

POLITECHNIKA WROCŁAWSKA  
WYDZIAŁ ELEKTRONIKI, FOTONIKI I  
MIKROSYSTEMÓW

---

KIERUNEK: Automatyka i Robotyka (AIR)  
SPECJALNOŚĆ: Robotyka (ARR)

**PRACA DYPLOMOWA  
INŻYNIERSKA**

Projekt systemu sensorycznego bazującego na  
protokole MQTT

Design a sensor system based on MQTT protocol

**AUTOR:**  
Marcin Bober

**PROWADZĄCY PRACĘ:**  
Dr inż., Mateusz Cholewiński, K29W12ND02

# Spis treści

# Rozdział 1

## Wprowadzenie

Rozwój technologii powoduje zmiany w każdej dziedzinie życia. Dotyka to nie tylko nasze codzienne otoczenie, ale i przede wszystkim gałęzie przemysłu. To właśnie między innymi potrzeby przemysłu napędzają innowacje poprzez wciąż rosnące zapotrzebowanie na nowe, lepsze i wydajniejsze rozwiązania.

Dzisiejsza technologia pozwala na bardzo precyzyjne sterowanie takim silnikami prądu stałego, a same sterowniki nie zajmują już połowy pokoju. Wręcz przeciwnie - technika analogowa coraz to częściej musi ustępować tej mikroprocesorowej. Powodów takiego stanu rzeczy jest bardzo wiele. Od większej uniwersalności na łatwość poprawy ewentualnych błędów kończąc.

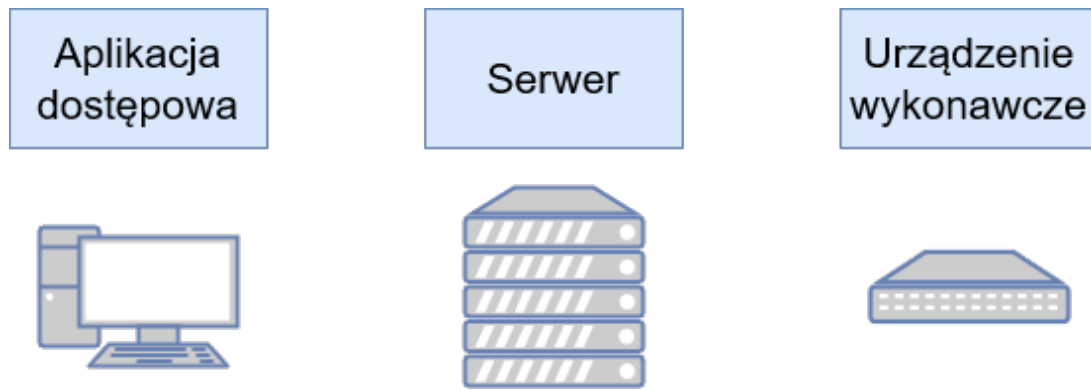
Jeżeli chodzi jednak o przemysł i produkcję urządzeń na wielką skalę to jest jeden parametr który przyćmiewa wszystkie inne. Są to oczywiście koszty produkcji. Koszty zaprojektowania i wdrożenia produktu na rynek są również ważne, ale to koszty produkcji są powodem dla którego księgowi rwą po nocach włosy z głowy szukając oszczędności na każdym drobnym elemencie. W tym miejscu pojawiają się mikroprocesory. Układy o bardzo dużej wszechstronności, które można w każdej chwili przeprogramować całkowicie zmieniając ich działanie. W rękach sprawnego programisty są bardzo wydajne, a jednocześnie niezwykle energooszczędne. Jednakże w mojej pracy energooszczędność nie jest najważniejszym celem.

Myślą przewodnią stojącą za powstaniem tego projektu jest zaprojektowanie systemu zdolnego do sterowania pracą silnika prądu stałego. Co więcej, sterowanie odbywać się będzie w sposób zdalny. Jeden sygnał z komputera i sterownik umieszczony nawet na innej półkuli wysterylizuje silnik tak, jak sobie tego zażyczymy. Projekt dopełniać będzie miła dla oka aplikacja okienkowa, która pozwoli na łatwą obsługę i przejrzysty wgląd w najważniejsze parametry pracy silnika.

