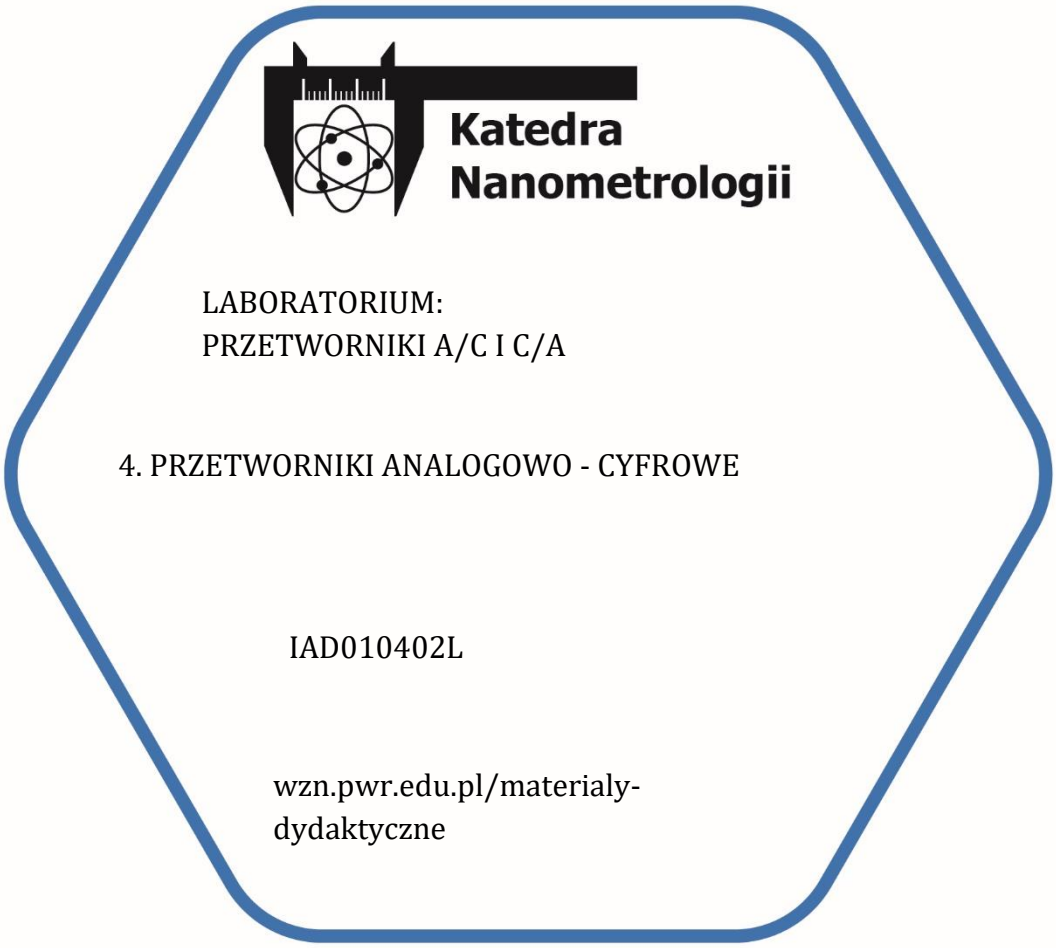


LABORATORIUM:
PRZETWORNIKI A/C I C/A

4. PRZETWORNIKI ANALOGOWO - CYFROWE

IAD010402L

wzn.pwr.edu.pl/materialy-dydaktyczne

DR HAB. INŻ. TOMASZ PIASECKI,
PROF. UCZELNI

SPIS TREŚCI

1. ZAGADNIENIA DO PRZYGOTOWANIA

Przed przystąpieniem do zajęć wymagane jest opanowanie wiedzy z następujących zagadnień:

1. Kondycjonowanie sygnału analogowego przed przetwarzaniem analogowo – cyfrowym.
2. Przetwarzanie analogowo – cyfrowe.
3. Widmo sygnału spróbkowanego, twierdzenie Nyquista-Shannona, aliasing.
4. Dyskretna transformata Fouriera: sposoby na jej obliczenie i prezentację, zjawisko wycieku widma i wpływ funkcji okienkujących na wyciek widma.
5. Szum kwantyzacji przetwornika A/C.
6. Gęstość widmowa mocy, sposoby jej estymowania.
7. Parametry przetworników analogowo – cyfrowych. SNR, SINAD, ENOB.
8. Sposób obsługi magistral SPI i równoległej w języku MicroPython.
9. Kodowanie informacji cyfrowej w kodzie naturalnym binarnym i U2. Jak działają fragmenty kodu wskazane w punkcie 3.4?
10. Funkcje do obliczania widm sygnałów dyskretnych, gęstości widmowej mocy oraz mocy i wartości skutecznej w programie Octave. Przynieś na zajęcia kod funkcji utworzonych w trakcie kursu Przetwarzanie Sygnałów. Przydadzą się tu.

Do realizacji niektórych punktów ćwiczenia niezbędna jest wiedza bezpośrednio powiązana z kursem Przetwarzanie Sygnałów. Wykonanie ćwiczenia będzie łatwiejsze po odświeżeniu znajomości pakietu Octave oraz informacji merytorycznych z ćwiczeń laboratoryjnych nr 3 i 4 tego kursu.

2. MAKIETA DYDAKTYCZNA

Na makiecie znajduje się kilka bloków funkcjonalnych:

- blok przetworników analogowo-cyfrowych,
- blok układów kondycjonowania sygnałów,
- blok filtrów antyaliasingowych,
- blok układów wprowadzających zniekształcenia.

W dalszej części przedstawione zostaną schematy blokowe bądź ideowe poszczególnych części makiety. Schematy ideowe zawierają nazwy poszczególnych komponentów. Możesz je w ten sposób samodzielnie odnaleźć na płycie drukowanej makiety.

Makieta zawiera również elementy, których rolą jest tylko zabezpieczenie wejść przed podaniem zbyt dużych napięć, prądów czy wyładowaniami elektrostatycznymi. Komponenty te nie mają zasadniczego wpływu na działanie makiety. Na schematach ideowych zaznaczono je na szaro.

