

Graficzne programowanie mikrokontrolerów

M5Stack / UIFlow

Laboratorium 3

Pomiary I2C i sterowanie napędami

Czas trwania: 3 × 45 min**Tryb:** studia niestacjonarne**Jednostka:** Katedra Elektrotechniki i Technologii Nadprzewodnikowych**Prowadzący:** dr inż. Michał Łanczont

Cel ćwiczenia

Celem ćwiczenia jest budowa toru pomiarowo-sterującego wykorzystującego I2C i napędy. Student odczytuje czujniki środowiskowe przez gotowe sterowniki oraz samodzielnie konfiguruje ADS1115 i odczytuje jego rejestry. Porównuje pomiar joysticka przez ADC ESP32 i ADS1115, przelicza wyniki na napięcie i wspólną skalę sterowania. Poznaje sterowanie serwowym, silnikiem DC oraz silnikiem krokowym, stosując ograniczenia, diagnostykę i jednoznaczny stan bezpieczny.

1. Wprowadzenie

Wyposażenie: Core + Demo Board, SHT30, ADS1115, joystick, serwo, silnik DC oraz silnik krokowy. Projekt odczytu joysticka z ćwiczenia 2.

Organizacja pracy

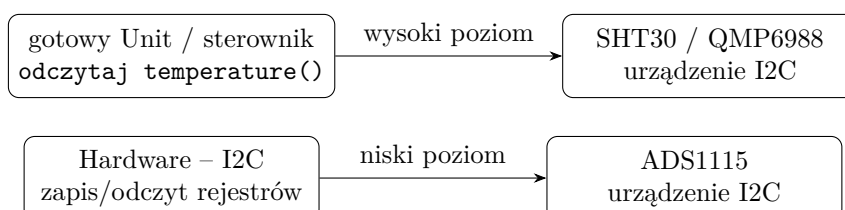
Zapoznaj się z opisem stosowanych elementów, wykonaj cztery zadania cząstkowe, a następnie połącz przygotowane funkcje w jednym zadaniu końcowym. Polecenia, szczegółowe opisy, algorytmy i testy znajdują się bezpośrednio przy zadaniach. Zachowuj działające wersje projektu i wykorzystuj je w kolejnych etapach.

Obowiązuje zakres określony w poleceniach zadań cząstkowych i zadania końcowego. Opisane możliwości rozbudowy pozwalają pogłębić materiał. Wyniki testów i zdjęcia stanowiska zbierz podczas zajęć, a sprawozdanie opracuj po zajęciach.

Tor pomiarowy I2C

Urządzenie I2C można obsługiwać na różnym poziomie. Gotowy Unit lub sterownik czujnika udostępnia funkcje wysokiego poziomu, np. odczyt temperatury albo ciśnienia. Programista nie musi wtedy znać kolejności bajtów ani adresów rejestrów wewnętrznych. Gotowy sterownik wykorzystaj do pomiarów środowiskowych.

Do obsługi ADS1115 zostanie użyty ogólny interfejs **Hardware – I2C**. Student będzie samodzielnie wykonywał zapis i odczyt rejestrów ADS1115. Pozwala to zobaczyć, co w praktyce wykonuje gotowy sterownik urządzenia.



Magistrala I2C

I2C wykorzystuje dwie linie: SDA do przesyłania danych i SCL do sygnału zegarowego. Wiele urządzeń może współdzielić te same przewody, jeżeli posiadają różne adresy. Kontroler rozpoczyna transmisję, podaje adres urządzenia i określa kierunek transmisji.

Dla płytki **M5Stack DEMO BOARD** istotne są m.in. następujące adresy:

Układ	Adres I2C	Funkcja
SHT30	0x44	temperatura i wilgotność
QMP6988	0x76	ciśnienie atmosferyczne
ADS1115	0x48	4-kanalowy przetwornik ADC

ENV-III a czujniki na DEMO BOARD

Zewnętrzny Unit ENV-III również zawiera SHT30 i QMP6988, ale jego QMP6988 pracuje typowo pod adresem 0x70. Nie należy więc bez sprawdzenia traktować konfiguracji ENV-III jako identycznej z czujnikami wbudowanymi w DEMO BOARD. Do odczytu czujników środowiskowych użyj bloku/sterownika zgodnego z rzeczywistym wyposażeniem stanowiska i zweryfikuj adresy przez skanowanie I2C.

ADS1115 – zewnętrzny przetwornik analogowo-cyfrowy

ADS1115 jest 16-bitowym przetwornikiem ADC z czterema wejściami analogowymi AIN0–AIN3. Mikrokontroler nie odczytuje tych wejść bezpośrednio. Wysyła przez I2C konfigurację pomiaru, czeka na zakończenie konwersji, a następnie pobiera wynik z rejestru konwersji.

W ćwiczeniu wykorzystywane są dwa rejestry:

Adres	Nazwa	Znaczenie
0x00	Conversion	wynik ostatniej konwersji, 16 bitów
0x01	Config	konfiguracja kanału, zakresu i trybu pracy

Dla pomiaru pojedynczego wejścia względem masy można wykorzystać następujące gotowe słowa konfiguracji:

Kanał	Konfiguracja	Znaczenie
AIN0	0xC383	pojedyncza konwersja, $\pm 4,096$ V, 128 SPS
AIN1	0xD383	pojedyncza konwersja, $\pm 4,096$ V, 128 SPS
AIN2	0xE383	pojedyncza konwersja, $\pm 4,096$ V, 128 SPS
AIN3	0xF383	pojedyncza konwersja, $\pm 4,096$ V, 128 SPS

Przy zakresie $\pm 4,096$ V jeden krok wyniku odpowiada około $125 \mu\text{V}$. Po zapisaniu konfiguracji przy częstotliwości 128 SPS wystarczy w prostym ćwiczeniu odczekać około 10 ms przed odczytem wyniku.

Konflikt ADC i DAC na DEMO BOARD

Na DEMO BOARD układ ADS1115 oraz DAC6574 korzystają z adresu 0x48. Producent zaznacza, że funkcje ADC i DAC tej płyty nie powinny być używane jednocześnie. W czasie tego ćwiczenia sekcja DAC ma pozostać nieużywana.

programowanie obsługi interfejsu **I2C** należy rozpocząć od dodania go do zakładki **HARDWARE**. Interfejs definiowany jest tylko raz, w bloku Setup. Dla mikrokontrolera *M5Stack Core* jest ona dostępna na portach **GPIO21 (SDA)** i **GPIO22 (SCL)**.