Dodatkowym celem dla projektu jest zachowanie jak najniższej ceny, co będzie później warunkowało wybór konkretnych elementów systemu. Ma to również wpływ na poziom trudności projektu, w szczególności regulatora napięcia elementu wykonawczego, ponieważ tanie elementy cechują znacznie gorsze parametry aniżeli drogie, markowe produkty.

### 1.1 Zgrubny opis komponentów

Projekt opiera się o współpracę trzech kluczowych elementów ciągle komunikujących się ze sobą. Komponenty wchodzące w skład systemu zostały umieszczone na rysunku ??.



Rysunek 1.1: Elementy systemu

### 1.1.1 Aplikacja dostępowa

W celu komfortowej obsługi sterownika silnika, została stworzona aplikacja dostępowa w języku C++, która będzie komunikowała się w protokole MQTT. W tym celu wykorzystano otwarto źródłową wersję narzędzia programistycznego jakim jest QT [?]. Biblioteki QT posiadają dobrze zdefiniowane warstwy abstrakcji oraz bardzo wylewną dokumentację, co sprawia że tworzenie zaawansowanych, przenośnych aplikacji okienkowych jest niezwykle łatwe i przyjemne.

### 1.1.2 Urządzenie wykonawcze

Urządzenie końcowe jest dedykowanym rozwiązaniem przygotowanym specjalnie na poczet tego projektu. Bazuje ono na nowoczesnym SoC z rodziny ESP32, który łączy się z brokerem MQTT. Odbierane z niego dane wykorzystane są w procesie sterowania silnikiem. Poza odczytem informacji z brokera, urządzenie publikuje również aktualny stan silnika.

### 1.1.3 Serwer

Ostatnim elementem projektu jest Broker MQTT. Jego zadaniem jest bycie łącznikiem pomiędzy urządzeniami. Pozwala on na łatwe subskrybowanie jak i publikowanie informacji. Ta rola przypadła dla otwarto źródłowego brokera Mosquitto wydanego przez fundację Eclipse. Jest to jeden z popularniejszych programów tego typu, a zawdzięcza to małym wymaganiom sprzętowym, skalowalności oraz dostępności na wielu architekturach sprzętowych. Co więcej, w celu wdrożenia przenośności projektu, oprogramowanie to zostało poddane konteneryzacji, dzięki czemu można łatwo transportować go i uruchamiać na innych komputerach wraz z całą konfiguracją przy użyciu ekosystemu Docker.

# Rozdział 2

## Schemat działania projektu

### 2.1 Założenia

Zostały zdefiniowane ogólne założenia dotyczące każdego z elementów, których spełnienie definiuje kryterium sukcesu.

- System składa się z trzech głównych elementów, połączonych ze sobą:
  - urządzenia wykonawczego,
  - brokera MQTT,
  - aplikacji dostępowej.
- Komunikacja pomiędzy częściami składowymi odbywa się z poprzez sieć bezprzewodową wykonaną w technologii WiFi z wykorzystaniem protokołu MQTT.
- Urządzenie wykonawcze ma zadawać napięcia na silnik prądu stałego w taki sposób, aby uzyskać parametry jak najbardziej zbliżone do zadanych przez aplikację dostępową.
- Aplikacja dostępowa ma za zadanie stanowić prosty i przejrzysty interfejs do sterowania urządzeniem. Umożliwia ona:
  - zadawania prędkości obrotowej silnika,
  - odczytu aktualnej prędkości silnika,
  - zmiany nastaw regulatora PID,
  - odczytu napięcia zasilania urządzenia,
  - odczytu wypełnienia sygnału PWM.
- Broker MQTT on za zadanie być lekkim i szybkim pośrednikiem w komunikacji między aplikacją dostępową, a urządzeniem wykonawczym.

