Podstawy Systemów Mikroprocesorowych Laboratorium

Laboratorium 6: SPI — ADC, DAC i wyświetlacz TFT

Autorzy: Dariusz Tefelski, Angelika Tefelska

Spis treści

1 Wprowadzenie **1**
1.1 Tryby SPI 1

2 Wyposażenie stanowiska **2**

3 Zadanie 1: Konfiguracja magistrali SPI **3**

4 Zadanie 2: Przetwornik cyfrowo-analogowy MCP4921 **4**
4.1 Połączenie MCP4921 4

5 Zadanie 3: Obserwacja transmisji SPI **7**

6 Zadanie 4: Zewnętrzny przetwornik ADC MCP3202 **8**
6.1 Połączenie MCP3202 8

7 Zadanie 5: Wyświetlacz graficzny ST7789V **11**
7.1 Połączenie wyświetlacza 11

8 Zadanie 6: Trzy urządzenia na jednej magistrali SPI **15**

9 Podsumowanie **16**

1 Wprowadzenie

Cel ćwiczenia

Celem ćwiczenia jest poznanie synchronicznego interfejsu szeregowego SPI oraz wykorzystanie go do komunikacji z zewnętrznym przetwornikiem cyfrowo-analogowym MCP4921, analogowo-cyfrowym MCP3202 oraz graficznym wyświetlaczem TFT HC28010-D ze sterownikiem ST7789V. Wszystkie trzy urządzenia zostaną podłączone do wspólnej magistrali SPI, ale będą wybierane niezależnymi liniami CS.

Na Laboratorium 5 poznaliśmy I^2C, gdzie wiele urządzeń współdzieli dwie linie i jest rozróżnianych na podstawie adresów. SPI działa inaczej. Urządzenia mogą współdzielić linie zegara i danych, ale każde urządzenie jest zazwyczaj wybierane osobnym sygnałem **CS** (*Chip Select*).

Podstawowe linie SPI

Typowa magistrala SPI wykorzystuje:

- **SCK** (*Serial Clock*) — sygnał zegarowy generowany przez urządzenie sterujące,
- **MOSI** (*Controller Out, Peripheral In*) — dane wysyłane z mikrokontrolera do urządzenia,
- **MISO** (*Controller In, Peripheral Out*) — dane wysyłane z urządzenia do mikrokontrolera,
- **CS** (*Chip Select*) — wybór konkretnego urządzenia, najczęściej aktywny stanem niskim.

W starszej dokumentacji spotkasz również rozwinięcia:

MOSI = *Master Out, Slave In*, MISO = *Master In, Slave Out*.

SPI jest transmisją synchroniczną i pełnodupleksową

Każdy impuls SCK może jednocześnie przesuwać jeden bit w obu kierunkach: przez MOSI oraz MISO. Oznacza to, że podczas każdej transmisji mikrokontroler jednocześnie **wysyła i odbiera** dane, nawet jeżeli w konkretnym urządzeniu interesuje nas tylko jeden z tych kierunków.
MCP4921 wykorzystuje jedynie kierunek mikrokontroler → DAC, natomiast MCP3202 wymaga zarówno wysłania konfiguracji, jak i odebrania wyniku pomiaru.

1.1 Tryby SPI

Urządzenia SPI muszą uzgodnić nie tylko częstotliwość zegara, ale również sposób interpretacji zboczy SCK. Określają to parametry:

- **CPOL** — poziom spoczynkowy zegara,

- **CPHA** — zbocze, na którym odbiornik próbkuje dane.

W MicroPythonie odpowiadają im argumenty:

`polarity` oraz `phase`.

W tym laboratorium wykorzystamy **SPI Mode 0**:

`polarity=0`, `phase=0`.