Należy zwrócić uwagę na specyfikę typu niektórych wartości wykorzystywanych przy realizacji zapytań i otrzymywanych odpowiedzi. Są to wartości jedno- i dwu-bajtowe. Wartości jedno-bajtowe (np.: **0x01**) można zapisywać bezpośrednio w blokach zapytań. Natomiast do w przypadku wartości dwu-bajtowych należy utworzyć odpowiednio sformatowane zmienne:

- z grupy **Bytes**, *create bytes with* - jeżeli znana jest wartość (np.: **0xc383**) należy zdefiniować dwu-bajtowy element i pierwszemu przypisać **0xc3** a drugiemu **0x83**
- z grupy **Bytearray**, *create empty bytearray with length* - jeżeli nie jest znana wartość i będzie odczytywana, należy podać liczbę odczytywanych bajtów (w przypadku odczytu z układu ADS1115 jest to liczba **2**).

Procedura odczytu przez interfejs I2C składa się zazwyczaj z czterech etapów:

- Wysłanie zapytania, zapisania określonej wartości (np.: **0xc383**) pod wskazanym adresem (np.: **0x01**) na uprzedzeniu o określonym id (np.: **0x48**). W UiFlow 2.0 jest to blok **write buf**.
- odczekanie określonego czasu aż czujnik lub urządzenie wykona zadanie
- odczytanie wartości z określonego adresu (np.: **0x00**) na urządzeniu o podanym id (np.: **0x48**), w przypadku UiFlow 2.x będzie to blok **read from addr**.
- przetworzenie odczytanej wartości zgodnie z algorytmem zapisanym w dokumentacji czujnika, elementu, itp. W przypadku przetwornika ADS1115 będzie to połączenie dwój jedno-bajtowych wartości w wielkość dwu-bajtową, a następnie przeliczenie na wartość napięcia zgodnie z procedurą opisaną w dokumentacji.

Na płycie **DEMO BOARD** dla każdego kanału przetwornika ADC są dostępne dwa złącza. Przewód łączący punkt odczytu (np.: port kanału X JoyStick'a) należy podpiąć do **prawego** złącza wejściowego przetwornika.

Elementy wykonawcze i sterowanie ruchem

Serwomechanizm jest sterowany wartością odpowiadającą żadanemu położeniu. Silnik DC wymaga określenia kierunku i poziomu sterowania prędkością. Silnik krokowy wykonuje ruch jako sekwencję kroków, dzięki czemu możliwe jest pozycjonowanie względne.

Nie należy utożsamiać tych trzech metod sterowania. Interfejs operatora może być podobny, ale algorytm sterowania napędem jest inny.

Stan bezpieczny

Każdy program sterujący napędem musi posiadać jednoznaczny stan ZATRZYMANIE. Po uruchomieniu programu napęd nie powinien rozpocząć ruchu bez polecenia użytkownika. Utrata sterowania lub błąd powinny prowadzić do zatrzymania, o ile dany układ wykonawczy na to pozwala.

Bezpieczeństwo

Nie dotykaj ruchomych elementów podczas pracy. Przed zmianą przewodów wyłącz zasilanie odpowiedniego modułu. Nie blokuj mechanicznie wału silnika i nie wymuszaj ruchu serwa poza jego dopuszczalnym zakresem. **Nie wymuszaj ręcznie ruchu serwomechanizmu!!!**

Wskazówki do wariantu realizowanego na zajęciach

Na zajęciach wykonaj jeden kanał ADS1115 oraz krótkie próby trzech rodzajów napędu. Pełne opisy dwóch kanałów ADS1115, QMP6988, porównania torów ADC, sekwencji pełnokrokowych i półkrokowych, ramp oraz wspólnego panelu trzech napędów pozostają dostępne w tej instrukcji. W zakresie obowiązkowym zrealizuj warianty sprzętowe wskazane w poleceniach.

Magistrala i przetworniki

I2C używa linii SDA i SCL. Urządzenia współdzielą magistralę, lecz wymagają właściwych adresów i sterowników. Dla Core przyjęto $SDA=GPIO21$ i $SCL=GPIO22$. Interfejs inicjalizuj tylko raz. Skan potwierdza odpowiedź adresu, ale sam nie identyfikuje typu układu.

SHT30 mierzy temperaturę i wilgotność. ADS1115 zamienia napięcie analogowe na wynik cyfrowy odczytywany przez I2C. Joystick można dzięki temu mierzyć zarówno wewnętrznym ADC ESP32, jak i zewnętrznym ADS1115.

Przygotowanie przed zajęciami

Prowadzący sprawdza sterowniki czujnika, konfigurację wyjść serwa/DC i mapę faz silnika krokowego dla rewizji płyty. Udostępnia projekt startowy z inicjalizacją sprzętu i funkcjami `ustaw_serwo(kat)`, `ustaw_dc(kierunek,pwm)`, `ustaw_fazy(A,B,C,D)`. Funkcja DC przyjmuje wystawienie 0–100% i przelicza je na zakres sterownika. Student buduje logikę, ograniczenia i testy. Napędy są testowane kolejno, bez obciążenia.

Układ	Adres / sposób obsługi
SHT30	0x44; gotowy sterownik temperatury i wilgotności.
QMP6988	0x76 na opisywanej Demo Board; opcjonalny pomiar ciśnienia.
ADS1115	0x48; Config=0x01, Conversion=0x00.

Adresy potwierdź skanem. QMP6988 w zewnętrznym ENV-III może mieć inny adres (typowo 0x70). Nie traktuj tych konfiguracji jako zamiennych. Konflikt ADS1115/DAC pod 0x48 rozwiąż zgodnie ze schematem stanowiska; sekcja DAC pozostaje nieużywana.

Pomiar ADS1115

Połącz ADC_X joysticka z AIN0 sekcji ADC (na Demo Board prawe złącze wejściowe). Zapisz do Config bajty 0xC3, 0x83: AIN0 względem GND, zakres $\pm 4,096$ V, pojedyncza konwersja 128 SPS. Po około 10 ms odczytaj dwa bajty Conversion, starszy jako pierwszy.

```
surowy = bajt0*256 + bajt1
jesli surowy >= 32768: surowy = surowy - 65536
napiecie_V = surowy * 0.000125
```

Zakres przetwornika nie oznacza dopuszczenia takiego napięcia na wejściu. W tym ćwiczeniu stosuj wyłącznie źródło 0–3,3 V i połączenia zatwierdzone dla stanowiska.

