko przez osiem taktów zegara.
5. **N**: Pico kończy odczyt bitem NACK (SDA = 1). Jest to prawidłowe zakończenie odbioru ostatniego bajtu, a nie błąd.
6. **P**: STOP — SDA narasta, gdy SCL jest w stanie wysokim.

Bity odczytuj z SDA podczas wysokiego poziomu SCL. Ramka zawiera 18 taktów: 8 bitów adresu z R/W, ACK, 8 bitów danych i NACK.

**Uwaga:** `readfrom(0x50, 1)` odczytuje EEPROM spod bieżącego wewnętrznego wskaźnika adresu; kolejne odczyty mogą zwracać różne bajty. Wartość `0xFF` z rysunku jest przykładem, a nie wymaganym wynikiem. Odczyt z wybranej komórki poznasz w zadaniu 3.

# 5 Zadanie 3: Pamięć EEPROM

Pamięć EEPROM jest pamięcią **nieulotną**: zachowuje zapisane dane po zaniku zasilania. Zastosowana na płytce pamięć M24C02 ma pojemność 2 kbit, czyli:

$$2\,\text{kbit} = 2048\,\text{bit} = 256\,\text{B}. \tag{1}$$

Każda komórka może przechować jeden bajt.

## 5.1 Pierwszy zapis i odczyt z pamięci EEPROM

**Zadanie 3.1 — Pierwszy zapis i odczyt EEPROM**

Uruchom:

```
from utime import sleep_ms

EEPROM_ADDR = 0x50
MEM_ADDR = 0x20

data = b"PSM"

i2c.writeto_mem(EEPROM_ADDR, MEM_ADDR, data)
sleep_ms(10)

read_data = i2c.readfrom_mem(EEPROM_ADDR, MEM_ADDR, len(
    data))

print(read_data)
print(read_data.decode())
```

**Dwa różne adresy w jednej operacji**

W wywołaniu:

`i2c.writeto_mem(0x50, 0x20, data)`

`0x50` jest adresem **układu EEPROM na magistrali I²C**, natomiast `0x20` wskazuje miejsce **wewnątrz pamięci EEPROM**.

**Co należy zapamiętać?**

EEPROM ma ograniczoną liczbę cykli zapisu. Nie zapisuj tej samej komórki bez potrzeby w bardzo szybkiej pętli. Odczyt nie powoduje takiego zużycia pamięci.

## 5.2 Test nie ulotności pamięci EEPROM

**Zadanie 3.2 — Dane zachowane po restarcie**



Zapisz w EEPROM dowolny tekst o długości nie większej niż 16 bajtów, rozpoczynając od adresu `0x40`.
Następnie:

1. zatrzymaj program,
2. wykonaj restart Raspberry Pi Pico,
3. uruchom osobny program, który **tylko odczytuje** zapisane dane,
4. sprawdź, czy zawartość pamięci pozostała niezmieniona.

Nie zapisuj danych ponownie przed wykonaniem odczytu.

# 6 Zadanie 4: Zegar czasu rzeczywistego RTC

PCF8563 jest zewnętrznym zegarem czasu rzeczywistego. Niezależny oscylator kwarcowy 32,768 kHz umożliwia zliczanie sekund, minut, godzin oraz daty.

Układ RTC znajduje się pod adresem:

`0x51`

**Zadanie 4.0 — Odszukanie rejestrów w dokumentacji RTC**



Otwórz dokumentację układu PCF8563 dostępną na stronie producenta:
https://www.nxp.com/docs/en/data-sheet/PCF8563.pdf
Odszukaj tabelę opisującą mapę rejestrów (*register overview*) i uzupełnij poniższą tabelę adresami rejestrów przechowujących poszczególne elementy czasu i daty:

| **Wielkość** | **Adres rejestru** |
|---|---|
| sekundy | |
| minuty | |
| godziny | |
| dzień miesiąca | |
| dzień tygodnia | |
| miesiąc | |
| rok | |

Zwróć uwagę, że nie wszystkie bity w każdym rejestrze służą do zapisu wartości czasu. Odszukaj również w dokumentacji, które bity należy pominąć podczas odczytu sekund, minut, godzin, dnia miesiąca i miesiąca.
Po uzupełnieniu tabeli sprawdź, czy rejestry czasu tworzą ciągły obszar pamięci. Informacja ta będzie potrzebna w następnym zadaniu.