2 Wyposażenie stanowiska

Do wykonania ćwiczenia potrzebne są:

- płytka dydaktyczna z Raspberry Pi Pico 2 W,
- przewód USB i przewody połączeniowe,
- przetwornik DAC MCP4921 znajdujący się na płytce,
- przetwornik ADC MCP3202 znajdujący się na płytce,
- potencjometr POT1 na płytce,
- wyświetlacz graficzny HC28010-D ze sterownikiem ST7789V znajdujący się na płytce,
- Analog Discovery II z programem WaveForms.

W tym laboratorium wykorzystamy sprzętowy interfejs SPI0:

Sygnał	**Raspberry Pi Pico 2 W**
MISO	`GP16`
SCK	`GP18`
MOSI	`GP19`

Linie CS będą zwykłymi wyjściami GPIO:

- MCP4921: `GP17`,
- MCP3202: `GP20`,
- wyświetlacz ST7789V: `GP21`.

Wyświetlacz wymaga dodatkowo:

- D/C: `GP22`,
- RST: `GP28`,
- BL: `GP27`.

Uwaga

Przed zmianą połączeń wyłącz zasilanie. W całym laboratorium przetworniki zasilaj napięciem **3,3 V**. Dzięki temu również sygnały analogowe pozostają w bezpiecznym zakresie wejść Raspberry Pi Pico 2 W.

3 Zadanie 1: Konfiguracja magistrali SPI

Zadanie 1.1 — Sprawdzenie dokumentacji

Odszukaj dokumentacje układów MCP4921 i MCP3202:

https://www.microchip.com/en-us/product/MCP4921

https://www.microchip.com/en-us/product/MCP3202

W dokumentacji znajdź:

1. informację, jakie tryby SPI obsługuje MCP4921,
2. informację, jakie tryby SPI obsługuje MCP3202,
3. maksymalną częstotliwość zegara SPI obu układów,
4. informację, na którym zboczu SCK dane są próbkowane.

Sprawdź, czy oba układy mogą pracować w trybie używanym podczas tego laboratorium.

Zadanie 1.2 — Inicjalizacja SPI0

Uruchom:

```
from machine import Pin, SPI

spi = SPI(
    0,
    baudrate=1_000_000,
    polarity=0,
    phase=0,
    bits=8,
    firstbit=SPI.MSB,
    sck=Pin(18),
    mosi=Pin(19),
    miso=Pin(16)
)

print(spi)
```

SPI nie zarządza automatycznie linią CS

Obiekt `SPI` obsługuje linie SCK, MOSI i MISO. Linię CS najczęściej sterujemy niezależnie jako zwykłe wyjście GPIO.
Przykładowo, jeżeli CS urządzenia został podłączony do `GP17`, najpierw tworzymy obiekt `Pin`:

```
cs = Pin(17, Pin.OUT, value=1)
```

Argument `Pin.OUT` konfiguruje pin jako wyjście. Wartość początkowa `value=1` pozostawia urządzenie **nieaktywne**, ponieważ w większości układów SPI — w tym MCP4921 i MCP3202 — linia CS jest aktywna stanem niskim.
Schemat pojedynczej transakcji wygląda więc typowo tak:

```
cs.value(0)       # wybierz urzadzenie
spi.write(data)
cs.value(1)       # zakoncz transakcje
```

Jeżeli do tej samej magistrali podłączonych jest kilka urządzeń, w danym momencie aktywna powinna być tylko linia CS urządzenia, z którym chcemy się komunikować.

4 Zadanie 2: Przetwornik cyfrowo-analogowy MCP4921

MCP4921 jest 12-bitowym przetwornikiem DAC. Otrzymuje przez SPI 16-bitowe słowo składające się z:

- 4 bitów konfiguracyjnych,
- 12-bitowej wartości cyfrowej.

Dla $V_{REF} = 3{,}3\,\mathrm{V}$ i wzmocnienia $1\times$ idealne napięcie wyjściowe wynosi:

$$U_{OUT} = V_{REF}\frac{N}{4096}, \tag{1}$$

gdzie N przyjmuje wartości od 0 do 4095.