Trzy rodzaje napędu

Servo otrzymuje zadaną pozycję, DC — kierunek i wystawienie PWM, a silnik krokowy — sekwencję faz. Bez sprzężenia zwrotnego nie mierzysz rzeczywistej prędkości DC ani położenia wału. Czterofazowy silnik Demo Board jest sterowany przez układ pośredniczący; nie podłączaj uzwojeń do GPIO.

Przykładowa sekwencja jednofazowa krokowca: 1000, 0100, 0010, 0001. Każdy zapis oznacza cztery wartości logiczne A–D, nie liczbę dziesiętną. Zmiana kierunku odwraca kolejność. Licznik kroków daje pozycję programową, a nie fizyczne bazowanie.

Bezpieczeństwo

Przewody zmieniaj bez zasilania. Nie dotykaj ruchomych elementów, nie blokuj wału i nie obracaj serwa ręką. Zakres serwa, limit PWM i czas zatrzymania przed zmianą kierunku ustala prowadzący dla stanowiska.

Odczyt opisu pinów i rejestrów

W opisie I2C oznaczenie **SDL** należy odczytywać jako **SCL**. Adres układu (np. **0x48**) i adres jego rejestru (np. **0x01**) to różne parametry transakcji. Zachowaj tę różnicę przy wyborze bloków.

2. Zadania cząstkowe

Wykonuj zadania w podanej kolejności, rozwijając ten sam projekt. Każde zadanie zawiera polecenie, opis wykonania, przykłady i sposób sprawdzenia wyniku. Przykłady pokazują również dodatkowe warianty działania; zakres wymagany określa polecenie na początku zadania. Zapisuj działającą wersję po każdym etapie.

Zadanie 1. Czujnik i skan I2C

Polecenie zadania

Uruchom skan I2C i przypisz wykryte adresy do wyposażenia. Gotowym sterownikiem odczytuj temperaturę i wilgotność SHT30 co 1 s. Wyświetl jednostki °C i %RH. Dla błędu odczytu pokaż BRAK DANYCH zamiast zera.

Warunek odbioru: Lista adresów jest zapisana, wskazania mają jednostki, a student wyjaśnia rolę gotowego sterownika.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

Odczyt czujników środowiskowych z użyciem gotowych bloków

Dodaj w UIFlow odpowiedni Unit lub sterownik dla czujników środowiskowych dostępnych na stanowisku. Odczytaj temperaturę, wilgotność i ciśnienie bez samodzielnego wykonywania operacji na rejestrach I2C. Wyniki wyświetl na ekranie wraz z jednostkami.

Sposób realizacji i weryfikacja

1. Dobierz sterownik zgodny z faktycznym czujnikiem i rewizją stanowiska.
2. Temperaturę wyświetlaj w °C, wilgotność w %RH, a ciśnienie w hPa.
3. Ustal okres odczytu około 1 s. Nie odczytuj czujników w każdej iteracji pętli.
4. Jeżeli odczyt jest niepoprawny, pokaż komunikat BRAK DANYCH; nie zastępuj błędu wartością 0.

Sprawdzenie: student potrafi wskazać, że gotowy Unit ukrywa szczegóły transakcji I2C i udostępnia bezpośrednio wielkości fizyczne.

Skanowanie magistrali i identyfikacja adresów

Uruchom skanowanie I2C. Zapisz wykryte adresy i przypisz je do urządzeń na podstawie dokumentacji stanowiska. Porównaj wynik z adresami czujników oraz ADS1115.

```
adresy = skanuj_i2c()  
dla adres in adresy:  
    wyswietl(adres)
```

Weryfikacja

Przygotuj tabelę: adres -> układ -> funkcja. Zwróć uwagę, że sam adres nie mówi, jak interpretować dane; do poprawnej komunikacji potrzebna jest również mapa rejestrów konkretnego układu.

Zadanie 2. ADS1115 przez rejestry**Polecenie zadania**

Przy wyłączonym zasilaniu odłącz oś X od GPIO35 i podłącz ją do AIN0. Zrealizuj sekwencję zapisu konfiguracji i odczytu opisaną we wstępie. Wyświetl surowy wynik i napięcie. Zmierz minimum, środek i maksimum dla tego toru; nie kopiuj liczb kalibracyjnych z wewnętrznego ADC.

Warunek odbioru: Dla trzech położen napięcie zmienia się zgodnie z wychyleniem. Student wskazuje oba rejestry i kolejność bajtów.

Ograniczenie zakresu ćwiczenia

Obowiązkowy jest jeden kanał ADS1115 i jeden czujnik środowiskowy. Odczyt AIN1, porównanie wielu czujników i zmiana zakresów ADC są dodatkowymi rozwinięciami.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

Pierwsza konwersja ADS1115 – kanał AIN0

Podłącz źródło napięcia z zakresu 0–3,3 V do wejścia AIN0 sekcji ADC płytki DEMO BOARD. Może to być jedna z osi joysticka używanego w ćwiczeniu 2. Wykonaj pojedynczą konwersję ADS1115 poprzez bezpośredni zapis i odczyt rejestrów I2C.

Algorytm:

zapis 0xC383 do rejestru 0x01 → czekaj 10 ms → odczytaj 0x00 → złóż wynik
16-bitowy

```
adres_ads = 0x48
konfiguracja = 0xC383

zapis16_i2c(adres_ads, 0x01, konfiguracja, BIG_ENDIAN)
czekaj_ms(10)
wynik = odczytaj16_i2c(adres_ads, 0x00, BIG_ENDIAN)

if wynik >= 32768:
    wynik = wynik - 65536

napięcie = wynik * 0.000125
```

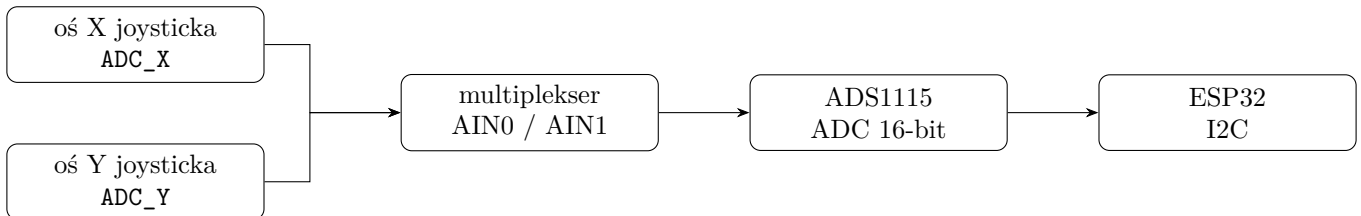
Sposób realizacji i weryfikacja

1. Pokaż surowy wynik oraz napięcie w woltach.
2. Zmień napięcie wejściowe i sprawdź, czy wynik zmienia się monotonicznie.
3. Wyjaśnij, dlaczego do urządzenia zapisano najpierw rejestr **Config**, a wynik odczytano z rejestru **Conversion**.

