# Podstawy Systemów Mikroprocesorowych Laboratorium

Laboratorium 7: 1-Wire i microSD — rejestrator danych

Autorzy: Dariusz Tefelski, Angelika Tefelska

# Spis treści

**1 Wprowadzenie** **1**

**2 Wyposażenie stanowiska** **1**

**3 Zadanie 1: Magistrala 1-Wire i identyfikacja czujnika** **2**

**4 Zadanie 2: Pomiar temperatury** **4**

**5 Zadanie 3: Co dzieje się na linii 1-Wire?** **5**

**6 Zadanie 4: Karta microSD i system plików** **5**

**7 Zadanie 5: Zapis i odczyt plików** **8**

**8 Zadanie 6: RTC + DS18B20 + microSD** **10**

**9 Zadanie końcowe: Rejestrator temperatury** **10**

**10 Podsumowanie** **12**

# 1 Wprowadzenie

**Cel ćwiczenia**

Celem ćwiczenia jest poznanie magistrali 1-Wire oraz budowa prostego rejestratora danych. W czasie laboratorium wykorzystasz cyfrowy czujnik temperatury DS18B20, zegar czasu rzeczywistego RTC oraz kartę microSD jako pamięć masową.

Na poprzednich laboratoriach poznaliśmy interfejsy UART, $I^2C$ i SPI. Magistrala 1-Wire ma inną cechę charakterystyczną: do komunikacji z urządzeniami wykorzystuje pojedynczą linię danych, oprócz masy i zasilania.

Czujnik DS18B20 jest typowym urządzeniem 1-Wire. Każdy egzemplarz ma unikatowy 64-bitowy identyfikator ROM, dzięki czemu do jednej linii danych można podłączyć wiele czujników i wybierać je indywidualnie.

Druga część laboratorium dotyczy zapisywania danych na karcie microSD. Karta będzie pracowała w trybie SPI, ale tym razem nie analizujemy szczegółowo protokołu karty. Interesuje nas przede wszystkim **pamięć masowa, system plików i bezpieczny zapis danych**.

# 2 Wyposażenie stanowiska

Do wykonania ćwiczenia potrzebne są:

- płytka dydaktyczna z Raspberry Pi Pico 2 W,
- czujnik temperatury DS18B20 znajdujący się na płytce,
- zegar czasu rzeczywistego PCF8563,
- gniazdo microSD na płytce,
- karta microSD sformatowana w systemie FAT,
- przewody połączeniowe,
- Analog Discovery II z programem WaveForms.

W laboratorium wykorzystamy następujące połączenia:

| Urządzenie / sygnał | Raspberry Pi Pico 2 W |
|---|---|
| DS18B20 DQ | `GP14` |
| SD CLK | `GP10` |
| SD CMD / MOSI | `GP11` |
| SD DAT0 / MISO | `GP12` |
| SD DAT3 / CS | `GP13` |
| SD DETECT | `GP15` |
| RTC SDA | `GP16` |
| RTC SCL | `GP17` |

**Rezystor podciągający DS18B20**

Linia DQ magistrali 1-Wire wymaga rezystora podciągającego do zasilania. W typowych przykładach spotkasz wartość około 4,7 kΩ.
Na płytce dydaktycznej rezystor podciągający jest już zamontowany i ma wartość 10 kΩ, dlatego **nie dodawaj dodatkowego rezystora**.

# 3 Zadanie 1: Magistrala 1-Wire i identyfikacja czujnika

1-Wire wykorzystuje jedną dwukierunkową linię danych. Urządzenia nie mają klasycznych adresów odpowiadających rodzajowi układu. Zamiast tego każde urządzenie posiada fabrycznie zapisany unikatowy kod ROM.

Dla DS18B20 kod ROM ma długość 64 bitów:

- 8 bitów — kod rodziny urządzenia,
- 48 bitów — unikatowy numer seryjny,
- 8 bitów — CRC.