4.1 Połączenie MCP4921

Wykonaj połączenia:

MCP4921	Połączenie
VDD	3,3 V
VREF	3,3 V
VSS	GND
SCK	`GP18`
SDI	`GP19`
CS	`GP17`
LDAC	GND
VOUT	Analog Discovery II

Zadanie 2.1 — Format słowa wysyłanego do DAC

Odszukaj w dokumentacji MCP4921 opis 16-bitowego słowa zapisu. Zidentyfikuj bity:

- BUF,
- GA,
- SHDN,
- D11–D0.

W tym laboratorium użyjemy konfiguracji:

- wejście VREF niebuforowane,
- wzmocnienie 1×,
- wyjście DAC aktywne.

Dla takiej konfiguracji cztery najbardziej znaczące bity można zapisać jako wartość `0x3`. Dlatego kompletne 16-bitowe słowo przygotujemy przez:

```
word = 0x3000 | value
```

gdzie `value` jest liczbą od 0 do 4095.

Co robi operator |?

Operator `|` wykonuje bitową operację OR.
Stała:

`0x3000`

ustawia wymagane bity konfiguracyjne, natomiast `value` zajmuje 12 najmłodszych bitów.
Na przykład dla:

`value = 2048 = 0x800`

otrzymamy:

`0x3000 | 0x0800 = 0x3800.`

Zadanie 2.2 — Pierwsze napięcie z DAC

Uruchom:

```
cs_dac = Pin(17, Pin.OUT, value=1)

def dac_write(value):
    value = max(0, min(4095, value))

    word = 0x3000 | value

    data = bytes([
        (word >> 8) & 0xFF,
        word & 0xFF
    ])

    cs_dac.value(0)
    spi.write(data)
    cs_dac.value(1)

dac_write(2048)
```

Zmierz napięcie VOUT za pomocą Analog Discovery II.
Następnie sprawdź wartości:

$$0, \quad 1024, \quad 2048, \quad 3072, \quad 4095.$$

Dla każdej wartości oblicz napięcie idealne i porównaj je z pomiarem.

Jak z 16-bitowego słowa powstają dwa bajty?

MCP4921 wymaga przesłania jednego słowa o długości 16 bitów. Interfejs SPI skonfigurowaliśmy jednak do przesyłania elementów 8-bitowych, dlatego słowo trzeba rozdzielić na **starszy** i **młodszy bajt**.
Dla przykładowej wartości:

`word = 0x3800`

wykonujemy:

```
high_byte = (word >> 8) & 0xFF
low_byte  = word & 0xFF

data = bytes([high_byte, low_byte])
```

Krok po kroku:

- `word >> 8` przesuwa słowo o 8 bitów w prawo, dzięki czemu starszy bajt trafia na osiem najmłodszych pozycji,
- `& 0xFF` pozostawia tylko osiem najmłodszych bitów,
- `word & 0xFF` bez przesuwania wybiera młodszy bajt,
- `bytes([high_byte, low_byte])` tworzy obiekt zawierający dwa kolejne

bajty do wysłania przez SPI.
Dla `word = 0x3800` otrzymamy:

`high_byte = 0x38`, `low_byte = 0x00`,

czyli przez SPI zostaną przesłane kolejno bajty `0x38` i `0x00`.
W funkcji `dac_write()` ten sam zapis został wykonany krócej:

```
data = bytes([
    (word >> 8) & 0xFF,
    word & 0xFF
])
```

5 Zadanie 3: Obserwacja transmisji SPI

Pomiar 3.1 — Dekoder SPI w Analog Discovery II

Podłącz wejścia cyfrowe Analog Discovery II do:

- SCK — `GP18`,
- MOSI — `GP19`,
- CS DAC — `GP17`,
- GND — GND płytki.

W WaveForms uruchom **Logic** i dodaj dekoder SPI. Ustaw:

- Mode 0,
- MSB first,
- 8 bitów na słowo.

Uruchom ponownie:

`dac_write(2048)`

Sprawdź, czy analizator pokazuje kolejno bajty:

`0x38`, `0x00`.