Odczyt dwóch osi joysticka przez ADS1115

W poprzednim przykładzie wykonano odczyt pojedynczego kanału ADS1115. W tym etapie przetwor-
nik zostanie wykorzystany do pomiaru obu osi joysticka. Sygnał ADC_X należy połączyć z wejściem
AIN0, a sygnał ADC_Y z wejściem AIN1.

ADS1115 ma jeden tor konwersji poprzedzony multiplekserem wejściowym. Oznacza to, że kanały
nie są mierzone równocześnie. Program kolejno wybiera kanał AIN0, wykonuje konwersję i odczytuje
wynik osi X, a następnie wybiera kanał AIN1 i wykonuje drugi pomiar dla osi Y.



Wybór kanału przez rejestr Config

Zmiana mierzonego wejścia odbywa się przez pole MUX rejestru Config pod adresem 0x01. Dla
ustawień zastosowanych w ćwiczeniu wykorzystane zostaną dwa słowa konfiguracyjne:

Oś	Wejście	MUX	Konfiguracja
X	AIN0 względem GND	100	0xC383
Y	AIN1 względem GND	101	0xD383

Wartości są przesyłane do ADS1115 jako dwa bajty, od starszego do młodsze:

```
konfiguracja_x = b'\xc3\x83' # 0xC383 - AIN0
konfiguracja_y = b'\xd3\x83' # 0xD383 - AIN1
```

Dla AIN0 konfiguracja 0xC383 wybiera pomiar wejścia AIN0 względem masy, zakres $\pm 4,096$ V, tryb
pojedynczej konwersji oraz 128 próbek/s. Konfiguracja 0xD383 zachowuje te same parametry, ale
przełącza multiplekser na AIN1.

Sekwencja pomiaru pojedynczego kanału

Dla każdego kanału należy wykonać trzy operacje:

1. zapisać dwa bajty konfiguracji do rejestru 0x01,
2. odczekać około 10 ms na zakończenie pojedynczej konwersji,
3. odczytać dwa bajty wyniku z rejestru 0x00.

W kodzie wygenerowanym przez UIFlow zapis konfiguracji AIN0 ma postać:

```
i2c0.writeto_mem(0x48, 0x01, konfiguracja_x)
time.sleep_ms(10)
i2c0.readfrom_mem_into(0x48, 0x00, wynik_x)
```

Znaczenie parametrów:

0x48	adres układu ADS1115 na magistrali I2C,
0x01	adres rejestru Config, do którego zapisywana jest konfiguracja,
konfiguracja_x	dwa bajty 0xC3 0x83, wybierające AIN0,
0x00	adres rejestru Conversion zawierającego wynik ostatniej konwersji,
wynik_x	dwuelementowa tablica bajtów, do której zapisany zostaje wynik.

ADS1115 zwraca wynik jako dwa bajty. Starszy bajt jest odbierany jako pierwszy. Złożenie ich w jedną wartość 16-bitową wykonuje operacja:

```
wartosc_x = wynik_x[0] << 8 | wynik_x[1]
```

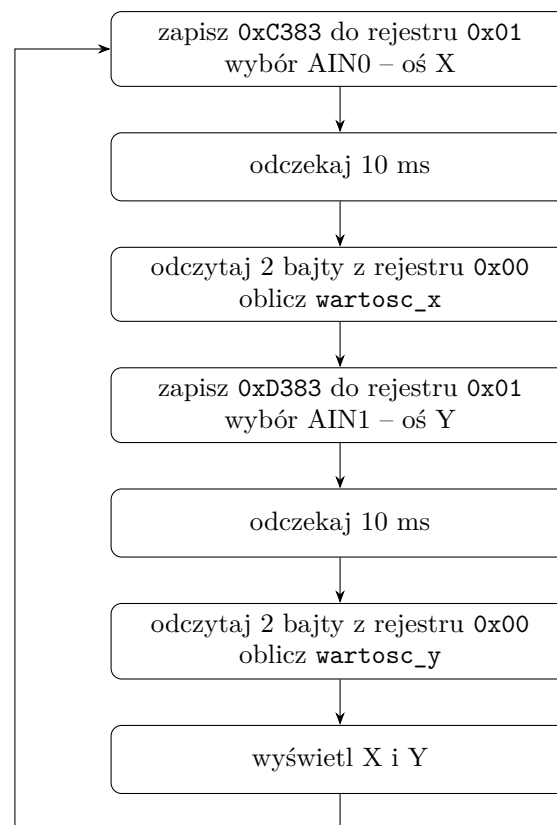
co odpowiada zależności:

$$\text{wartosc_x} = \text{wynik_x}[0] \cdot 256 + \text{wynik_x}[1].$$

Dla joysticka mierzonego względem GND oczekiwane wartości są dodatnie. W przypadku użycia ADS1115 do pomiarów różnicowych należy pamiętać, że rejestr Conversion przechowuje 16-bitową liczbę ze znakiem.

Odczyt obu osi

Pełny cykl pomiaru joysticka wygląda następująco:



Przeliczenie wyniku na napięcie

Dla zakresu $\pm 4,096$ V wartość jednego kroku ADS1115 wynosi:

$$q = \frac{4,096 \text{ V}}{32768} = 0,000125 \text{ V} = 125 \mu\text{V}.$$

Napięcia obu osi można więc obliczyć jako:

$$U_X = \text{wartosc_x} \cdot 0,000125, \quad U_Y = \text{wartosc_y} \cdot 0,000125.$$

W programie:

```
napiecie_x = wartosc_x * 0.000125
napiecie_y = wartosc_y * 0.000125
```

W środkowym położeniu joysticka napięcia obu osi powinny być zbliżone do połowy napięcia zasilania. Nie należy jednak przyjmować wartości idealnej. Położenie środkowe, minima i maksima należy wyznaczyć eksperymentalnie.

Sposób realizacji i weryfikacja

1. Uruchom najpierw pomiar wyłącznie AIN0 i sprawdź reakcję na ruch joysticka w osi X.
2. Dodaj konfigurację 0xD383 i odczyt AIN1 dla osi Y.
3. Wyświetl równocześnie wartości surowe X i Y.
4. Przelicz obie wartości na napięcie.
5. Porusz joystickiem kolejno w lewo, w prawo, w górę i w dół. Sprawdź, która wartość zmienia się dla każdej osi.
6. Zanotuj wartości minimum, środek i maksimum dla X i Y.
7. Wyjaśnij, dlaczego przed odczytem każdej osi trzeba ponownie zapisać rejestr **Config**.