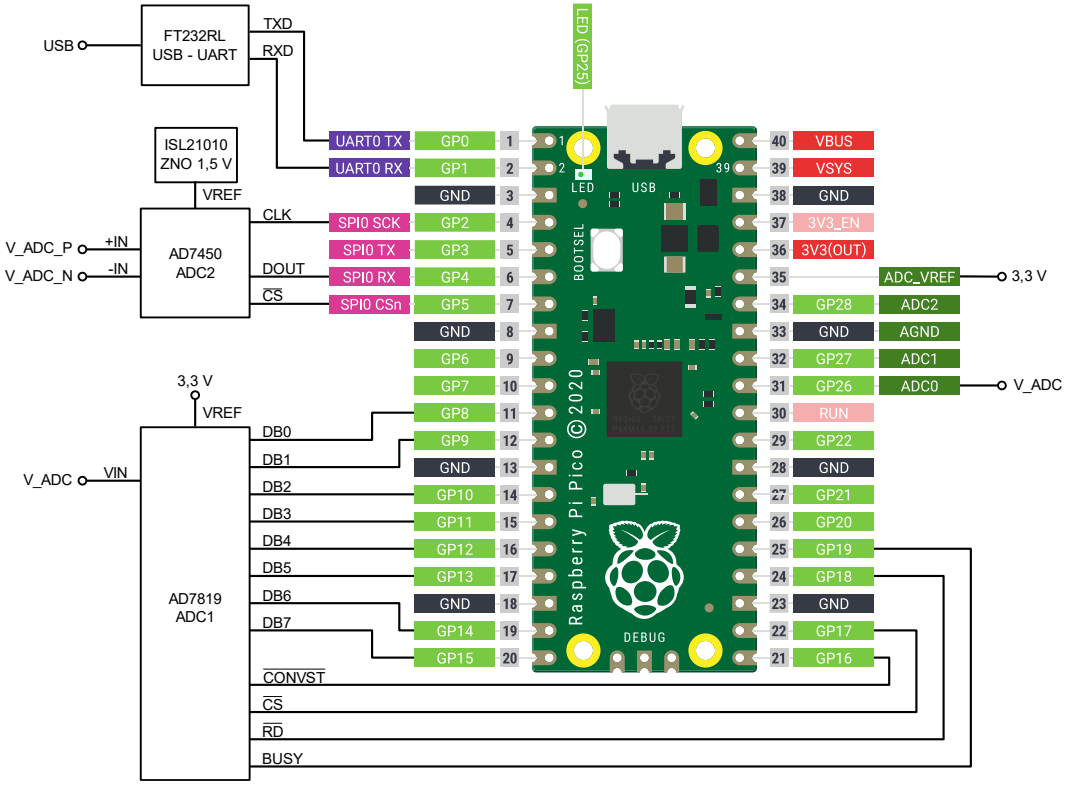
2.1. BLOK PRZETWORNIKÓW A/C

Makieta umożliwia korzystanie z trzech przetworników analogowo – cyfrowych o różnych właściwościach:

- przetwornik AD7450ARZ – 12-bitowy przetwornik SAR z wejściem różnicowym, komunikujący się magistralą cyfrową SPI,

- przetwornik AD7819YRZ – 8 bitowy przetwornik SAR z wejściem asymetrycznym, komunikujący się magistralą równoległą,
- przetwornik A/C wbudowany w mikrokontroler RP2040 na module Raspberry Pi Pico, 12 – bitowy SAR z wejściem asymetrycznym.

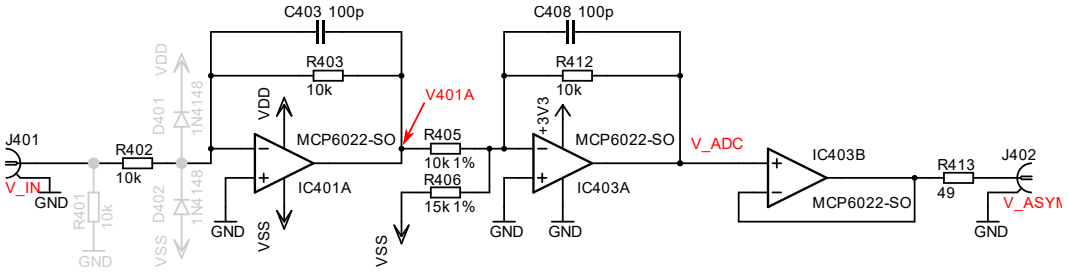
Schemat blokowy połączeń komponentów tego modułu przestawiono na rysunku 1.



Rysunek 1. Schemat blokowy modułu przetworników analogowo-cyfrowych.

2.2. BLOK UKŁADÓW KONDYCJONOWANIA SYGNAŁU

Przetworniki A/C wymagają różnych sygnałów, o różnych poziomach napięć. Układy kondycjonowania sygnału przekształcają sygnał asymetryczny, bipolarny z wejścia V_{IN} na napięcia potrzebne do wysterowania przetworników A/C.



Rysunek 2. Schemat układu przesuwnika napięć z poziomu 0 V do 1,65 V.

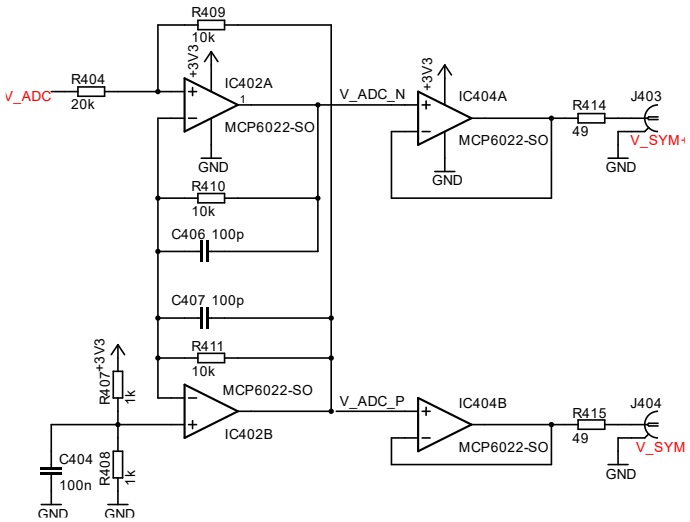
Układ z rysunku 2 to przesuwnik składowej stałej napięcia analogowego. Jego napięcie wejściowe to V_{IN}. Wzmacniacz IC_{401A} pracuje jako wzmacniacz odwracający:

$$V_{401A} = -\frac{R_{403}}{R_{402}} V_{IN}$$

Układ ten zasilany jest bipolarnie napięciami $V_{DD} = +2{,}5\ V, V_{SS} = -2{,}5\ V$. Jego wyjście podane jest do wzmacniacza odwracającego sumującego na układzie IC_{403A}. Jego napięcie wyjściowe to:

$$V_{ADC} = -\left(\frac{R_{412}}{R_{405}} V_{401A} + \frac{R_{412}}{R_{406}} V_{SS}\right) = \frac{R_{403} R_{412}}{R_{402} R_{405}} V_{IN} - \frac{R_{412}}{R_{406}} V_{SS} \cong V_{IN} + 1{,}65\ V$$

Napięcie V_{ADC} na wyjściu układu ma zatem dodaną składową stałą wynoszącą 1,65 V, co odpowiada połowie przedziału napięć przyjmowanych na wejścia stosowanych w makiecie przetworników z wejściem asymetrycznym i jest na nie kierowane. Napięcie to po przejściu przez wtórnik IC_{403B} jest dostępne na złączu V_{ASYM}. Ponieważ przy prawidłowej pracy makiety nie przewiduje się aby napięcie to przyjmowało wartości ujemne, wzmacniacze operacyjne w układzie IC_{403} zasilane są asymetrycznie napięciem $+3{,}3\ V$. Kondensatory C_{403} i C_{408} kształtują charakterystykę częstotliwościową układu ograniczając pasmo przenoszenia do około 150 kHz.