Zaobserwuj również:

- kiedy linia CS przechodzi w stan niski,
- ile impulsów SCK występuje podczas transmisji,
- na którym zboczu SCK zmieniają się dane MOSI,
- na którym zboczu są próbkowane.

Zadanie 3.1 — Generowanie przebiegu schodkowego

Napisz program, który kolejno ustawia na DAC wartości:

$$0, \quad 512, \quad 1024, \quad \ldots, \quad 4095$$

i po każdej zmianie czeka około 100 ms.
Obserwuj jednocześnie:

- napięcie VOUT w narzędziu Scope,
- transmisję SPI w narzędziu Logic.



Następnie zmniejsz opóźnienie i krok wartości tak, aby na wyjściu powstał przebieg zbliżony do piły.

6 Zadanie 4: Zewnętrzny przetwornik ADC MCP3202

MCP3202 jest 12-bitowym, dwukanałowym przetwornikiem analogowo-cyfrowym komunikującym się przez SPI. W przeciwieństwie do MCP4921 transmisja jest tutaj dwukierunkowa:

- przez MOSI wysyłamy konfigurację pomiaru,
- przez MISO odbieramy wynik konwersji.

6.1 Połączenie MCP3202

Wykonaj połączenia:

MCP3202	Połączenie
VDD	3,3 V
VSS	GND
CLK	`GP18`
DIN	`GP19`
DOUT	`GP16`
CS	`GP20`
CH0	POT1
CH1	GND

Pozostaw MCP4921 podłączony do tej samej magistrali. Jego linia CS powinna znajdować się w stanie wysokim, gdy komunikujesz się z MCP3202.

Zadanie 4.1 — Odczyt formatu transmisji z dokumentacji



W dokumentacji MCP3202 odszukaj rysunek przedstawiający komunikację z użyciem 8-bitowych transmisji SPI.
Zidentyfikuj bity:

- Start,
- SGL/DIFF,
- ODD/SIGN,
- MSBF,
- Null bit,
- B11–B0.

W tym ćwiczeniu użyjemy:

- trybu *single-ended*,

- kanału CH0,
- wyniku odczytywanego od najbardziej znaczącego bitu.

Zadanie 4.2 — Odczyt kanału CH0

Przed uruchomieniem programu **połącz wyjście POT1 z wejściem CH0 przetwornika MCP3202**. Potencjometr na płytce generuje napięcie z zakresu około 0–3,3 V.
Uruchom:

```
from utime import sleep_ms

cs_adc = Pin(20, Pin.OUT, value=1)

def adc_read(channel=0):
    channel = 1 if channel else 0

    tx = bytes([
        0x01,
        0xA0 | (channel << 6),
        0x00
    ])

    rx = bytearray(3)

    cs_adc.value(0)
    spi.write_readinto(tx, rx)
    cs_adc.value(1)

    value = ((rx[1] & 0x0F) << 8) | rx[2]

    return value

while True:
    raw = adc_read(0)
    voltage = raw * 3.3 / 4095

    print(raw, voltage)

    sleep_ms(200)
```

Obracaj potencjometrem POT1 i sprawdź zakres otrzymywanych wartości.

Jak powstaje bufor `tx` dla MCP3202?

MCP3202 nie otrzymuje zwykłej wartości liczbowej, lecz sekwencję bitów konfiguracyjnych. Przy sprzętowym SPI najwygodniej przesłać je jako trzy pełne bajty:

```
tx = bytes([
    0x01,
    0xA0 | (channel << 6),
    0x00
])
```

Znaczenie poszczególnych bajtów jest następujące:

- `0x01` — siedem zer poprzedzających transmisję oraz bit START ustawiony na 1,
- drugi bajt zawiera bity konfiguracyjne:
  - bit 7: `SGL/DIFF = 1` — pomiar *single-ended*,
  - bit 6: `ODD/SIGN` — wybór kanału CH0 lub CH1,
  - bit 5: `MSBF = 1` — wynik zwracany od najbardziej znaczącego bitu,
  - pozostałe bity są nieistotne (*don't care*),
- `0x00` — bajt nie niesie już konfiguracji, ale jego wysłanie generuje osiem kolejnych impulsów SCK potrzebnych do odebrania młodszych bitów wyniku.