Sprawdzenie: program naprzemiennie konfiguruje AIN0 i AIN1, wykonuje dwie niezależne konwersje i poprawnie prezentuje wartości obu osi joysticka.

Porównanie z wewnętrznym ADC ESP32

Porównaj odczyt tego samego joysticka wykonany dwiema metodami:

ADC ESP32	ADS1115
joystick → GPIO35/36 → ADC ESP32	joystick → AIN0/AIN1 → ADS1115 → I2C → ESP32
wewnętrzny ADC mikrokontrolera bez magistrali komunikacyjnej	zewnętrzny 16-bitowy ADC konfiguracja i odczyt przez rejestry I2C

Dla położenia środkowego i czterech skrajnych wychyleń zapisz wyniki obu metod. Porównaj stabilność wskazań, liczbę poziomów pomiarowych oraz złożoność programu. Nie oczekuj identycznych liczb surowych – oba przetworniki mają inną skalę i charakterystykę.

Normalizacja odczytu ADS1115

Wykorzystaj wartości minimum, środek i maksimum uzyskane w zadaniu 2 i przelicz obie osie do zakresu $-100 \dots 100$, tak jak w ćwiczeniu 2. Dzięki temu dalsza część programu może działać niezależnie od tego, czy źródłem pomiaru jest ADC mikrokontrolera, czy ADS1115.

```
if surowa >= srodek:
    wartosc = mapuj(surowa, srodek, maksimum, 0, 100)
else:
    wartosc = mapuj(surowa, minimum, srodek, -100, 0)
```

Rezultat przykładu

Program korzysta z gotowego sterownika do odczytu czujników środowiskowych, potrafi zeskanować magistralę I2C, a następnie samodzielnie konfiguruje ADS1115, odczytuje dwa kanały analogowe i przelicza wartości joysticka do wspólnej skali $-100 \dots 100$.

Tor sterowania napędem

polecenie operatora → ograniczenie i walidacja → sterownik napędu → napęd → stan/diagnostyka

Zadanie 3. Serwo i DC**Polecenie zadania**

Korzystając z przygotowanych funkcji sprzętowych, zadaj serwu trzy pozycje: minimum, środek i maksimum dopuszczalnego zakresu. Następnie zatrzymaj aktualizację serwa i przetestuj DC: ZATRZYMANIE, LEWO, PRAWO przy małym wysterowaniu. Zmiana kierunku wymaga PWM=0 i ustalonej zwłoki przed nowym kierunkiem. C zawsze zatrzymuje DC.

Warunek odbioru: Serwo nie przekracza zakresu, DC zmienia kierunek przez zatrzymanie, a C przerywa pracę również podczas zwłoki.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

Serwomechanizm

Steruj pozycją serwa wartością 0–100, przeliczaną na bezpieczny zakres kątowy ustalony przez prowadzącego. Użyj enkodera do zmiany nastawy. Dodaj ograniczenie programowe zapobiegające przekroczeniu zakresu.

Sposób realizacji i weryfikacja

1. Ustal dozwolony zakres mechaniczny przed rozpoczęciem ciągłego sterowania; nie testuj skrajnych kątów bez kontroli stanowiska.
2. Przelicz 0–100 na zakres kątowy i zawsze zastosuj ograniczenie przed wysłaniem polecenia do serwa.
3. Zmieniaj pozycję małymi krokami enkodera i obserwuj, czy serwo nie dociska mechanizmu do ogranicznika.

Pseudokod / zapis algorytmu:

```
kat = mapuj(nastawa, 0, 100, kat_min, kat_maks)
kat = ogranicz(kat, kat_min, kat_maks)
ustaw_serwo(kat)
```

Pozycje zapamiętane

Zdefiniuj trzy pozycje serwa: MIN, SRODEK i MAKS. Przypisz je do przycisków A/B/C. Na ekranie pokazuj aktualną wartość zadaną i nazwę pozycji.

Sposób realizacji i weryfikacja

1. Zdefiniuj MIN/SRODEK/MAKS jako stałe nastawy, a nie trzy osobne sekwencje sterowania.
2. Po naciśnięciu przycisku najpierw ustaw wartość zadaną, potem wykorzystaj tę samą funkcję do wysłania polecenia i odświeżenia GUI.
3. Sprawdź przejścia między każdą parą pozycji; w razie potrzeby zastosuj łagodną zmianę zamiast skoku.

Pseudokod / zapis algorytmu:

```
pozycja_zapisana(przycisk):
    nastawa = POZYCJE_ZAPISANE[przycisk]
    ustaw_serwo_z_nastawy(nastawa)
```

Silnik DC – kierunek

Przygotuj trzy stany: LEWO, ZATRZYMANIE, PRAWO. Zmiana kierunku musi przechodzić przez ZATRZYMANIE. Nie przełączaj bezpośrednio z pełnej prędkości w jednym kierunku na pełną prędkość w drugim.

Sposób realizacji i weryfikacja

1. Reprezentuj polecenie napędu znakiem: wartość ujemna LEWO, 0 ZATRZYMANIE, dodatnia PRAWO.
2. Jeżeli znak nowego polecenia jest przeciwny do poprzedniego, najpierw ustaw PWM=0, odczekaj krótki czas i dopiero zmień kierunek.
3. ZATRZYMAJ ma mieć priorytet nad innymi poleceniami i natychmiast zerować wartośćysterowania.

Pseudokod / zapis algorytmu:

```
if znak(nowe_polecenie) != znak(poprzednie_polecenie) and poprzednie_polecenie != 0:
    pwm = 0
    czekaj_ms(czas_bezpiecznej_zmiany)
ustaw_kierunek(znak(nowe_polecenie))
ustaw_pwm(abs(nowe_polecenie))
```

Silnik DC – joystick i prędkość

Wykorzystaj oś X joysticka do sterowania kierunkiem i wartością prędkości od -100 do +100. Dodaj martwą strefę wokół zera. Na GUI prezentuj kierunek, wartość zadaną i stan napędu.

Sposób realizacji i weryfikacja

1. Użyj znormalizowanej osi X z ćwiczenia 2; zastosuj martwą strefę przed wyznaczeniem kierunku.
2. Przelicz wartość bezwzględną na PWM, a znak wykorzystaj wyłącznie do wyboru kierunku.
3. Na GUI pokazuj jednocześnie polecenie -100...100, kierunek oraz rzeczywisty stan ZATRZYMANIE/PRACA.