Rysunek 3. Układ tworzący napięcie różnicowe, symetryczne.

Napięcie V_{ADC} podane jest również na wejście układu z rysunku 3, wytwarzającego napięcie różnicowe, symetryczne, jakiego wymaga jeden z przetworników ADC na makiecie. Na wyjściu układu wytwarzane są dwa napięcia V_{ADC_P} i V_{ADC_N}, symetrycznie zależne od V_{IN}:

$$V_{ADC_P} - V_{ADC_N} = V_{IN}$$

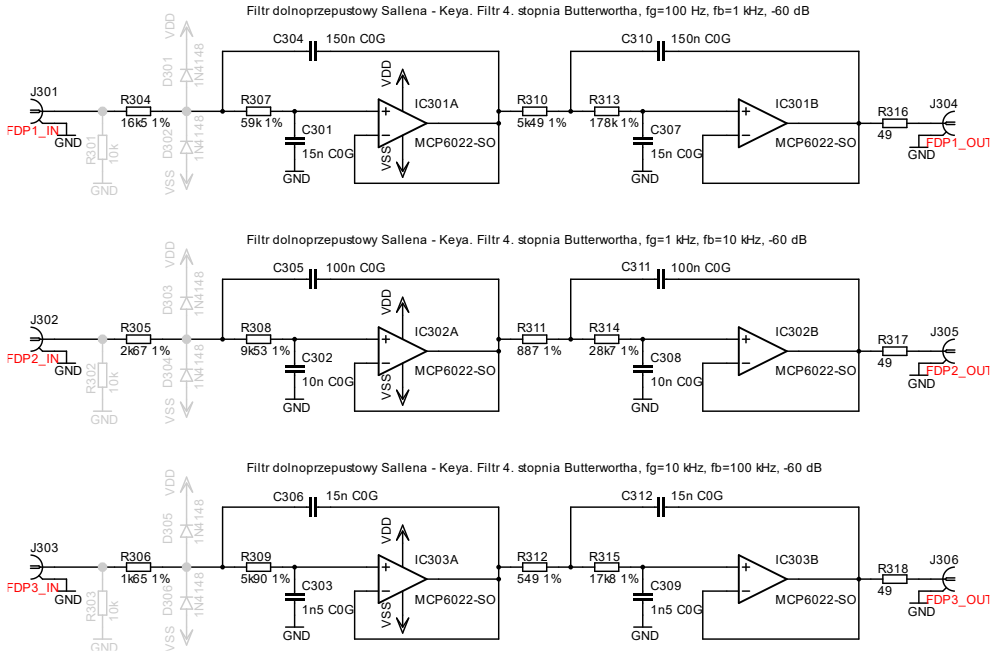
których składowa wspólna wynosi:

$$\frac{V_{ADC_P} + V_{ADC_N}}{2} \cong 1{,}65\ V$$

Układ ten oparto na wzmacniaczach w układzie IC_{402}. Napięcia wytwarzane przez układ po przejściu przez wtórniki dostępne są na złączach V_{SYM+} i V_{SYM-}.

2.3. BLOK FILTRÓW ANTYALIASINGOWYCH

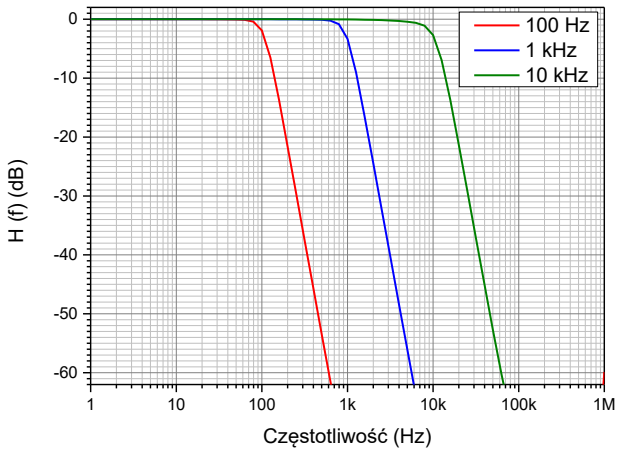
Większość przetworników analogowo – cyfrowych wymaga stosowania filtrów dolnoprzepustowych ograniczających pasmo sygnału na ich wejściach dla uniknięcia zjawiska aliasingu. Makietę wyposażono w zestaw trzech filtrów dolnoprzepustowych, których schemat przedstawiono na rysunku 4.



Rysunek 4. Schemat układów aktywnych filtrów antyaliasingowych Sallena – Keya.

Są to aktywne filtry dolnoprzepustowe Sallena – Keya, 4. rzędu o charakterystyce Butterwortha. Projektowano je na częstotliwości graniczne 100 Hz, 1 kHz i 10 kHz a ich stromość miała zapewniać co najmniej 60 dB tłumienia przy częstotliwości większej o rząd wielkości od częstotliwości granicznej. Filtry projektowano używając narzędzia dostarczonego przez Analog Devices, dostępnego pod adresem https://tools.analog.com/en/filterwizard/

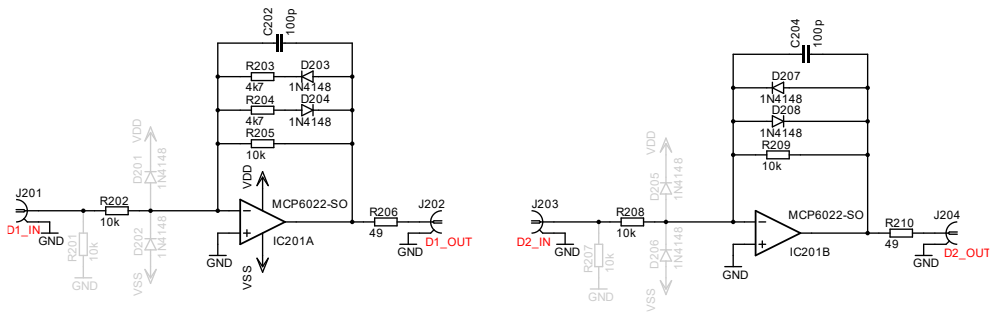
Zmierzone charakterystyki częstotliwościowe filtrów przedstawiono na rysunku 5.