### 2.2 Wymiana informacji

Tematy tworzone i zarządzane przez aplikację okienkową to:

- kp - Wartość członu proporcjonalnego. Parametr sterujący pracą regulatora PID,

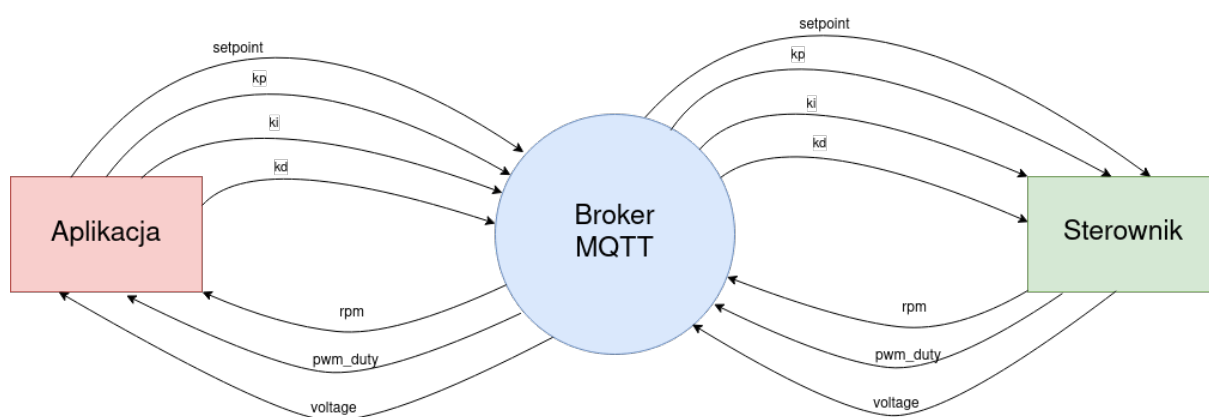
- $k_i$  - Wartość członu całkującego. Parametr sterujący pracą regulatora PID,
- $k_d$  - Wartość członu różniczkującego. Parametr sterujący pracą regulatora PID,
- $\text{setpoint}$  - Wartość zadana dla regulatora. Wyrażona w obrotach na minutę.

Należy zauważyć że są to tylko i wyłącznie wartości zarządzające pracą regulatora PID. Dzięki udostępnieniu ich poza obszar programu istnieje możliwość dynamicznej zmiany parametrów regulatora, co sprawia że nawet osoba nie mająca dużego doświadczenia z regulatorami PID potrafi empirycznie wyznaczyć zadowalające wartości.

Poza tematami utworzonymi przez aplikację dostępową, potrzebne są jeszcze trzy dodatkowe tematy:

- $\text{rpm}$  - ilość obrotów odczytana z silnika przy pomocy enkodera. Wyrażona w obrotach na minutę. Służy jedynie w celach kontrolnych. Daje możliwość zorientować się z jaką prędkością obraca się silnik w rzeczywistości i porównać wynik z wartością zadaną.
- $\text{pwm\_duty}$  - wartość wypełnienia PWM, wyrażona w procentach. Obrazuje stopień wykorzystania dostępnej mocy silnika.
- $\text{voltage}$  - Napięcie zasilania urządzenia.

Obaj klienci brokera MQTT subskrybują nawzajem swoje tematy. Schemat wymiany danych został przedstawiony na rysunku ???. Obrazuje on sposób komunikacji i zależności.



Rysunek 2.1: Schemat wymiany danych w systemie

# Rozdział 3

## Szczegółowy opis sterownika

Przygotowanie urządzenia wykonawczego pochłonęło lwią część czasu przeznaczanego na rozwój projektu. Wynika to z wielu części składowych niezbędnych do przygotowania kompleksowego produktu. Składa się na to między innymi projekt płytki drukowanej oraz projekt oprogramowania. Warto zaznaczyć że do skonstruowania w pełni autorskiego rozwiązania niezbędna jest szczegółowa wiedza z wielu dziedzin. Poczynając od elektroniki na programowaniu systemów wbudowanych kończąc.

### 3.1 Mikrokontroler

Jednostka obliczeniowa wybrana do przeprowadzania obliczeń mieści się w popularnym SoC (z angielskiego System on a chip) o oznaczeniu ESP32-WROOM-32UE. Jest to układ chińskiej firmy Espressif Systems. Został on wybrany ze względu na zintegrowany moduł WiFi oraz bardzo niską cenę wynoszącą poniżej 2\$ za sztukę, co perfekcyjnie wpisuje się w założenia projektu. Ponadto znajduje się w nim wiele ciekawych komponentów takich jak:

- dwurdzeniowy procesor o taktowaniu do 240MHz,
- 520 KiB RAM, 448 KiB ROM, 4 MB FLASH
- Bluetooth w standardzie 4.2 oraz BLE,
- 34 wyprowadzenia GPIO.