Pseudokod / zapis algorytmu:

```
polecenie = wychylenie_x
if abs(polecenie) < martwa_strefa: polecenie = 0
kierunek = znak(polecenie)
pwm = abs(polecenie)
```

Zadanie 4. Ruch krokowy**Polecenie zadania**

Przy wyłączonym DC uruchom krokowiec sekwencją jednofazową. Wykonaj +20 i -20 kroków z okresem bezpiecznym dla stanowiska. Po każdym wykonanym kroku zmieniaj licznik pozycji. Nowe żądania ruchu ignoruj do zakończenia; C przerywa ruch i wyłącza fazy.

Warunek odbioru: Dwa przeciwne ruchy przywracają licznik do zera. Przerwanie zachowuje liczbę rzeczywiście wysłanych kroków. Student odróżnia licznik od pomiaru położenia.

Po próbie wyłącz fazy krokowca. W zadaniu końcowym sterowany będzie tylko DC; nie buduj wspólnego panelu trzech napędów.

Sposób wykonania i przykłady

Wykorzystaj poniższe opisy i algorytmy do realizacji polecenia. Szczegółowe warianty wykraczające poza polecenie są możliwościami rozbudowy tego samego zadania.

Podstawowe algorytmy sterowania silnikiem krokowym

Silnik krokowy na płycie Demo Board jest silnikiem **czterofazowym, pięcioprzewodowym**. Poszczególne uzwojenia są załączane przez układ sterownika LV8548MC. W przeciwieństwie do typowego sterownika STEP/DIR, program musi określać **kolejność pobudzania faz A, B, C i D**. Kolejność ta decyduje o kierunku ruchu, a czas pomiędzy kolejnymi stanami - o prędkości obrotowej. Demo Board wykorzystuje właśnie taki czterofazowy silnik krokowy sterowany przez LV8548MC.

Idea sterowania

Jeden krok algorytmu polega na ustawieniu odpowiedniego stanu czterech faz silnika, odczekaniu zadanego czasu i przejściu do kolejnego stanu sekwencji. Zmiana kierunku polega na odwróceniu kolejności przechodzenia przez sekwencję.

1. Sterowanie pełnokrokowe - jedna faza aktywna

Najprostsza sekwencja sterowania polega na załączaniu kolejno jednej fazy. Daje mały pobór prądu, ale również mniejszy moment obrotowy.

Krok	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Pseudokod:

```
sekwencja = [1000, 0100, 0010, 0001]
indeks = 0

dopoki liczba_krokov > 0:
    ustaw_fazy(sekwencja[indeks])
    czekaj_ms(opoznienie)
    indeks = (indeks + kierunek) modulo 4
    liczba_krokov = liczba_krokov - 1
```

2. Sterowanie pełnokrokowe - dwie fazy aktywne

Włączenie dwóch sąsiednich faz jednocześnie zwiększa moment trzymający i moment podczas ruchu. Typowa sekwencja ma postać:

Krok	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Pseudokod:

```
sekwencja = [1100, 0110, 0011, 1001]
wykonaj_kroki(liczba_krokow, kierunek, opoznienie, sekwencja)
```

3. Sterowanie półkrokowe

Sterowanie półkrokowe przeplata stany z jedną i dwiema aktywnymi fazami. Powoduje to zwiększenie liczby pozycji pośrednich, a więc zmniejszenie kąta pojedynczego kroku i uzyskanie płynniejszego ruchu.

Krok	1	2	3	4	5	6	7	8
A	1	1	0	0	0	0	0	1
B	0	1	1	1	0	0	0	0
C	0	0	0	1	1	1	0	0
D	0	0	0	0	0	1	1	1

Pseudokod:

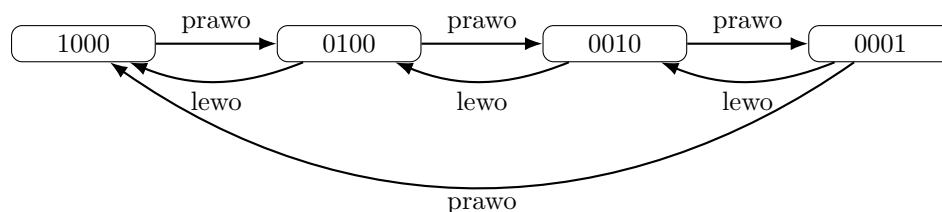
```
sekwencja = [1000, 1100, 0100, 0110,
              0010, 0011, 0001, 1001]
indeks = nastepny_indeks(indeks, kierunek, 8)
ustaw_fazy(sekwencja[indeks])
```

Kolejność faz

Podane sekwencje zakładają kolejność wyprowadzeń A–B–C–D zgodną z oznaczeniami modułu. Jeżeli rzeczywista kolejność przewodów silnika jest inna, silnik może drgać, zatrzymywać się albo obracać w inną stronę. W takim przypadku należy sprawdzić przyporządkowanie faz przed zwiększaniem prędkości.

4. Zmiana kierunku obrotów

Nie jest potrzebna osobna sekwencja dla drugiego kierunku. Wystarczy przechodzić przez tę samą tablicę w odwrotnej kolejności.

**Pseudokod zmiany indeksu:**

```
jezeli kierunek == PRAWO:
    indeks = (indeks + 1) modulo liczba_stanow
w_przeciwnym_razie:
    indeks = (indeks - 1 + liczba_stanow) modulo liczba_stanow
```

5. Sterowanie prędkością

Prędkość silnika krokowego zależy przede wszystkim od odstępu czasu pomiędzy kolejnymi krokami. Im krótszy czas oczekiwania, tym większa częstotliwość kroków i większa prędkość obrotowa.