**Zadanie 1.1 — Skanowanie magistrali 1-Wire**

Połącz linię DQ czujnika DS18B20 z `GP14` i uruchom:

```
from machine import Pin
import onewire
import ds18x20

ow = onewire.OneWire(Pin(14))
ds = ds18x20.DS18X20(ow)

roms = ds.scan()

```

```
print("Liczba urzadzen:", len(roms))

for rom in roms:
    print(rom)
```

Powinien zostać znaleziony co najmniej jeden czujnik.
Sprawdź typ i długość kodu ROM:

```
print(type(roms[0]))
print(len(roms[0]))
```

Następnie wyświetl kod ROM w czytelnej postaci szesnastkowej:

```
for rom in roms:
    rom_hex = "".join(
        "{:02X}".format(byte)
        for byte in rom
    )

    print(rom_hex)
```

### Dlaczego każdy czujnik ma własny kod ROM?

W magistrali 1-Wire wiele urządzeń może być podłączonych do tej samej linii DQ. Unikatowy kod ROM umożliwia wybranie konkretnego urządzenia bez stosowania osobnych linii CS ani konfigurowanych adresów.
Dla rodziny DS18B20 pierwszy bajt kodu ROM ma wartość `0x28`.

### Co to jest CRC?

CRC (*Cyclic Redundancy Check* — cykliczna kontrola nadmiarowa) jest mechanizmem służącym do **wykrywania błędów w danych**. Nie szyfruje danych i nie poprawia błędu — pozwala jedynie sprawdzić, czy odebrana informacja najprawdopodobniej nie została zmieniona podczas transmisji lub zapisu.
Nadajnik oblicza z przesyłanych danych krótką wartość kontrolną według określonego algorytmu i dołącza ją do danych. Odbiornik wykonuje to samo obliczenie i porównuje otrzymany wynik z przesłaną wartością CRC.
W kodzie ROM DS18B20:

- pierwsze 7 bajtów zawiera kod rodziny i numer seryjny,
- ósmy bajt zawiera **CRC-8** obliczone dla poprzednich 7 bajtów.

Jeżeli CRC policzone przez mikrokontroler jest takie samo jak CRC zapisane w ROM, dane uznajemy za poprawnie odczytane. Różne wartości świadczą o błędzie odczytu lub transmisji.

**Zadanie 1.2 — Sprawdzenie CRC kodu ROM**

Moduł `onewire` umożliwia obliczenie CRC-8.
Dla pierwszego znalezionego urządzenia sprawdź:

```
rom = roms[0]

calculated_crc = ow.crc8(rom[:7])
stored_crc = rom[7]

print("CRC policzone:", hex(calculated_crc))
print("CRC w ROM:    ", hex(stored_crc))
print("Poprawne:", calculated_crc == stored_crc)
```

Porównaj obie wartości i wyjaśnij, co oznacza ich zgodność oraz co mogłaby oznaczać różnica między nimi.

# 4 Zadanie 2: Pomiar temperatury

DS18B20 nie zwraca nowego pomiaru natychmiast po wywołaniu funkcji odczytu. Najpierw należy uruchomić konwersję temperatury, poczekać na jej zakończenie, a dopiero potem odczytać wynik.

Przy domyślnej rozdzielczości 12 bitów maksymalny czas konwersji wynosi około 750 ms.

**Zadanie 2.1 — Pierwszy pomiar temperatury**

Uruchom:

```
from utime import sleep_ms

while True:
    ds.convert_temp()

    sleep_ms(750)

    for rom in roms:
        temperature = ds.read_temp(rom)
        print("T = {:.2f} C".format(temperature))
```

Obserwuj temperaturę przez około minutę. Następnie ogrzej czujnik palcami i sprawdź, jak szybko zmienia się wskazanie.

**Po co `convert_temp()` i opóźnienie?**

Polecenie `convert_temp()` rozpoczyna fizyczny pomiar temperatury w czujniku. Funkcja `read_temp()` odczytuje wynik zapisany już w pamięci DS18B20.

Przy rozdzielczości 12 bitów konwersja może trwać do około 750 ms, dlatego natychmiastowy odczyt mógłby zwrócić wynik poprzedniej konwersji.

# 5 Zadanie 3: Co dzieje się na linii 1-Wire?

Transmisja 1-Wire nie ma osobnego zegara. Informacja jest kodowana czasem trwania impulsów na linii DQ.