Dla kanału CH0:

`0xA0 | (0 << 6) = 0xA0`,

natomiast dla kanału CH1:

`0xA0 | (1 << 6) = 0xE0`.

Operator `<< 6` przesuwa numer kanału na pozycję bitu `ODD/SIGN`, a operator `|` ustawia go razem z pozostałymi wymaganymi bitami konfiguracyjnymi.

Alternatywnie: napięcie z Analog Discovery II

Zamiast potencjometru możesz wykorzystać generator Wavegen w Analog Discovery II jako regulowane źródło napięcia stałego. Ustaw przebieg typu **DC** i zmieniaj poziom w zakresie 0–3,3 V.

Wyjście generatora podłącz do CH0 MCP3202, a masę Analog Discovery II połącz z GND płytki.

Nie ustawiaj napięcia ujemnego ani większego od 3,3 V.

Dlaczego używamy `write_readinto()`?

SPI przesyła dane jednocześnie w obu kierunkach. Funkcja:

`spi.write_readinto(tx, rx)`

podczas tych samych impulsów zegara:

- wysyła bajty z bufora `tx` przez MOSI,
- zapisuje bajty odebrane przez MISO do bufora `rx`.

MCP3202 potrzebuje zegara również wtedy, gdy już wysłał konfigurację i tylko zwraca wynik pomiaru. Dlatego trzeci bajt `0x00` jest przede wszystkim źródłem kolejnych impulsów SCK.

Składanie wyniku 12-bitowego

Po transmisji:

```
rx[1] & 0x0F
```

pozostawia cztery najmłodsze bity drugiego odebranego bajtu. Są to bity B11–B8 wyniku.
Przesunięcie:

```
(rx[1] & 0x0F) << 8
```

ustawia je na odpowiednich pozycjach, a operator OR dołącza osiem młodszych bitów z `rx[2]`.

7 Zadanie 5: Wyświetlacz graficzny ST7789V

Na płytce dydaktycznej znajduje się kolorowy wyświetlacz TFT HC28010-D o rozdzielczości 240×320 pikseli. Zastosowany kontroler ST7789V komunikuje się z mikrokontrolerem przez SPI.

W przeciwieństwie do prostych przetworników ADC/DAC wyświetlacz wymaga wielu poleceń inicjalizacyjnych oraz przesyłania dużej ilości danych. W tym zadaniu wykorzystamy dlatego gotowy plik `st7789.py`, ale nadal świadomie konfigurujemy magistralę SPI i wszystkie linie sterujące.

7.1 Połączenie wyświetlacza

Pozostaw MCP4921 i MCP3202 podłączone do magistrali. Dołącz wyświetlacz:

Wyświetlacz	Raspberry Pi Pico 2 W
SCL/SCK	GP18
SDA/MOSI	GP19
SDO/MISO	GP16
CSX	GP21
D/C	GP22
RST	GP28
BL	GP27

Dodatkowe linie D/C, RST i BL

Oprócz typowych linii SPI wyświetlacz wykorzystuje:

- **D/C** — informuje kontroler, czy przesyłany bajt jest poleceniem, czy danymi,
- **RST** — sprzętowy reset kontrolera,
- **BL** — sterowanie podświetleniem przez tranzystor znajdujący się na płytce dydaktycznej.

Linia D/C nie zastępuje CS. CS wybiera urządzenie na magistrali, natomiast D/C określa znaczenie przesyłanych bajtów.