$$f_k = \frac{1}{T_k}$$

Jeżeli silnik wykonuje N_k pełnych kroków na jeden obrót, prędkość obrotową można oszacować z zależności:

$$n = \frac{60 f_k}{N_k} \quad [\text{obr/min}]$$

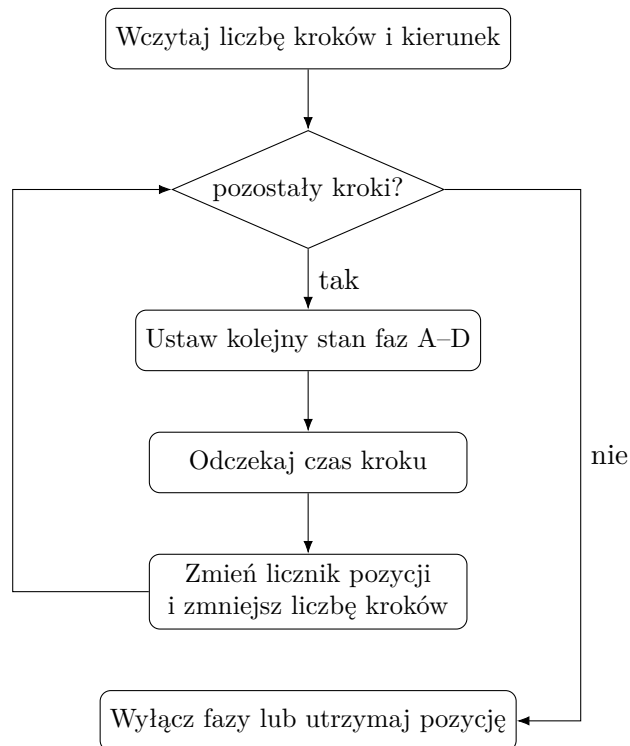
Pseudokod:

```
opoznienie = mapuj(nastawa_predkosci,
                   0, 100,
                   opoznienie_maks, opoznienie_min)

ustaw_fazy(sekwencja[indeks])
czekaj_ms(opoznienie)
```

6. Ruch o zadanej liczbie kroków

Podstawowy algorytm pozycjonowania względnego zlicza wykonane kroki. Kierunek może być zapisany jako +1 lub -1, dzięki czemu ta sama zmienna służy zarówno do wyboru kolejności faz, jak i do aktualizacji pozycji programowej.

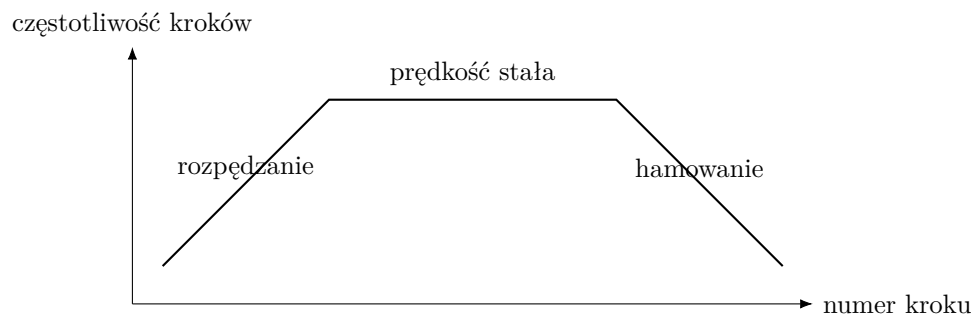


Pseudokod:

```
wykonaj_ruch(ile_krokow, kierunek):
    powtorz abs(ile_krokow) razy:
        ustaw_nastepny_stan_faz(kierunek)
        czekaj_ms(opoznienie)
        pozycja = pozycja + kierunek
```

7. Łagodne rozpędzanie i hamowanie

Silnik krokowy nie zawsze może rozpocząć ruch od razu z dużą częstotliwością. Przy zbyt gwałtownym starcie moment może być niewystarczający i silnik zgubi kroki. Prostim rozwiązaniem jest zastosowanie rampy: początkowo odstęp między krokami jest duży, następnie stopniowo maleje, a przed zatrzymaniem ponownie rośnie.



Prosty algorytm rampy:

```
co każdy krok:
    jeżeli wykonane_kroki < liczba_krokow_rampy:
        opoznienie = opoznienie - zmiana_opoznienia
    jeżeli pozostale_kroki < liczba_krokow_rampy:
        opoznienie = opoznienie + zmiana_opoznienia

    ustaw_nastepny_stan_faz(kierunek)
    czekaj_ms(opoznienie)
```

Uwaga dydaktyczna

W prostym ćwiczeniu wystarczy sterowanie ze stałym opóźnieniem. Rampa przyspieszania i hamowania jest rozszerzeniem pokazującym, dlaczego rzeczywiste układy pozycjonujące nie powinny zawsze przełączać silnika natychmiast na maksymalną prędkość.

Silnik krokowy

Przygotuj komendy ruchu względnego: -100 , -10 , $+10$, $+100$ kroków oraz logiczną funkcję POZYCJA_0 oznaczającą wyzerowanie licznika pozycji programu. Jeżeli stanowisko nie posiada fizycznego czujnika bazowania, POZYCJA_0 nie oznacza rzeczywistego pomiaru położenia.

Sposób realizacji i weryfikacja

1. Prowadź programowy licznik pozycji i aktualizuj go dopiero po wykonaniu zadanej liczby kroków.
2. Komendy względne mają być kolejgowane albo blokowane podczas trwającego ruchu - nie nadpisuj aktywnego polecenia bez świadomej decyzji.
3. Wyraźnie odróżnij POZYCJA_0=0 programowe od fizycznego bazowania z czujnikiem krańcowym.

Pseudokod / zapis algorytmu:

```
cel = pozycja + zmiana
dopoki pozycja != cel and not zatrzymanie_aktywne:
    wykonaj_krok(kierunek)
    pozycja += kierunek
```

Wspólny panel sterowania

Utwórz ekran wyboru napędu. Po wyborze SERWO, DC lub SILNIK KROKOWY pokaż tylko właściwe elementy sterowania. Przycisk ZATRZYMAJ ma być dostępny niezależnie od wybranego napędu.

Sposób realizacji i weryfikacja

1. Przechowuj aktywny napęd w zmiennej `wybrany_naped`; wybór zmienia GUI, ale nie może sam uruchamiać napędu.
2. Wspólny przycisk `ZATRZYMAJ` wywołuje funkcję zatrzymującą wszystkie trzy napędy, niezależnie od aktualnego ekranu.
3. Po zmianie napędu odśwież tylko odpowiednie kontrolki, ale stale pokazuj globalny stan `ZATRZYMANIE/ALARM`.

Pseudokod / zapis algorytmu:

```
zatrzymaj_wszystko():
    ustaw_serwo_bezpiecznie()
    zatrzymaj_silnik_dc()
    zatrzymaj_silnik_krokowy()
```

Rezultat przykładu

Student potrafi wskazać różnicę między sterowaniem serwowym, silnikiem DC i silnikiem krokowym, a każdy tryb posiada ograniczenia oraz funkcję bezpiecznego zatrzymania.

3. Zadanie końcowe**Polecenie zadania końcowego****Sterownik DC z pomiarem i diagnostyką**

Rozbuduj istniejący projekt: joystick odczytywany przez ADS1115 steruje kierunkiem iysterowaniem DC. Ekran pokazuje temperaturę, wilgotność, napięcie joysticka, polecenie oraz stan.