Użycie gotowego SoC znacznie upraszcza konstrukcję i gwarantuje poprawną współpracę zawartych w nim układów. Zainstalowany w nim mikrokontroler to ESP32-D0WD-V3.

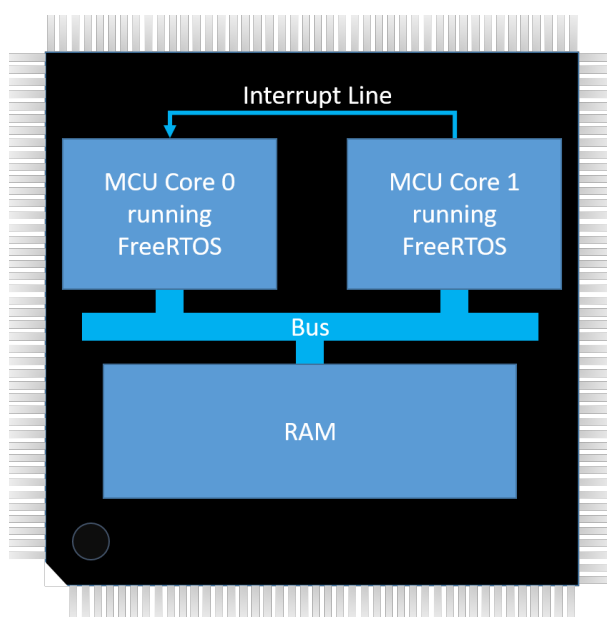


Rysunek 3.1: Wykorzystana jednostka centralna

## 3.2 System operacyjny

Jednym z wielu wyzwań stawianych przed programistą systemów wbudowanych jest stworzenie rozbudowanego programu wykonującego wiele zadań na pojedynczej jednostce obliczeniowej. Powoduje to konieczność wykorzystywania rozbudowanych maszyn stanów i bardzo skomplikowanej logiki. Z pomocą przychodzą systemy operacyjne, które umożliwiają tworzenie programów wielowątkowych i uprzyjemniają korzystanie z procesorów wielordzeniowych. Ze względu na to że użyty w projekcie procesor ma dwa rdzenie logiczne oraz że program ten powinien wykonywać wiele czynności jednocześnie, zdecydowano się skorzystać z funkcjonalności oferowanej przez systemy operacyjne.

Program wykonuje się więc pod kontrolą otwarto źródłowego systemu czasu rzeczywistego FreeRTOS [?]. Jest on znany z dużej szybkości działania. Implementuje on wszystkie elementy niezbędne do pracy z wieloma procesami takie jak muteksy, semaforey czy kolejki priorytetowe. Bez niego obsługa WiFi czy protokołu MQTT byłaby zdecydowanie bardziej skomplikowana i czasochłonna.



Rysunek 3.2: Topologia procesora z użyciem FreeRTOS

Mówiąc o zaletach systemów operacyjnych warto też wspomnieć o ich wadach. Najważniejszą z nich jest narzut obliczeniowy wynikający między innymi z pracy planisty. Z tego powodu część czasu procesora poświęcana jest na pracę samego systemu i nie bierze udziału w wykonywaniu obliczeń na rzecz programu, aczkolwiek nie są to jednak znaczące wartości. Z tego powodu systemy operacyjne znajdują szerokie zastosowanie nie tylko w zastosowaniach profesjonalnych, bo dzięki prostocie FreeRTOS, jest on coraz to częściej wykorzystywanych przez amatorów i hobbistów.

## 3.3 Framework

Przy opracowywaniu kodu źródłowego sterownika został użyty framework ESP-IDF [?] przygotowany przez producenta procesora. Na stronie dostawcy tego rozwiązania można znaleźć także obszerną dokumentację tłumaczącą wykorzystanie frameworka w praktyce



wraz z wieloma przykładami użycia. Zastosowanie gotowej abstrakcji zdejmuje z dewelopera obowiązek implementacji wielu skomplikowanych aspektów programu. W tym projekcie szczególnie przydatna jest implementacja stosu TCP/IP i protokołu MQTT.