Zadanie 5.1 — Pierwsza grafika

Skopiuj do pamięci Raspberry Pi Pico dostarczony plik `st7789.py`.
Następnie uruchom:

```
from machine import Pin, SPI
from st7789 import (
    ST7789, BLACK, WHITE,
    RED, GREEN, BLUE, YELLOW
)

spi = SPI(
    0,
    baudrate=800_000,
    polarity=0,
    phase=0,
    bits=8,
    firstbit=SPI.MSB,
    sck=Pin(18),
    mosi=Pin(19),
    miso=Pin(16)
)

display = ST7789(
    spi,
    cs=21,
    dc=22,
    rst=28,
    bl=27,
    width=240,
    height=320
)

display.fill(BLACK)

```

```
display.text("PSM - SPI", 10, 10, WHITE)

display.fill_rect(20, 50, 60, 40, RED)
display.fill_rect(90, 50, 60, 40, GREEN)
display.fill_rect(160, 50, 60, 40, BLUE)

display.rect(20, 120, 200, 50, YELLOW)
display.line(20, 200, 220, 280, WHITE)

display.show()
```

Sprawdź działanie programu, a następnie:

1. zmień napisy i kolory,
2. narysuj co najmniej dwa własne prostokąty,
3. za pomocą `polygon()` narysuj własny kształt zdefiniowany przez co najmniej pięć punktów (x, y), a następnie narysuj drugi kształt za pomocą `fill_polygon()`,
4. dodaj poziomy pasek o długości zależnej od wybranej liczby 0–100.

Własne kształty na podstawie współrzędnych

Biblioteka udostępnia funkcje `pixel()`, `line()`, `rect()` i `fill_rect()`. Dodatkowo:

- `polygon(points, color)` rysuje obrys wielokąta,
- `fill_polygon(points, color)` rysuje wielokąt wypełniony kolorem.

W obu przypadkach kształt definiuje się przez listę współrzędnych (x, y) kolejnych wierzchołków.
Przykład:

```
points = [
    (120, 180),
    (160, 240),
    (140, 240),
    (140, 280),
    (100, 280),
    (100, 240),
    (80, 240)
]

display.polygon(points, YELLOW)

points2 = [
    (120, 80),
    (160, 140),
    (140, 140),
```

```
    (140, 170),
    (100, 170),
    (100, 140),
    (80, 140)
]

display.fill_polygon(points2, GREEN)

display.show()
```

Funkcja `polygon()` łączy kolejne punkty odcinkami i na końcu łączy ostatni punkt z pierwszym. Funkcja `fill_polygon()` dodatkowo wypełnia wnętrze kształtu.
Można w ten sposób rysować zarówno wielokąty wypukłe, jak i wklęsłe, np. trójkąty, strzałki lub własne symbole. Należy jednak unikać wielokątów, których krawędzie przecinają się ze sobą.

Dlaczego tworzymy bufor obrazu?

Biblioteka `st7789.py` wykorzystuje moduł `framebuf`. Najpierw operacje takie jak `text()`, `rect()` czy `fill_rect()` zmieniają zawartość bufora w pamięci RAM mikrokontrolera. Dopiero:

`display.show()`

wysyła cały bufor do wyświetlacza.
Dla obrazu 240×320 w formacie RGB565 jeden piksel zajmuje 2 bajty, więc pełna klatka wymaga:

$$240 \cdot 320 \cdot 2 = 153600 \text{ B}. \tag{2}$$

To około 150 kB danych przesyłanych przy jednym pełnym odświeżeniu.

Jedna magistrala, różne wymagania częstotliwości

MCP3202 pracuje ze znacznie niższą maksymalną częstotliwością zegara niż wyświetlacz ST7789V. Dlatego urządzenia współdzielą linie SPI, ale przed komunikacją można zmieniać parametry magistrali.
Biblioteka wyświetlacza przed przesłaniem obrazu ustawia szybsze SPI. Funkcja odczytująca MCP3202 powinna natomiast przed swoją transakcją przywrócić bezpieczniejszą częstotliwość, np. 800 kHz.