Rysunek 5. Charakterystyki częstotliwościowe filtrów antyaliasingowych.

2.4. BLOK UKŁADÓW WPROWADZAJĄCYCH ZNIEKSZTAŁCENIA

Blok zawiera dwa wzmacniacze z elementami nieliniowymi w układzie sprzężeń zwrotnych. Służą one intencjonalnemu dodaniu zniekształceń nieliniowych do sygnału.



Rysunek 6. Schemat układów wprowadzających zniekształcenia.

Układ $D1$ jest wzmacniaczem odwracającym, który dla sygnałów o małej amplitudzie ma wzmocnienie $-1\frac{V}{V}$. Jeśli napięcie chwilowe sygnału przekracza napięcie przewodzenia diod D_{203} i D_{204} efektywna rezystancja w sprzężeniu zwrotnym wzmacniacza maleje aż do wartości rezystancji równoległego połączenia rezystorów w szeregu z diodami i R_{205}, przez co wzmocnienie również ulega zmianie. Układ ten wprowadza zatem zniekształcenia dla sygnałów o amplitudach większych od napięcia przewodzenia diody.

Układ $D2$ jest dyskryminatorem diodowym. Diody D_{207} i D_{208} ograniczają napięcia wyjściowe wzmacniacza do napięć przewodzenia tych diod „przycinając" sygnał.

3. KOMUNIKACJA Z PRZETWORNIKAMI A/C

Omówione zostaną ich najważniejsze właściwości oraz sposób sterowania przetwornikami A/C z poziomu programu realizowanego przez interpreter MicroPython dla RP2040.

3.1. PRZETWORNIK AD7450

Przetwornik AD7450 jest dołączony do makiety za pomocą magistrali SPI. Spójrz na rysunek 1 identyfikując numery wyprowadzeń Raspberry Pi Pico oznaczone GPxx, gdzie xx to numer wyprowadzenia, którymi sterowane są linie interfejsu SPI przetwornika AD7450. Potrzebne są numery wyprowadzeń zegara SCK, danych odbieranych RX oraz linii wyboru układu CS (od ang. *chip select*). Zobacz też, który z interfejsów SPI procesora RP2040 (SPI0, SPI1 czy inny) jest użyty do komunikacji.

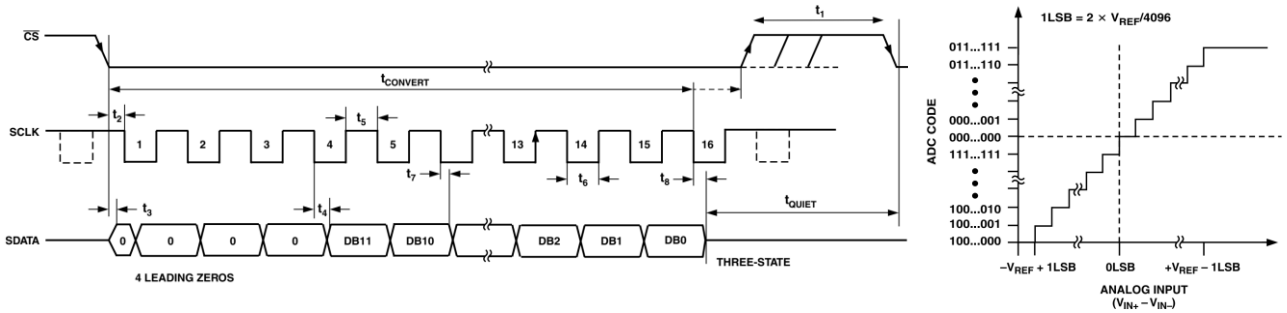
Zainicjowanie komunikacji z przetwornikiem może zostać zrealizowane następującymi poleceniami:

```
#zaimportowanie modułów Pin i SPI
import machine
from machine import Pin, SPI
#utworzenie obiektu klasy SPI obsługującej interfejs
spi = SPI(SPI_n, 1000000, bits=8, sck=Pin(pin_SCK), miso=Pin(pin_RX))
#utworzenie obiektu klasy Pin do obsługi wyprowadzenia Chip Select
cs = Pin(pin_CS, Pin.OUT, value=1)
```

Zaznaczone na czerwono fragmenty należy zastąpić numerem interfejsu SPI oraz numerami wyprowadzeń sterujących odpowiednimi liniami magistrali. Zmienne `spi` oraz `cs` są obiektami służącymi do obsługi magistrali oraz linii wyboru układu Chip Select. Ta ostatnia pracuje w logice ujemnej, czyli nieaktywny jej stan to stan wysoki. Stąd inicjowana jest ona jako wyjście (`Pin.OUT`) z początkową wartością 1 (`value=1`).

Procedura odczytu wskazana jest na rysunku 7a.

a) b)



Rysunek 7. Fragmenty dokumentacji technicznej przetwornika AD7450: a) diagram czasowy komunikacji SPI, b) wyniki przetwarzania napięcia [1].

Odczytanie wartości z przetwornika wymaga:

- aktywacji linii CS przez zmianę jej stanu na niski,
- odczytu 16 bitów na magistrali SPI,
- powrotu linii CS do stanu nieaktywnego (wysokiego).

Umożliwią to następujące kolejne polecenia języka MicroPython:

```
#zmiana stanu CS na niski
cs.off()
#odczyt 1 porcji 2x8 bitów SPI i umieszczenie wyniku w zmiennej data typu bytes
data = spi.read(2)
#zmiana stanu CS na wysoki
cs.on()
```

Po tym zmienna `data` zawierać będzie ciąg bajtów z kodem będącym wynikiem przetwarzania. Przetwornik jest 12-bitowy zatem wartość ta będzie miała 12 bitów, cztery najbardziej znaczące to zera. Poszczególne kody odpowiadają napięciom, zgodnie z rysunkiem 7b. Zwróć uwagę, że dane są 12-bitowymi słowami w kodzie U2. Sposób ich konwersji na typ całkowity `int` opisano w punkcie 3.4.