## 3.4 Kolejki i wątki

### 3.4.1 Opis

Największą zaletą posiadania systemu operacyjnego jest możliwość tworzenia osobnego procesu dla każdego zadania. To znowu pozwala dokładnie kontrolować czasy wywołań i częstotliwości odpowiednich wydarzeń. Główne zadania stawiane temu sterownikowi to:

- obsługa silnika wraz z liczeniem PID,
- pomiary napięcia zasilania,
- wysyłanie danych przez MQTT,
- obsługa WiFi, odbieranie danych z MQTT, planista i obsługa przerwań

W ten sposób kształtują się cztery główne procesy, które biorą udział w pracy systemu:

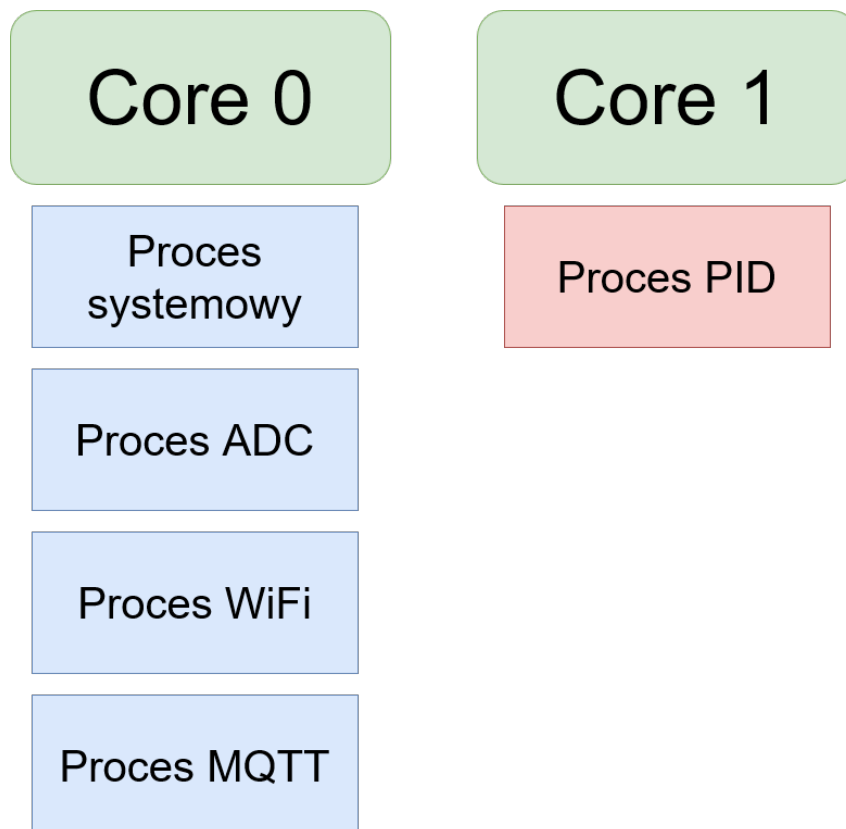
**Główny** Jest głównym procesem w systemie. Odpowiada on za początkową konfigurację peryferiów przy starcie systemu. Z niego powstaje także reszta procesów. Po poprawnym starcie zajmuje się on obsługą połączenia z siecią, klientem MQTT, przerwaniem oraz planistą systemu.

**PID** Proces PID wykonuje się dokładnie co 10ms. Każde odchylenie w czasie tego procesu powodowałoby niedokładną pracę regulatora PID, co skutkowałoby niestabilnym zachowaniem się silnika. Z tego też powodu, ten proces został na stałe powiązany do drugiego rdzenia procesora oraz otrzymał bardzo wysoki priorytet. Jego schemat działania został zamieszczony na rys. ??

**ADC** Proces ADC odpowiedzialny jest za pomiary zasilania. Wykonuje się co około 20ms, ale ze względu na ustawiony bardzo niski priorytet ustępuje każdemu innemu zadaniu. Wykonuje się więc tylko gdy procesor nic innego nie robi. Nie stanowi to żadnego problemu, ponieważ opóźnienia w wykonywaniu tego procesu nie wpływają z znacznym stopniem na prawidłową pracę systemu. Informacja przez niego generowana trafia wyłącznie na panel kontrolny użytkownika i jest wyświetlana w aplikacji dostępowej, gdzie aktualizowana jest z częstotliwością około 5Hz. Drobne odchyłki w czasie aktualizacji są więc niewykrywalne. Proces ten został przypięty do rdzenia pierwszego.