**BCD — liczby dziesiętne zapisane cyframi**

Większość pól czasu w PCF8563 jest zapisana w formacie BCD (*Binary-Coded Decimal*). Każda cyfra dziesiętna jest kodowana osobno w czterech bitach. Na przykład liczba 25 jest zapisana jako:

`0x25`

a nie jako binarna wartość dziesiętna 25, czyli `0x19`.

## 6.1 Odczyt czasu z RTC

**Zadanie 4.1 — Odczyt czasu z RTC**

Użyj funkcji:

```
def bcd_to_dec(value):
    return ((value >> 4) * 10) + (value & 0x0F)
```

Na podstawie adresów znalezionych w dokumentacji odczytaj jednym poleceniem siedem kolejnych bajtów rozpoczynając od rejestru sekund:

```
RTC_ADDR = 0x51

raw = i2c.readfrom_mem(
    RTC_ADDR,
    tu_wpisz_rejestr_sekund,
    7
)

seconds = bcd_to_dec(raw[0] & 0x7F)
minutes = bcd_to_dec(raw[1] & 0x7F)
hours   = bcd_to_dec(raw[2] & 0x3F)
day     = bcd_to_dec(raw[3] & 0x3F)
weekday = raw[4] & 0x07
month   = bcd_to_dec(raw[5] & 0x1F)
year    = 2000 + bcd_to_dec(raw[6])

print(year, month, day, hours, minutes, seconds)
```

Uruchom program kilka razy w odstępach kilku sekund i sprawdź, czy czas jest zwiększany.

## 6.2 Ustawienie daty i godziny

**Zadanie 4.2 — Ustawienie daty i godziny**



Do konwersji zwykłej liczby dziesiętnej na zapis BCD wykorzystaj funkcję:

```
def dec_to_bcd(value):
    return ((value // 10) << 4) | (value % 10)
```

Argument `value` powinien być liczbą całkowitą typu `int`. Funkcja również zwraca liczbę typu `int`, której bity zawierają wartość zakodowaną w BCD.
Przygotuj zmienne typu `int` opisujące aktualną datę i godzinę, np.:

```
year = 2026
month = 10
day = 26
weekday = 0
hours = 12
minutes = 30
seconds = 0
```

W tym ćwiczeniu przyjmij umownie, że poniedziałek ma numer 0, wtorek 1, ..., a niedziela 6.
Następnie utwórz **siedmiobajtowy obiekt typu** `bytes`. Konstruktor `bytes([...])` przyjmuje listę liczb całkowitych z zakresu 0–255 i zapisuje każdą z nich jako jeden bajt:

```
data = bytes([
    dec_to_bcd(seconds),
    dec_to_bcd(minutes),
    dec_to_bcd(hours),
    dec_to_bcd(day),
    weekday,
    dec_to_bcd(month),
    dec_to_bcd(year - 2000)
])
```

Zapisz przygotowany obiekt do kolejnych rejestrów RTC, rozpoczynając od rejestru sekund:

```
i2c.writeto_mem(
    RTC_ADDR,
    0x02,
    data
)
```

Ustaw aktualną datę i godzinę, a następnie ponownie użyj programu z zadania 4.1 i sprawdź poprawność odczytu.
Dla uproszczenia przyjmij lata z zakresu 2000–2099.

# 7 Zadanie 5: Kilka urządzeń na jednej magistrali

Do tej samej magistrali dołączymy teraz moduł SparkFun Environmental Combo Breakout ENS160/BME280, nie odłączaj pozostałych elementów. Moduł zawiera **dwa niezależne układy I²C**:

- ENS160 — czujnik jakości powietrza,
- BME280 — czujnik temperatury, wilgotności i ciśnienia.