Każda komunikacja rozpoczyna się sekwencją RESET. Następnie urządzenie odpowiada impulsem *presence*, informując urządzenie sterujące o swojej obecności.

**Pomiar 3.1 — RESET i presence pulse**

Podłącz wejście cyfrowe Analog Discovery II do linii DQ DS18B20 oraz połącz masę analizatora z GND płytki.
Uruchom w pętli:

```
from utime import sleep_ms

while True:
    presence = ow.reset()

    print(presence)

    sleep_ms(20)
```

W narzędziu Logic zaobserwuj:

- impuls RESET generowany przez Raspberry Pi Pico,
- odpowiedź *presence pulse* generowaną przez DS18B20.

Za pomocą kursorów zmierz czasy impulsów i porównaj je z wartościami podanymi w dokumentacji czujnika.

**1-Wire jest interfejsem zależnym od czasu**

W UART odbiornik zna szybkość transmisji, a w SPI urządzenia korzystają z osobnego zegara. W 1-Wire znaczenie sygnału zależy od tego, **jak długo linia pozostaje w stanie niskim lub wysokim**.
Dlatego poprawna obsługa 1-Wire wymaga precyzyjnego generowania impulsów. W MicroPythonie szczegóły te ukrywa moduł `onewire`.

# 6 Zadanie 4: Karta microSD i system plików

Gniazdo microSD na płytce można obsługiwać w trybie SPI. W tym trybie wykorzystujemy:

- CLK jako SCK,
- CMD jako MOSI,
- DAT0 jako MISO,
- DAT3 jako CS.

Wykorzystamy osobny interfejs SPI1:

`SCK=GP10`, `MOSI=GP11`, `MISO=GP12`, `CS=GP13`.

**Sterownik karty a system plików**

Karta microSD jest urządzeniem blokowym. Sterownik `sdcard.py` umożliwia odczyt i zapis bloków danych, ale nie tworzy jeszcze plików takich jak `log.csv`. Dopiero warstwa systemu plików FAT pozwala używać:

- nazw plików,
- katalogów,
- funkcji `open()`,
- operacji odczytu i zapisu tekstu.

**Zadanie 4.1 — Detekcja obecności karty**

Przed zamontowaniem karty połącz wyprowadzenie **DETECT** gniazda microSD z `GP15`.

Uruchom:

```
from machine import Pin
from utime import sleep_ms

detect = Pin(15, Pin.IN)

while True:
    print(detect.value())
    sleep_ms(300)
```

Sprawdź stan wejścia:

- bez karty,
- po włożeniu karty.

Na podstawie obserwacji określ, który poziom logiczny oznacza obecność karty.

**Uwaga**

Nie wyjmuj zamontowanej karty podczas zapisu danych. Przed fizycznym wyjęciem karty należy zamknąć otwarte pliki i odmontować system plików.

## Zadanie 4.2 — Montowanie karty microSD

Skopiuj do pamięci Raspberry Pi Pico dostarczony plik `sdcard.py`.
Następnie uruchom:

```
from machine import Pin, SPI
import os
import vfs
import sdcard

spi_sd = SPI(
    1,
    baudrate=1_000_000,
    polarity=0,
    phase=0,
    sck=Pin(10),
    mosi=Pin(11),
    miso=Pin(12)
)

cs_sd = Pin(13, Pin.OUT, value=1)

sd = sdcard.SDCard(spi_sd, cs_sd)

fs = vfs.VfsFat(sd)
vfs.mount(fs, "/sd")

print(os.listdir("/sd"))
```

Jeżeli karta została poprawnie zamontowana, program powinien wyświetlić listę plików znajdujących się na karcie.

### Co robią `SDCard`, `VfsFat` i `mount()`?

W kodzie pojawiają się trzy kolejne warstwy:

1. `sdcard.SDCard(spi_sd, cs_sd)` tworzy obiekt reprezentujący **urządzenie blokowe**. Sterownik potrafi odczytywać i zapisywać sektory karty przez SPI, ale na tym etapie nie operujemy jeszcze na plikach.
2. `vfs.VfsFat(sd)` tworzy obiekt systemu plików **FAT** korzystający z urządzenia blokowego `sd`.
3. `vfs.mount(fs, '/sd')` dołącza ten system plików do wirtualnego systemu plików MicroPythona pod ścieżką `/sd`.