**WiFi** Proces WiFi. Rola tego procesu jest bardzo skromna i ogranicza się jedynie do inicjalizacji połączenia z WiFi. Po wykonaniu tego zadania proces ten się kończy, a rolę obsługi połączenia bezprzewodowego przejmuje proces systemowy.

**MQTT** Zadaniem procesu MQTT jest odbieranie danych z kolejek priorytetowych FIFO i wysyłanie ich do brokera. Otrzymał on średni priorytet. Wykonuje się w momencie gdy, w którejś z kolejek pojawiają się jakieś nowe dane. On również został przypisany do rdzenia pierwszego.



Rysunek 3.3: Diagram wątków

### 3.4.2 Implementacja menadżera wątków

Został stworzony specjalny moduł odpowiedzialny za zarządzanie wątkami. Pozwala to na łatwe dodawanie i usuwanie funkcjonalności z systemu. Dodanie nowego procesu ogranicza się do wypełnienia struktury przedstawionej na listingu ??, a następnie dopisania kolejnej pozycji w liście z listingu ??.

Listing 3.4.1: Struktura konfiguracyjna wątku

Listing 3.4.2: Lista wątków

Implementacja wątków zawartych w systemie przedstawiona została na listingu ??. Jest to tablica wcześniej wspomnianych struktur. Pierwsze pole struktury jest uchwyt do wybranej funkcji. Następnie należy podać nazwę procesu. Jest to niezwykle przydatne przy szukaniu błędów, ponieważ nazwa ta jest używana podczas wypisywania informacji diagnostycznych. Kolejne pola to wielkość stosu, przekazywane parametry, priorytet wykonywania, uchwyt do procesu oraz numer rdzenia który ma zajmować się obsługą skonfigurowanego zadania.

Listing 3.4.3: Konfiguracja wątków

### 3.4.3 Implementacja menadżera kolejek

Zastosowanie kolejek priorytetowych umożliwia komunikację między procesami. Można je deklarować i inicjalizować w modułach, które polegają na przesyłanych danych, ale sprowadza to znaczny bałagan w kodzie. Dużo lepszym rozwiązaniem jest umieszczenie ich w jednym module, który zadba o ich konfigurację i inicjalizację. Zostało to zaprojektowane w sposób bliźniaczy do menadżera wątków. Podstawą jest struktura pokazana w listingu ??.

Listing 3.4.4: Struktura konfiguracyjna kolejki

Struktura ta zawiera uchwyt do kolejki, wielkość transportowanego elementu, dopuszczalną ilość elementów w kolejce oraz nazwę. Przykład wykorzystania został przedstawiony w listingu ??.

Listing 3.4.5: Konfiguracja kolejek FIFO

Uchwyty do kolejek są przypisywane podczas inicjalizacji kolejek. Kod dotyczący inicjalizacji umieszczony jest pod listingiem ??.

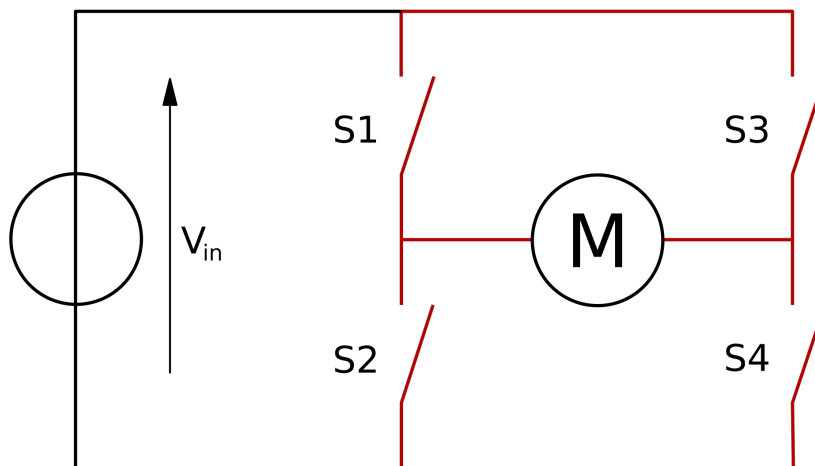
Listing 3.4.6: Inicjalizacja kolejek FIFO

Wykorzystanie kolejek poza modulem byłoby mocno utrudnione bez zdefiniowania odpowiednich makr, które odnoszą się do przypisanych uchwytów. Kod znajduje się w listingu ??.