Zmodyfikuj funkcję `adc_read()` z zadania 4.2, dodając na początku:

```
spi.init(
    baudrate=800_000,
    polarity=0,
    phase=0,
    bits=8,
```

```
    firstbit=SPI.MSB
)
```

Dzięki temu odczyt MCP3202 pozostaje poprawny również po wcześniejszym szybkim odświeżeniu TFT.

8 Zadanie 6: Trzy urządzenia na jednej magistrali SPI

W tej chwili wspólne linie SPI wykorzystują trzy urządzenia:

- MCP4921 — CS na `GP17`,
- MCP3202 — CS na `GP20`,
- ST7789V — CS na `GP21`.

SPI a I²C — wybór urządzenia

W I²C urządzenie wybieraliśmy poprzez adres wysyłany na SDA.
W SPI nie wysyłamy adresu urządzenia. Zamiast tego mikrokontroler aktywuje odpowiednią linię CS:

- `GP17`=0 — komunikacja z MCP4921,
- `GP20`=0 — komunikacja z MCP3202,
- `GP21`=0 — komunikacja z wyświetlaczem.

Pozostałe linie CS powinny w tym czasie pozostawać w stanie wysokim.

Zadanie 6.1 — Pętla DAC–ADC z prezentacją na TFT

Odłącz POT1 od CH0 MCP3202 i połącz:

VOUT MCP4921 → CH0 MCP3202.

Napisz program, który dla wartości:

0, 512, 1024, 1536, 2048, 2560, 3072, 3584, 4095

wykonuje kolejno:

1. zapis wartości do MCP4921,
2. krótkie odczekanie na ustalenie napięcia,
3. odczyt MCP3202,
4. obliczenie idealnego napięcia DAC,
5. obliczenie napięcia wynikającego z ADC,
6. obliczenie różnicy w mV,
7. wyświetlenie wyników na ekranie TFT.

Na wyświetlaczu powinny znaleźć się co najmniej:

- kod zadany DAC,
- kod odczytany ADC,

- napięcie idealne,
- napięcie zmierzone,
- błąd w mV,
- poziomy pasek pokazujący aktualny poziom napięcia.

Przykładowy wygląd:

```
DAC: 2048
ADC: 2036
Uset: 1.650 V
Uadc: 1.640 V
Error: -10 mV

[############------]
```

Wyznacz:

- największą wartość bezwzględną błędu,
- średni błąd dla wszystkich punktów.

Na zakończenie odpowiedz na pytanie: dlaczego nawet dwa 12-bitowe przetworniki połączone bezpośrednio ze sobą nie muszą zwracać dokładnie tego samego kodu cyfrowego?

9 Podsumowanie

Po tych zajęciach powinieneś umieć

- wskazać linie SCK, MOSI, MISO i CS interfejsu SPI,
- wyjaśnić znaczenie CPOL i CPHA,
- skonfigurować sprzętowy interfejs SPI w MicroPythonie,
- sterować linią CS jako zwykłym GPIO,
- wysyłać dane do urządzenia za pomocą `spi.write()`,
- jednocześnie wysyłać i odbierać dane za pomocą `write_readinto()`,
- konstruować 16-bitowe słowo polecenia dla MCP4921,
- odtworzyć 12-bitowy wynik pomiaru MCP3202 z dwóch bajtów,
- wykorzystać graficzny wyświetlacz SPI i wyjaśnić rolę linii D/C,
- wykorzystać kilka urządzeń na wspólnej magistrali SPI,
- wyjaśnić różnicę pomiędzy wyborem urządzenia w I^2C i SPI.

Uwaga

Przed opuszczeniem stanowiska zatrzymaj program i usuń z pamięci Raspberry Pi Pico pliki utworzone podczas tego laboratorium. Interpreter MicroPython powinien pozostać zainstalowany. Rozłącz dodatkowe połączenia pomiędzy DAC i ADC, pozostaw wyświetlacz w stanie zgodnym z organizacją stanowiska i uporządkuj przewody.