Dopiero po zamontowaniu można wykonywać zwykłe operacje na plikach, np. `open('/sd/test.txt', ...)`.
Polecenie:

```
os.listdir('/sd')
```

zwraca listę plików i katalogów znajdujących się w katalogu głównym karty. Przed wyjęciem karty należy wykonać:

```
vfs.umount('/sd')
```

co odłącza system plików od VFS. Dzięki temu system wie, że karta nie będzie już używana.

**Co oznacza /sd?**

Po zamontowaniu system plików karty staje się częścią drzewa plików MicroPythona.
Przykładowo:

- `/sd/test.txt` — plik na karcie microSD,
- `/main.py` — plik w pamięci flash Raspberry Pi Pico.

Dzięki temu z kartą można pracować przez standardową funkcję `open()`.

# 7 Zadanie 5: Zapis i odczyt plików

**Zadanie 5.1 — Pierwszy plik na karcie**

Poniższy program jest kompletnym przykładem: inicjalizuje kartę, montuje system plików, zapisuje plik, zamyka go i na końcu odmontowuje kartę.

```
from machine import Pin, SPI
import vfs
import sdcard

spi_sd = SPI(
    1,
    baudrate=1_000_000,
    polarity=0,
    phase=0,
    sck=Pin(10),
    mosi=Pin(11),
    miso=Pin(12)
)

cs_sd = Pin(13, Pin.OUT, value=1)

sd = sdcard.SDCard(spi_sd, cs_sd)
fs = vfs.VfsFat(sd)
vfs.mount(fs, '/sd')

with open('/sd/test.txt', 'w') as file:
```

```
    file.write('PSM - Laboratorium 7\\n')
    file.write('Karta microSD dziala!\\n')

vfs.umount('/sd')

print('Plik zapisany.')
```

Po wykonaniu programu ponownie zamontuj kartę tak jak w zadaniu 4.2. Następnie odczytaj plik. **Poniższy fragment nie jest pełnym programem** — zakłada, że karta została wcześniej zainicjalizowana i zamontowana pod ścieżką `/sd`:

```
with open('/sd/test.txt', 'r') as file:
    text = file.read()

print(text)
```

Po odczycie odmontuj kartę za pomocą `vfs.umount('/sd')`. Sprawdź również, czy plik jest widoczny po odczytaniu karty na komputerze.

### Tryby otwierania pliku

Najczęściej używane tryby funkcji `open()`:

- `'r'` — odczyt,
- `'w'` — zapis od początku; poprzednia zawartość jest usuwana,
- `'a'` — dopisywanie na końcu istniejącego pliku.

W kodzie Pythona można stosować zarówno pojedyncze, jak i podwójne cudzysłowy do zapisu napisów. W tekście instrukcji używamy apostrofów, ponieważ znak podwójnego cudzysłowu ma specjalne znaczenie w polskiej konfiguracji LaTeX.

Konstrukcja:

`with open(...) as file:`

automatycznie zamyka plik po zakończeniu bloku.

### Zadanie 5.2 — Plik CSV z temperaturą

Utwórz plik:

`/sd/temperature.csv`

z nagłówkiem:

`sample,temperature_C`

Wykonaj 20 pomiarów temperatury i zapisuj je kolejno do pliku.

Przykładowa linia:

`7,23.44`

Po zakończeniu pomiarów odczytaj cały plik w Thonny i sprawdź jego zawartość.

# 8 Zadanie 6: RTC + DS18B20 + microSD

W prawdziwym rejestratorze danych numer próbki nie wystarcza. Potrzebujemy informacji, **kiedy** wykonano pomiar. Wykorzystamy więc RTC PCF8563 z Laboratorium 5.

Podłącz RTC do:

SDA → `GP16`, SCL → `GP17`.

**Zadanie 6.1 — Znacznik czasu z RTC**



Wykorzystaj funkcję odczytującą RTC przygotowaną podczas Laboratorium 5. Na jej podstawie przygotuj napis w formacie:

`YYYY-MM-DD,HH:MM:SS`

np.:

`2026-11-02,17:35:08`.