Listing 3.4.7: Makra do uchwytów kolejek FIFO

## 3.5 Mostek H

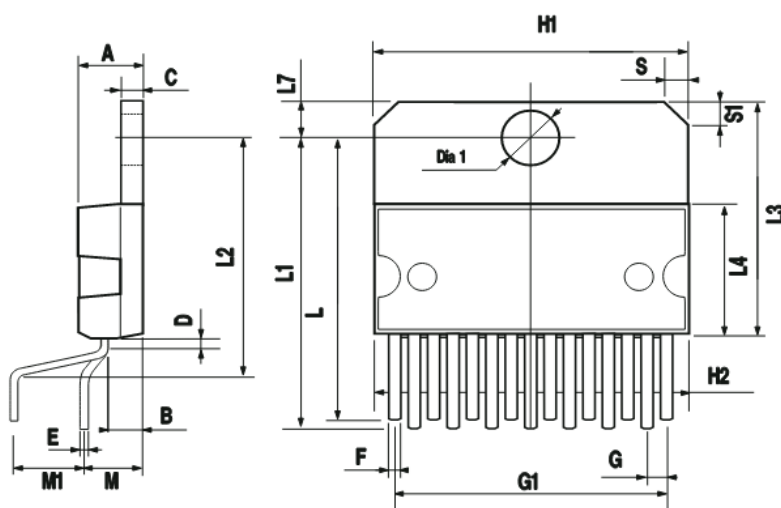
Głównym zadaniem mikrokontrolerów jest zbieranie danych z czujników, wykonywanie obliczeń i podejmowanie decyzji. Nie zostały jednak stworzone do sterowania elementami o wyższym napięciu czy dużych prądach. Wyjścia mikrokontrolerów potrafią dostarczyć zazwyczaj około kilkudziesięciu miliamperów prądu. W przypadku mikrokontrolera użytego w projekcie dokumentacja [?] informuje o bardzo wysokim prądzie wyjściowym dochodzącym nawet do 40mA w sprzyjających warunkach. Jest to wystarczające natężenie prądu do zasilenia diody LED (ok. 20mA) czy sterowania tranzystorem. Z drugiej strony jest to wielokrotnie za mało, aby uruchomić silnik prądu stałego. Takie silniki zazwyczaj nie dość że działają na znaczenie wyższych napięciach niż 3.3V to często potrafią także zużyć ponad 2A prądu podczas obciążenia. Jednym ze sposobów kontrolowania prędkości obrotowej takiego jest zastosowanie wcześniej wspomnianego tranzystora lub przełącznika. Minusem takiego rozwiązania jest brak możliwości zmiany kierunku obrotów. Z tego powodu najczęściej stosowanym rozwiązaniem, które pozwoli na sterowanie silnikiem w obu kierunkach jest użycie mostka H. Jego schemat przedstawiono na rys. ??.