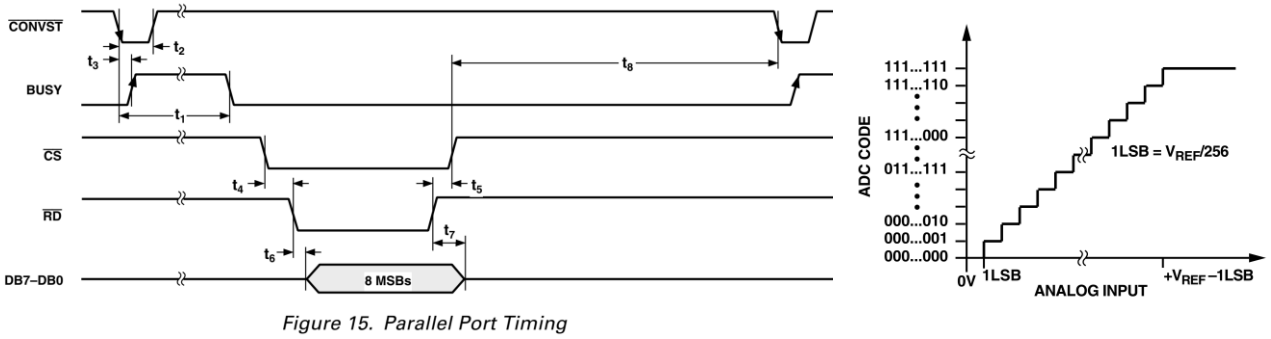
3.2. PRZETWORNIK AD7819

Ten 8-bitowy przetwornik SAR dołączony jest 8-bitową magistralą równoległą do przesyłu danych oraz czterema liniami dodatkowymi, jak na rysunku 1. Linie danych oznaczono DB0-DB7. Trzy z linii dodatkowych służą do sterowania pracą przetwornika, wszystkie pracują w logice ujemnej. Są to:

- $\overline{CONVST}$ – sygnalizacja początku przetwarzania,
- $\overline{CS}$ – wybór układu (chip select),
- $\overline{RD}$ – odczyt (read) aktywujący bufory trójstanowe magistrali danych w przetworniku,

oraz linia $BUSY$ którą przetwornik sygnalizuje czas trwania przetwarzania. Zależności czasowe na tych wyprowadzeniach przedstawia rysunek 8a.

a) b)



Rysunek 8. Fragmenty dokumentacji technicznej przetwornika AD7819: a) diagram czasowy komunikacji magistralą równoległą, b) wyniki przetwarzania napięcia [2].

Procedura przetwarzania wygląda zatem następująco:

- na początku linie $\overline{CONVST}$, $\overline{CS}$ i $\overline{RD}$ muszą być nieaktywne (stan wysoki),
- na linii $\overline{CONVST}$ należy wygenerować ujemny impuls krótszy niż 3 μs; jego czas trwania wpływa na tryb pracy przetwornika [2],
- przetwornik na linii $BUSY$ wystawi wysoki stan logiczny na czas trwania przetwarzania,
- gdy przetwarzanie się zakończy można aktywować układ i bufory wyprowadzeń sprowadzając w niski stan linie $\overline{CS}$ i $\overline{RD}$ …
- … a po upływie czasu t_6 (10 ns [2]) na liniach danych pojawi się wynik przetwarzania; może go wówczas odczytać mikrokontroler,
- po odczycie deaktywuje się linie $\overline{CS}$ i $\overline{RD}$.

Sterowanie tak dużą liczbą wyprowadzeń oraz odczyt wartości wyprowadzanej równolegle na wiele linii GPIO pojedynczymi poleceniami języka MicroPython jest możliwe choć niezbyt wygodne ani szybkie. Jako zadanie dodatkowe możesz spróbować taki kod napisać.

Mikrokontroler RP2040 będący sercem Raspberry Pi Pico posiada wbudowaną maszynę stanów (ang. *state machine*) obsługującą programowo polecenia wejścia/wyjścia (ang. PIO – programmed input/output) działającą niezależnie od CPU. Omówienie szczegółowo maszyn stanów PIO wykracza poza zakres tej instrukcji. Opisano ją w dokumentacji mikrokontrolera [3]. Poniższy fragment kodu zawiera procedurę PIO odczytu próbki z AD7819.

```
import machine
from machine import Pin, rp2

# procedura PIO odczytująca po 1 słowo z przetwornika AD7819 po każdym zapisie do kolejki FIFO
@rp2.asm_pio(#ustawienia pinow, kolejno: CONVST, CS, RD, BUSY
             set_init=(rp2.PIO.OUT_LOW, rp2.PIO.OUT_HIGH, rp2.PIO.OUT_HIGH, rp2.PIO.IN_LOW),
             #ustawienie wejść danych
             out_init=(rp2.PIO.IN_LOW,)*8)
def smproc():
    #poczatek petli wrap
    wrap_target()
    #oczekiwanie az do FIFO wejsciowego maszyny stanow zostanie przekazana wartosc
    pull(block)
    #ustawienie RD|CS|CONVST na 1|1|1 i opoznienie dodatkowe 2 takty zegara
    set(pins,0b111) [2]
    #ustawienie RD|CS|CONVST na 1|1|0 i opoznienie dodatkowe 1 takty zegara
    set(pins,0b110) [1]
    #oczekiwanie na 0 na pinie BUSY
    wait(0, pin, 3)
    #ustawienie RD|CS|CONVST na 0|0|0 - aktywacja wyjsc ADC
    set(pins,0b000) [0]
    #odczyt 8 bitow slowa z ADC do ISR
    in_(pins, 8)
    #umieszczenie ISR w kolejce FIFO
    push()
    #ustawienie RD|CS|CONVST na 1|1|0
    set(pins,0b110)
    #koniec programu i zapetlenie do wrap_target()
    wrap()
```

Procedura `smproc()` napisana jest z użyciem poleceń maszyny stanów PIO. Maszyna stanów realizuje kolejne kroki komunikacji. Procedura poprzedzona jest dyrektywą `@rp2.asm_pio()` definiującą rodzaj i liczbę wyprowadzeń mikrokontrolera obsługiwanych przez PIO.

Procedura PIO czeka z sygnalizowaniem przetwornikowi startu przetwarzania do momentu otrzymania danych przez PIO z rdzenia CPU mikrokontrolera. Wartość przekazana jest ignorowana, nie ma ona znaczenia. Po zakończeniu przetwarzania i odebraniu danych słowo z magistrali równoległej przekazywane jest do kolejki FIFO i staje się dostępna dla CPU.