W konfiguracji fabrycznej modułu typowe adresy wynoszą:

| Układ | Adres I²C |
|---|---|
| ENS160 | `0x53` |
| BME280 | `0x77` |

**Co należy zapamiętać?**

Moduł SparkFun SEN-22858 zasilaj napięciem 3,3 V. Na płytce znajdują się rezystory podciągające linie I²C.

**Zadanie 5.1 — Cztery urządzenia na dwóch przewodach**

Podłącz moduł ENS160/BME280 do magistrali I²C zgodnie z poniższą tabelką. EEPROM i RTC pozostają podłączone.

| Moduł ENS160/BME280 | Połączenie |
|---|---|
| GND | GND płytki |
| VCC | 3,3 V |
| SDA | `GP16` |
| SCL | `GP17` |

Uruchom ponownie skaner z zadania 1.1.

Przy domyślnej konfiguracji powinny pojawić się cztery adresy:

`0x50`, `0x51`, `0x53`, `0x77`

Przypisz każdy adres do odpowiedniego urządzenia.

## Zadanie 5.2 — Identyfikacja układów przez rejestry

BME280 przechowuje identyfikator układu w rejestrze `0xD0`. Odczytaj go:

```
BME_ADDR = 0x77

bme_id = i2c.readfrom_mem(BME_ADDR, 0xD0, 1)[0]

print(hex(bme_id))
```

Dla BME280 oczekiwana wartość to `0x60`.

### Dlaczego na końcu polecenia znajduje się `[0]`?

Funkcja:

`i2c.readfrom_mem(BME_ADDR, 0xD0, 1)`

zwraca obiekt typu `bytes`. Ponieważ żądamy odczytu jednego bajtu, wynik może wyglądać np. tak:

`b'\x60'`

Zapis `[0]` wybiera **pierwszy element** tego obiektu. Otrzymujemy wtedy zwykłą liczbę całkowitą:

$$0x60 = 96.$$

Można więc zapisać tę samą operację w dwóch krokach:

```
raw_id = i2c.readfrom_mem(BME_ADDR, 0xD0, 1)
bme_id = raw_id[0]
```

ENS160 przechowuje dwubajtowy identyfikator od rejestru `0x00`. Odczytaj dwa bajty i połącz je:

```
ENS_ADDR = 0x53

part = i2c.readfrom_mem(ENS_ADDR,0x00,2)

ens_id = part[0] | (part[1] << 8)

print(hex(ens_id))
```

Oczekiwana wartość identyfikatora ENS160 to `0x0160`.

### Little endian

ENS160 zwraca identyfikator w kolejności *little endian*: najpierw młodszy bajt, a następnie starszy. Dlatego dla dwóch bajtów `0x60` i `0x01` otrzymujemy:

$$0x01 \cdot 256 + 0x60 = 0x0160. \tag{2}$$

### Zadanie 5.3 — Odczyt danych środowiskowych

Do pełnego przeliczenia danych BME280 potrzebne są współczynniki kalibracyjne zapisane indywidualnie w każdym egzemplarzu czujnika, dlatego w tym zadaniu wykorzystamy gotowe sterowniki MicroPython dostarczone przez prowadzącego.

Skopiuj do pamięci Raspberry Pi Pico pliki sterowników oraz program przykładowy `env_read.py`. Uruchom program i sprawdź, czy odczytywane są:

- temperatura,
- wilgotność względna,
- ciśnienie atmosferyczne,
- AQI,
- TVOC,
- $eCO_2$.

Następnie zmodyfikuj program tak, aby wyniki były wypisywane w jednej linii co około 2 s, w formacie zbliżonym do CSV.

Przykład:

```
23.4,45.2,1008.6,1,45,610
```

### ENS160 potrzebuje czasu po włączeniu

ENS160 wykorzystuje elementy gazoczułe wymagające rozgrzania i stabilizacji. Bezpośrednio po włączeniu nie traktuj wskazań AQI, TVOC i $eCO_2$ jako ustabilizowanych wartości pomiarowych. Sterownik powinien umożliwiać sprawdzenie statusu czujnika.