W tym zadaniu **wyświetl przygotowany znacznik czasu w konsoli Thonny**. Nie zapisuj go jeszcze do pliku — zapis znacznika czasu razem z temperaturą będzie częścią zadania końcowego.

# 9 Zadanie końcowe: Rejestrator temperatury

**Zadanie 7.1 — Logger CSV**

Zbuduj rejestrator temperatury.
**Każde uruchomienie programu powinno tworzyć nowy plik**, tak aby wcześniejsze pomiary nie zostały nadpisane. Kolejne pliki nazwij:

`dane1.csv`, `dane2.csv`, `dane3.csv`, ...

Program powinien:

1. sprawdzić obecność karty microSD,
2. zamontować kartę,
3. znaleźć pierwszy wolny numer pliku,
4. utworzyć nowy plik `daneN.csv`,
5. zapisać nagłówek CSV,
6. odczytywać temperaturę z DS18B20,
7. pobierać aktualny czas i datę z RTC,
8. co 2 s zapisywać kolejny pomiar,
9. jednocześnie wyświetlać aktualny pomiar w konsoli Thonny,
10. po wykonaniu 30 pomiarów zamknąć plik i odmontować kartę.

Plik powinien mieć postać:

```
date,time,temperature_C
2026-11-02,17:35:08,23.44
2026-11-02,17:35:10,23.50
2026-11-02,17:35:12,23.50
```

**Wskazówka dotycząca wyboru nazwy pliku**

Funkcja:

```
files = os.listdir('/sd')
```

zwraca listę nazw plików znajdujących się na karcie.
W Pythonie operator `in` pozwala sprawdzić, czy dany element znajduje się na liście. Na przykład:

```
if 'dane1.csv' in files:
    print('Plik istnieje')
```

Rozpocznij od numeru:

```
number = 1
```

Następnie przygotuj nazwę pliku:

```
name = 'dane{}.csv'.format(number)
```

i sprawdzaj w pętli, czy znajduje się ona już na liście plików:

```
while name in files:
    number += 1
    name = 'dane{}.csv'.format(number)
```

Po zakończeniu pętli zmienna `name` zawiera pierwszą wolną nazwę pliku. Pełną ścieżkę utwórz za pomocą:

```
filename = '/sd/' + name
```

Zwróć uwagę, że funkcja `os.listdir('/sd')` zwraca same nazwy plików, np. `dane1.csv`, a nie pełne ścieżki do plików.
Ponieważ przy każdym uruchomieniu tworzysz nowy plik, otwórz go w trybie `'w'`.
Po wykonaniu 30 pomiarów plik powinien zostać zamknięty, a karta prawidłowo odmontowana:

```
vfs.umount('/sd')
```

Wyjmij kartę, otwórz utworzony plik na komputerze i sprawdź poprawność danych. Uruchom program ponownie i upewnij się, że powstał kolejny plik, a poprzedni pozostał niezmieniony.

# 10 Podsumowanie

**Po tych zajęciach powinieneś umieć**



- wyjaśnić podstawową zasadę działania magistrali 1-Wire,
- odnaleźć urządzenia 1-Wire na wspólnej linii,
- wyjaśnić rolę 64-bitowego kodu ROM,
- wykonać pomiar temperatury za pomocą DS18B20,
- rozpoznać sekwencję RESET i presence pulse,
- podłączyć kartę microSD w trybie SPI,
- wyjaśnić różnicę pomiędzy urządzeniem blokowym a systemem plików,
- zamontować i odmontować system plików FAT,
- zapisywać i odczytywać pliki tekstowe,
- tworzyć pliki CSV z pomiarami i automatycznie wybierać kolejną nazwę pliku,
- połączyć czujnik, RTC i pamięć masową w kompletny rejestrator danych.

**Uwaga**



Przed opuszczeniem stanowiska:

1. zatrzymaj program,
2. zamknij wszystkie otwarte pliki,
3. odmontuj kartę microSD,
4. dopiero wtedy wyjmij kartę z gniazda,
5. usuń z pamięci Raspberry Pi Pico pliki utworzone podczas laboratorium.