Procedurę tę można użyć już w normalnym kodzie MicroPython w następujący sposób:

```
#inicjalizacja maszyny stanow 0, realizujace program w smproc, taktowanej 1 MHz,
#ktorej poczatkowym pinem do set jest Pin(16) a poczatkowym do in_ jest Pin(8)
sm = rp2.StateMachine(0, smproc, freq=1000000, set_base=Pin(16), in_base=Pin(8))
#aktywacja maszyny stanow
sm.active(1)
#wstawienie danej do kolejki wejściwej PIO, rozpoczynane jest przetwarzanie
sm.put(0)
#odczyt danej z PIO gdy te wstawi ją do swojej wyjściowej kolejki FIFO
code = sm.get()
```

Pierwsze polecenie tworzy obiekt `sm` klasy `rp2.StateMachine`, służący do obsługi maszyny stanów PIO. Konstruktor tej klasy ma cztery argumenty. Pierwszy to numer użytego PIO. Drugim argumentem jest delegacja do funkcji z kodem PIO. Parametr `freq` określa częstotliwość zegara taktującego PIO. Wartość 1 000 000 powoduje, że 1 takt zegara PIO trwa dokładnie 1 μs. Każdy rozkaz procedury PIO zapisanej w funkcji `smproc` wykonywany jest przez 1 tak zegara, tyle samo trwają dodatkowe opóźnienia. Stąd dokładnie wiadomo w jakim tempie wykonany zostanie kod funkcji `smproc`. Kolejne dwa argumenty oznaczają numer wyprowadzenia mikrokontrolera, od którego zaczynają się linie sterujące (`set_base`) oraz linie danych (`in_base`).

`sm.active(1)` aktywuje maszynę stanów PIO. To polecenie oraz poprzednie powinny być w kodzie programu wykonane jeden raz.

Polecenia `sm.put()` oraz `sm.get()` stanowią parę poleceń, które można już używać wielokrotnie jeśli potrzebne będzie odczytanie większej liczby próbek. `sm.get()` zwraca wartość przekazaną do FIFO wyjściowego przez maszynę stanów. Po wykonaniu całości powyższego kodu w zmiennej `code` znajduje się zatem słowo będące wynikiem przetwarzania. To, jakiemu napięciu odpowiada to słowo wyjaśnia rysunek 8b.

3.3. PRZETWORNIK ADC W RP2040

Mikrokontroler RP2040 zawiera w swojej strukturze 12-bitowy przetwornik ADC z pięcioma wejściami, z których ADC0 dołączone jest do napięcia V_{ADC} układu kondycjonowania napięć na makiecie a ADC4 do czujnika temperatury wewnątrz RP2040 [3].

Jego użycie w języku MicroPython jest bardzo proste. Ilustruje je poniższy kod:

```
import machine
from machine import ADC
#utworzenie obiektu klasy ADC obsługującej wejście przetwornika
adc = ADC(pin_ADC)
#odczyt słowa wyjściowego z ADC
code = adc.read_u16()
```

W kodzie tym napis `pin_ADC` zastąpić trzeba numerem wyprowadzenia modułu Raspberry Pi Pico, do którego dołączono napięcie V_{ADC}. Klasa ADC zawiera metodę `read_u16` zwracającą binarny wynik przetwarzania wyrównany do lewej strony, czyli z przedziału od 0 do 65535.

3.4. KONWERSJA TYPÓW DANYCH

Funkcje MicroPython pobierające wartości z magistral zwracają wynik w postaci ciągu bajtów czyli zmiennej typu `bytes` lub `bytearray`, której kolejne elementy zawierają kolejne bajty odebrane przez magistralę.

Zmiana typu na `int` wymaga konwersji. Można to osiągnąć realizując następujące polecenie:

```
import ustruct
code = ustruct.unpack(">H", data)[0]
```

Moduł `ustruct` zawiera szereg funkcji do konwersji danych binarnych. Funkcja `ustruct.unpack` konwertuje według wskazanego wzorca ciąg bajtów `data` na tablicę wartości. Zmiennej `code` przyporządkowujemy jedynie początkowy element zwracanej tablicy bo w przykładzie konwertowana jest jedna tylko wartość. Różne wzorce opisuje dokładnie dokumentacja języka Python [4]. W przykładzie znak '>' wzorca oznacza kolejność bajtów „big endian", a 'H' daną typu 16-bitowe słowo w kodzie binarnym. Znak 'h' oznaczałby daną typu 16-bitowe słowo w kodzie U2.

Nie wszystkie przetworniki zwracają słowo dzielące się równo na bajty. Przetwornik 12-bitowy daje wynik przetwarzania D 12-bitowy, złożony z bitów od d11 do d0. MicroPython odpowiedź zapisze w tablicy składającej się łącznie z 16 bitów, umieszczonych w dwóch kolejnych bajtach tabeli data:

data[0]								data[1]							
0	0	0	0	d11	d10	d9	d8	d7	d6	d5	d4	d3	d2	d1	d0

Najstarsze cztery bity są zerami, co nie powoduje problemów, gdy bity $d_{11} \dots d_0$ są słowem w kodzie dwójkowym. Gdy jest to kod U2 bit d_{11} jest bitem znaku o ujemnej wadze! Liczby z $d_{11} = 1$ powinny być ujemne a tak się nie stanie, co możesz sprawdzić realizując kod:

```
data = bytes ([0b00001111, 0b11111111])
code = ustruct.unpack(">h", data)[0]
print(code)
```

W tym przykładzie wszystkie bity $d_{11} \dots d_0$ są równe 1. Dwunastobitowe słowo binarne w kodzie U2, składające się z samych jedynek, to wartość -1. Program jednak da wynik 4095. Bierze się to stąd, że funkcja `ustruct.unpack` za bit znaku uważa najstarszy z całego ciągu dwóch bajtów, którym w naszym przykładzie stale jest 0.

Aby konwersja odbywała się prawidłowo należy naprawić to nieporozumienie. Wartości ujemne mają $d_{11} = 1$. Funkcja `ustruct.unpack` potraktuje ten bit jako bit o wadze 2048, co da wartość `code` równą 2048 lub więcej. Wynik można zatem skorygować zmieniając wagę tego bitu na -2048, czyli odejmując od `code` różnicę tych wag:

```
data = bytearray ([0b00001111, 0b11111111])
code = ustruct.unpack(">h", data)[0]
if code >= 2048:
    code = code - 4096
print(code)
```

Teraz uzyskasz prawidłowy wynik, wynoszący -1. Czy jesteś w stanie wytłumaczyć dlaczego tak się stało?

3.5. POBIERANIE CIĄGU PRÓBEK

Dotychczas poznaliście metody na pobranie z przetworników jednej próbki. Jak rozszerzyć możliwości programów MicroPython o pobieranie ciągu próbek z równomiernym próbkowaniem? Wykorzystać do tego można licznik czasu (ang. timer).