# 8 Zadanie końcowe: Prosty monitor środowiskowy

### Zadanie 6.0 — Jakie warunki w pomieszczeniu są prawidłowe?



Przed napisaniem programu odszukaj w dokumentacji i wiarygodnych źródłach zalecane wartości parametrów opisujących warunki w pomieszczeniu.
W szczególności znajdź:

- zalecany zakres temperatury w pomieszczeniu przeznaczonym do pracy lub nauki,
- zalecany zakres wilgotności względnej,
- interpretację wartości AQI zwracanej przez ENS160,
- interpretację wartości $eCO_2$ zwracanej przez ENS160.

Dla AQI i $eCO_2$ skorzystaj z dokumentacji producenta ENS160 i odszukaj tabele opisujące klasy jakości powietrza oraz zalecenia dotyczące wentylacji. Dla temperatury i wilgotności wykorzystaj wiarygodne źródło dotyczące warunków w pomieszczeniach, np. materiały instytucji publicznej: https://www.gov.pl/web/psse-wieliczka/ergonomia-pracy-zdalnej?utm_source=chatgpt.com.
Zapisz wybrane zakresy/progi oraz źródła informacji. Umieść je również jako komentarz na początku programu, np.:

```
# Kryteria oceny:
# temperatura: ...
# wilgotnosc: ...
# AQI: ...
# eCO2: ...
# Zrodla: ...
```

Jeżeli różne wiarygodne źródła podają nieco inne zakresy, wybierz jeden z nich i konsekwentnie zastosuj go w programie.

### Zadanie 6.1 — RTC + czujniki + ocena warunków

Napisz program, który łączy elementy poznane podczas laboratorium.
Program powinien:

1. odczytywać aktualną godzinę z RTC,
2. pobierać dane z modułu ENS160/BME280,
3. co około 1 min wyświetlać w jednym wierszu czas i wszystkie dostępne wyniki pomiarów,
4. porównywać temperaturę, wilgotność, AQI oraz $eCO_2$ z progami znalezionymi w zadaniu 6.0,
5. wyświetlać czytelną ocenę warunków, np. `WARUNKI OK`, `ZA CIEPLO`, `ZA SUCHO` lub `WYWIETRZ POMIESZCZENIE`,
6. zapalać wybraną diodę LED, jeżeli co najmniej jeden z ocenianych parametrów znajduje się poza przyjętym zakresem,

7. nie oceniać jakości powietrza na podstawie ENS160, jeżeli czujnik zgłasza, że dane nie są jeszcze poprawne.

Ciśnienie atmosferyczne wyświetlaj razem z pozostałymi wynikami, ale nie musisz wykorzystywać go do oceny komfortu w pomieszczeniu.

Przykładowa linia wyjściowa może mieć postać:

```
12:34:08,T=23.4,H=45.2,P=1008.6,AQI=1,eCO2=610, WARUNKI OK
```

Pamiętaj, że diody LED na płytce dydaktycznej są aktywne stanem niskim.

# 9 Podsumowanie

**Po tych zajęciach powinieneś umieć**



- skonfigurować sprzętową magistralę $I^2C$ w MicroPythonie,
- wyjaśnić rolę SDA, SCL oraz rezystorów podciągających,
- odnaleźć urządzenia za pomocą skanera $I^2C$,
- rozróżnić adres urządzenia od adresu rejestru lub komórki pamięci,
- obserwować adres, ACK oraz dane w analizatorze logicznym,
- zapisywać i odczytywać dane z EEPROM,
- przeliczać wartości BCD używane przez RTC,
- odczytać rejestr identyfikacyjny urządzenia,
- korzystać z kilku urządzeń o różnych adresach na wspólnej magistrali.

**Co należy zapamiętać?**



Przed opuszczeniem stanowiska zatrzymaj program i usuń z pamięci Raspberry Pi Pico pliki utworzone podczas tego laboratorium. Interpreter MicroPython powinien pozostać zainstalowany. Odłącz zewnętrzny moduł ENS160/BME280 i uporządkuj stanowisko.