- Wykorzystaj kalibrację `AIN0` i funkcję normalizacji z ćwiczenia 2; zastosuj martwą strefę. Znak polecenia określa kierunek, moduł — PWM ograniczone limitem prowadzącego.
- Po starcie obowiązuje `ZATRZYMANIE`. A uruchamia tylko przy poprawnych danych i joysticku w martwej strefie; C zatrzymuje. B symuluje błąd pomiaru.
- Błąd ADS1115 lub symulowany błąd ustawia `ALARM` i `PWM=0`. C kasuje alarm dopiero po poprawnym odczycie oraz powrocie joysticka do środka; wymagane jest nowe A.
- Steruj DC z jednej funkcji. Wyświetlajysterowanie, nie podpisuj go jako zmierzonej prędkości. Serwo pozostaje w uzgodnionej pozycji, krokowiec wyłączony.

Testy i odbiór

Próba	Oczekiwana reakcja
Start przy wychylnym joysticku	Ruch zablokowany.
Zmiana znaku polecenia	<code>PWM=0</code> przed zmianą kierunku.
B w czasie pracy / C	<code>ALARM</code> i wyłączenie / reset warunkowy.
Restart	DC wyłączony; konieczne nowe A.

Kryteria zaliczenia

Skan I2C, odczyt czujnika i ADS1115, demonstracja trzech rodzajów napędu oraz działający sterownik DC z ograniczeniami i obsługą błędów.

Sprawozdanie

Dołącz projekt, adresy I2C, pomiary AIN0, zakresy napędów, diagram stanów i wyniki testów. Materiały zbierz na zajęciach; PDF opracuj po zajęciach. Możliwość rozbudowy: drugi kanał ADS1115, półkrok i rampa krokowca.

Możliwości rozbudowy zadania końcowego

Zakres rozszerzeń

Zadanie końcowe możesz rozbudować o opisane niżej funkcje. Wykorzystaj działające moduły przygotowane w zadaniach cząstkowych. Możliwości rozbudowy nie zwiększają obowiązkowego zakresu zadania końcowego. Dobierz testy oraz dokumentację do faktycznie zrealizowanych funkcji.

Panel diagnostyczno-pomiarowy I2C

Wykorzystanie przygotowanych funkcji

Rozszerzenie projektu ma połączyć oba poziomy obsługi I2C. Czujniki środowiskowe należy odczytywać przez gotowe bloki/Unity, natomiast ADS1115 ma być obsługiwany bezpośrednio przez zapis i odczyt rejestrów I2C.

Zbuduj **panel diagnostyczno-pomiarowy I2C**. Na ekranie umieść:

- temperaturę, wilgotność i ciśnienie odczytywane przez właściwy sterownik czujnika,
- listę wykrytych adresów I2C,
- surowe wartości ADS1115 dla osi X i Y,
- napięcia obliczone dla obu kanałów,
- wartości X i Y po normalizacji do zakresu $-100 \dots 100$.

Program powinien wyraźnie odróżniać błąd komunikacji z czujnikiem od błędu komunikacji z ADS1115. Rozszerzenie: dodaj możliwość przełączania ekranu pomiędzy stroną ŚRODOWISKO i stroną ADC/I2C.

Poziom rozszerzony

Opracować odczyt temperatury i wilgotności poprzez bezpośredni odczyt i zapis rejestrów I2C czujnika **SHT30** na podstawie dostępnej *dokumentacji*.

Wspólny sterownik trzech napędów

Wykorzystanie przygotowanych funkcji

Wykorzystaj obsługę serwa, zapamiętanych pozycji i silnika DC z zadania 3 oraz obsługę silnika krokowego z zadania 4. Połącz te funkcje we wspólnym panelu, zachowując ograniczenia i bezpieczną zmianę kierunku. Każdy napęd ma korzystać z wcześniej przygotowanej funkcji sterującej.

Opracuj sterownik trzech napędów z jednym interfejsem operatorskim. Joystick steruje silnikiem DC, enkoder ustala położenie serwa, a przyciski realizują ruch względny silnika krokowego. Zaimplementuj wspólny stan **ZATRZYMANIE** oraz blokadę ruchu po wejściu w **ALARM** symulowany przyciskiem.

Rozszerzenie: po skasowaniu alarmu układ pozostaje w **ZATRZYMANIE** i wymaga ponownego świadomego polecenia ruchu dla wybranego napędu.

Dodatkowe testy przy rozbudowie zadania końcowego

W zależności od wybranych funkcji sprawdź poniższe wymagania. Dotyczą one odpowiednich rozszerzeń projektu.

- poprawny odczyt czujników środowiskowych z użyciem właściwego sterownika/Unitu,
- wykonanie skanowania magistrali i poprawna identyfikacja adresów,
- rozróżnienie gotowego sterownika od bezpośredniej obsługi rejestrów I2C,
- poprawny zapis konfiguracji ADS1115 i odczyt rejestru konwersji,
- odczyt co najmniej dwóch kanałów ADS1115,
- poprawne złożenie 16-bitowego wyniku i przeliczenie na napięcie,
- porównanie pomiaru joysticka z ADC ESP32 i ADS1115,
- normalizacja wartości joysticka do zakresu $-100 \dots 100$.
- sterowanie położeniem serwa z ograniczeniem zakresu,
- bezpieczna zmiana kierunku silnika DC,
- regulacja prędkości silnika DC,
- sterowanie ruchem względnym silnika krokowego,
- wspólny panel operatorski,
- jednoznaczny stan ZATRZYMANIE/ALARM.

Dokumentowanie dodatkowych funkcji

Przygotuj sprawozdanie PDF dotyczące wykonanego rozszerzenia i dołącz projekt UIFlow. Udokumentuj zastosowane rozwiązania, wybierając odpowiednie elementy poniższej listy.

- kod programu w UIFlow,
- tabelę wykrytych adresów I2C wraz z identyfikacją urządzeń,
- schemat obu torów pomiarowych joysticka: ADC ESP32 oraz ADS1115,
- tabelę wartości minimum, środek i maksimum dla obu kanałów ADS1115,
- opis sekwencji zapisu konfiguracji i odczytu wyniku z ADS1115,
- porównanie wyników z ćwiczenia 2,
- zdjęcia działającego stanowiska lub czytelne zrzuty ekranu,
- wnioski.
- krótki opis zastosowanego algorytmu wraz ze schematem blokowym, diagramem stanów lub pseudokodem,
- opis sposobu testowania programu,