Licznik czasu po jego konfiguracji i aktywacji co zadany przedział czasu będzie wywoływał procedurę zdefiniowaną przez programistę, której delegację przekazano w czasie inicjalizacji licznika.

Obecnie MicroPython na RP2040 realizuje wyłącznie liczniki czasu programowo. Jest to ograniczenie wyłącznie interpretera, sam mikrokontroler zawiera liczniki czasu sprzętowe. Dają one bardziej precyzyjną kontrolę nad okresem próbkowania stąd wraz z ich przyszłą implementacją w MicroPythonie zmieni się nieco przebieg ćwiczenia.

Przykładowy kod programu do pobierania ciągu próbek z próbkowaniem równomiernym, używający programowego licznika czasu, wygląda następująco:

```
import time
import machine
from machine import Timer
# dodaj tu kolejne from machine import ... w razie potrzeby

# procedura odczytu słowa z przetwornika ADC
def adc_sample():
    # wpisz tu ciąg poleceń odczytu z wybranego ADC kończąc ją zwróceniem wyniku przetwarzania:
    return code

#liczba pobieranych próbek
SAMPLES = liczba_pobek
#licznik probek
n = int(0)
# czy przetwarzanie gotowe
done = int(0);

#wstępne zaalokowanie pamięci na tablicę adc_buf, w której znajdą się wyniki przetwarzania
adc_buf = [0] * SAMPLES

#procedura wywoływana za każdym “tyknięciem” zegara
def sample(t):
    #wskazanie zmiennych globalnych używanych w ciele funkcji
    global n, adc_buf, done
    #jezeli nie wypelniono calej tabeli adc_buf to uruchamiamy adc_sample
    if (n<SAMPLES):
        adc_buf[n] = adc_sample()
        n=n+1
    #w przeciwnym wypadku sygnalizujemy koniec zbierania probek
    else:
        done = 1

#timer – uruchomienie licznika czasu
tim = Timer(freq = sampling_frequency, mode = Timer.PERIODIC, callback = sample)
#czekamy az zostanie zasygnalizowane koniec przetwarzania z komunikatami o progresie probkowania
while done == 0:
    print("not done, n = " + str(n))
    time.sleep(1)
#komunikat o zakończeniu przetwarzania i wyłączenie licznika czasu
print("done")
tim.deinit();
```

Fragmenty zaznaczone na czerwono należy wypełnić innym kodem bądź podać wartości liczbowe określające parametry próbkowania. Ze względu na ograniczenia precyzji działania programowych liczników czasu maksymalna częstotliwość próbkowania (`sampling_frequency`) nie powinna przekraczać 5000 Hz. Liczbę próbek należy dostosować do potrzeb.

Po wykonaniu tego kodu w tablicy `adc_buf` znajdzie się ciąg wartości próbek z przetwornika ADC.

3.6. WYMIANA INFORMACJI Z PAKIETEM OCTAVE

Co zrobić ciągiem danych zawierającymi próbki z przetwornika ADC? Warto je zwizualizować, przeliczyć bądź przetworzyć. Można zrobić to za pomocą Octave, który znasz z innych kursów. Przeniesienie danych pochodzących z wykonania programu w MicroPython może być zrealizowane w prosty sposób.

Jeśli wyniki przetwarzania w programie realizowanym przez Raspberry Pi Pico znajdują się w tablicy `adc_buf` zawierającej zmienne typu `int`, to za pomocą polecenia:

```
print ("x=" + str(adc_buf) + ";")
```

wyprowadzi się do konsoli ciąg znaków będący konkatenacją napisu „x=”, zawartości tablicy adc_buf skonwertowanej na ciąg znaków oraz znaku „;” na końcu. Przykładowo, poniższy kod:

```
adc_buf = []
adc_buf = [1, 5, 8, 13, 78]
print ("x="+str(adc_buf)+";")
```

spowoduje to wypisanie w konsoli napisu:

```
x=[1, 5, 8, 13, 78 ];
```

Tekst ten to gotowa linijka skryptu Octave! Tekst można zaznaczyć i skopiować w konsoli programu Thonny po czym wkleić w skrypt programu Octave. Czy pamiętasz, dlaczego na końcu tego polecenia jest średnik?

Dalsze polecenia skryptu Octave można tworzyć dowolne. Przykładowo, przebieg czasowy zawarty w wektorze danych x można wykreślić za pomocą poleceń:

```
N = length(x);
fs = ...; # wpisz tu czestotliwosc próbkowania w Hz, z jaka zbierano probki
plot((0:N-1)/fs, x)
```

Czy znasz działanie użytych tu instrukcji Octave? Czy pamiętasz składnię zakresów? Potrafisz wyjaśnić, czemu w poleceniu plot użyto takiego właśnie wyrażenia, jako pierwszy jego argument?

4. PRZEBIEG ĆWICZENIA

W trakcie realizacji ćwiczenia realizowane będą wskazane przez prowadzącego punkty z poniższej listy.

4.1. PRZYGOTOWANIE MAKIETY DO PRACY

Makieta wymaga podania trzech napięć zasilających. Dwa z nich tworzą parę napięć symetrycznych zapewniających zasilanie $\pm 6\ V$. Regulatory napięcia na makiecie tworzą z nich napięcia zasilające część makiety wymagających napięć $\pm 2{,}5\ V$. Trzecie napięcie równe $7\ V$ zasila regulator napięcia do układów wymagających $3{,}3\ V$.

Wszystkie wyjścia zasilacza laboratoryjnego ustaw wstępnie na właściwe napięcia. Zastosuj ograniczenia prądowe około $100\ mA$. Podłącz zasilacz laboratoryjny do wejść makiety. Skontroluj prawidłowość połączeń. Włącz zasilacz. Świecące się trzy diody LED zasygnalizują prawidłowe zasilenie makiety.

4.2. UKŁADY KONDYCJONOWANIA NAPIĘCIA

Prowadzący zajęcia wskaże trzy wartości napięć z zakresu od $-1\ V$ do $+1\ V$, jakie mają zostać przetworzone przez przetworniki A/C. Napięcia te będą podawane na wejście układu kondycjonującego V_{IN}. Oblicz, jakie wartości powinny mieć te napięcia po przejściu przez układ kondycjonujący. Sprawdź następnie, czy na wyjściach V_{ASYM}, V_{SYM+} i V_{SYM-} pojawiają się napięcia o właściwych wartościach.

4.3. PRZETWARZANIE NAPIĘĆ STAŁYCH

Przetwarzanie odbywać się będzie wszystkimi dostępnymi przetwornikami. Zidentyfikuj i zapisz w protokole, które wyprowadzenia mikrokontrolera (GPxx) oraz które peryferia mikrokontrolera są odpowiedzialne za sterowanie liniami magistral przetworników zewnętrznych i gdzie dołączono sygnał analogowy do przetwornika wbudowanego w RP2040 (Rysunek 1). Odnajdź tam również informacje o wartościach napięć odniesienia przetworników.

Na podstawie informacji z rozdziału 3 przygotuj trzy skrypty w języku MicroPython. Jeden do obsługi przetwornika z magistralą SPI, drugi z magistralą równoległą, trzeci do przetwornika wbudowanego w RP2040. Wszystkie skrypty mają zapewnić wykonanie pojedynczego odczytu.

Odczytaj wartości słów binarnych zwracanych przez przetworniki dla wszystkich wartości napięć stałych wskazanych przez prowadzącego w poprzednim punkcie. Znając wartości napięć odniesienia oraz sposób działania przetworników przelicz słowo bitowe na wartości napięcia. Czy wyniki zgadzają się z oczekiwaniami? Skomentuj uzyskane wyniki.

4.4. PRZETWARZANIE NAPIĘĆ PRZEMIENNYCH

Podłącz wyjście generatora funkcyjnego do wejścia V_{IN} makiety i oscyloskopu używając trójnika BNC. Wygeneruj przebieg okresowy o zadanych przez prowadzącego parametrach (amplituda, składowa stała, kształt, częstotliwość). Wartości minimalne i maksymalne przebiegu powinny mieścić się w przedziale od -1 V do +1 V a częstotliwość w zakresie od 5 Hz do 50 Hz. Przebieg ten przetwórz na postać cyfrową za pomocą wskazanego przetwornika. Dobierz częstotliwość próbkowania tak, aby spełnione było kryterium Shannona i jednocześnie aby kilka okresów sygnału zarejestrowane było w kilkuset próbkach. Częstotliwość próbkowania nie może przekraczać 5 kHz.

Wartości zwrócone przez przetwornik zapisz w wektorze skryptu Octave. Przelicz je na wartości napięć. Oblicz wartości czasów odpowiadających poszczególnym próbkom i wykreśl przebieg sygnału zarejestrowanego przez przetwornik w dziedzinie czasu. Porównaj go z oscylogramem.

4.5. WIDMO SYGNAŁU DYSKRETNEGO

Przyjmij częstotliwość próbkowania z zakresu 100 $Hz - 2\ kHz$, podaną przez prowadzącego. Używając generatora podaj na wejście V_{IN} sygnał sinusoidalny bez składowej stałej, o amplitudzie 1 V_{rms} o częstotliwości spełniającej kryterium Shannona. Pobierz zadaną przez prowadzącego liczbę próbek z zakresu 100 – 1000 stosując wskazany przez prowadzącego przetwornik. Używając pakietu Octave zaprezentuj widmo amplitudowe spróbkowanego sygnału na dwóch wykresach, używając liniowej oraz logarytmicznej skali amplitud [5]. Nałóż na sygnał jedno z okien [6] i wykreśl widmo ponownie na skali logarytmicznej.

Zmień częstotliwość sygnału aby przekraczała ona kryterium Shannona. Oblicz, przy jakiej częstotliwości w widmie oczekujesz pojawienia się prążka odpowiadającego sygnałowi. Pobierz nowy zestaw próbek i powtórz czynności z poprzedniego punktu. Skonfrontuj oczekiwania z wynikami obliczeń.

Powtórz czynności dla sygnału o szerszym paśmie. Zastosuj częstotliwość próbkowania 200 Hz lub 2 kHz. Częstotliwość podstawowa sygnału powinna stanowić mniej niż 5% częstotliwości próbkowania. Prowadzący wskaże jedno ze źródeł sygnału a może to być m. in.:

- sygnał sinusoidalny zniekształcony układami D1 lub D2,
- sygnał prostokątny bądź piłokształtny z generatora.

Sygnał zaobserwuj na oscyloskopie. Pobierz próbki i wykreśl widmo amplitudowe sygnału z zastosowanym okienkowaniem. W jakim przedziale częstotliwości mieści się sygnał?

Następnie sygnał przed podaniem na wejście V_{IN} przefiltruj przez odpowiedni filtr antyaliasingowy. Porównaj widma sygnału przed i po zastosowaniu filtra.

4.6. OCENA EFEKTYWNEJ ROZDZIELCZOŚCI PRZETWORNIKÓW A/C

Przyjmij częstotliwość próbkowania z zakresu $100\ Hz - 2\ kHz$, podaną przez prowadzącego. Używając generatora podaj na wejście V_{IN} sygnał sinusoidalny o amplitudzie $1\ V_{rms}$ o częstotliwości równej 1% częstotliwości próbkowania. Pobierz 1000 próbek stosując wskazany przez prowadzącego przetwornik. Używając pakietu Octave zaprezentuj widmo amplitudowe spróbkowanego sygnału.

Oblicz gęstość widmową mocy sygnału i wykreśl jego widmo. Oblicz moc sygnału bazując na jego gęstości widmowej całkując ją numerycznie [6].

Ręcznie wyzeruj w widmie gęstości mocy składowe zawierające sygnał. Pozostawisz w ten sposób gęstość widmową szumów i zniekształceń. Oblicz moc szumów bazując na jego gęstości widmowej całkując ja numerycznie.

Oblicz całkowity stosunek sygnał do szumu SINAD. Mając SINAD oblicz efektywną liczbę bitów przetwornika ENOB. Porównaj ją ze sprzętową rozdzielczością przetwornika.

5. BIBLIOGRAFIA

[1] „AD7450 Datasheet,” [Online]. Available: https://www.analog.com/en/products/ad7450.html.
[2] „AD7819 Datasheet,” [Online]. Available: https://www.analog.com/en/products/ad7819.html.
[3] „RP2040 Datasheet,” [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf.
[4] „Python standard library - struct,” [Online]. Available: https://docs.python.org/3/library/struct.html.
[5] T. Piasecki, „Przetwarzanie Sygnałów, instrukcja do ćwiczenia 3,” Katedra Nanometrologii.
[6] T. Piasecki, „Przetwarzanie Sygnałów, instrukcja do ćwiczenia 4,” Katedra Nanometrologii.

PROTOKÓŁ

CZŁONKOWIE GRUPY	
1.	
2.	
3.	
4.	
5.	
6.	

NR GRUPY	NUMER ĆWICZENIA	DATA ZAJĘĆ	PODPIS PROWADZĄCEGO