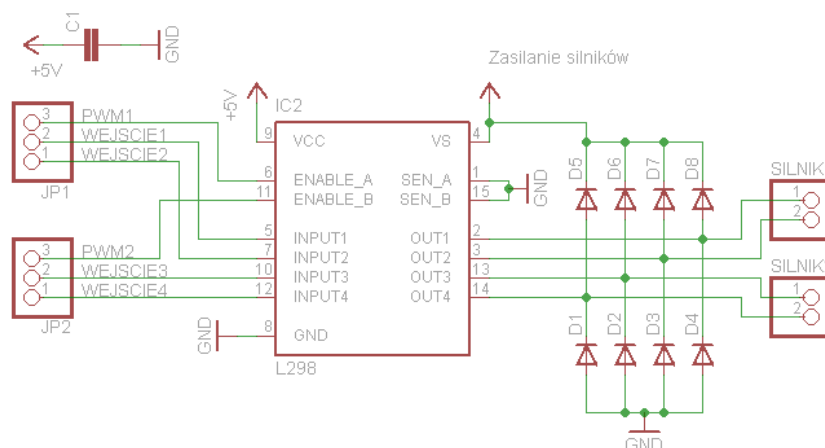
Rysunek 3.4: Ogólny schemat mostka H

W roli mostka H został wyselekcjonowany bardzo popularny układ scalony L298N w obudowie Multiwatt15. Nie jest to nowy układ scalony, ale jest on przetestowany i zdecydowanie wystarczający dla tego zastosowania. Może się on poszczycić napięciem zasilania silników do 46V i prądem szczytowym na poziomie 2A. W rzeczywistości posiada on dwa osobne kanały, co sprawia że z jego pomocą możliwe jest jednoczesne sterowanie dwoma silnikami. Jednakże projekt ten zakłada wykorzystanie tylko jednego silnika, więc drugi kanał pozostanie nieaktywny. Może to spowodować pozytywny wpływ na wolniejsze nagrzewanie się układu, co z pewnością przełoży się na jego stabilniejszą pracę.

Układ scalony zastosowany w projekcie został przedstawiony na rys. ???. Redukuje on znacząco poziom skomplikowania urządzenia ponieważ w celu uruchomienia go wystarczą cztery diody redukujące prądy indukowane na cewkach silników podczas ich pracy oraz kondensator filtrujący zasilanie [?]. Przykładowe podłączenie jest zaprezentowane na rys. ???.



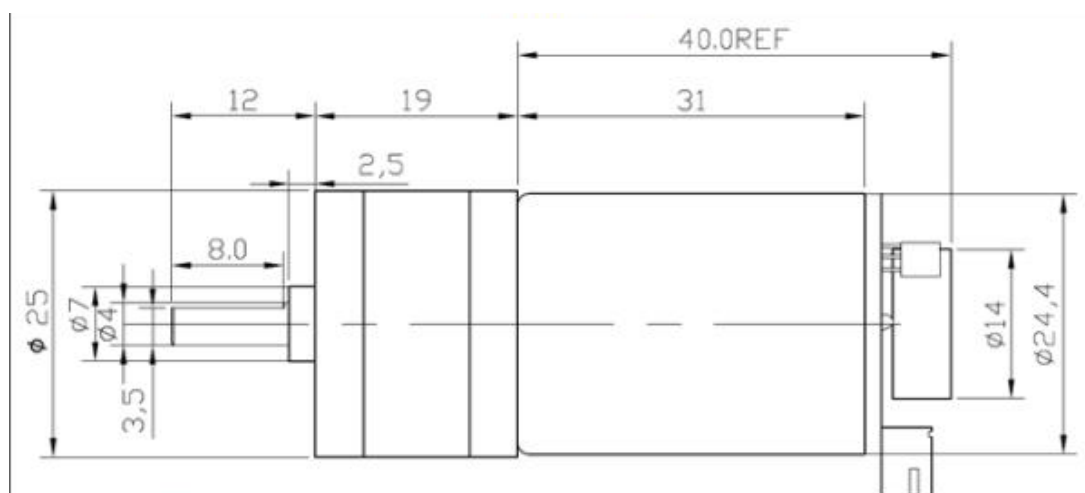
Rysunek 3.5: Schemat wykorzystanego mostka H



Rysunek 3.6: Schemat podłączenia mostka H

## 3.6 Silnik

Silnik jest to bardzo konkurencyjnie wyceniany produkt firmy DFROBOT. Został on wybrany w głównej mierze ze względu na swoją niską cenę i parametry, które w zupełności wystarczą do zrealizowania tego projektu. Jest on wyposażony w metalową przekładnię zapewniającą przełożenie napędu w proporcji 20:1. Ponadto, na końcu silnika znajduje się fabrycznie zamontowany enkoder kwadraturowy, który produkuje 11 impulsów na każdy obrót wałka głównego. Schemat silnika wraz z enkoderem umieszczony został na rys. ??



Rysunek 3.7: Schemat wykorzystanego silnika

## 3.7 Płytką PCB

### 3.7.1 Opis

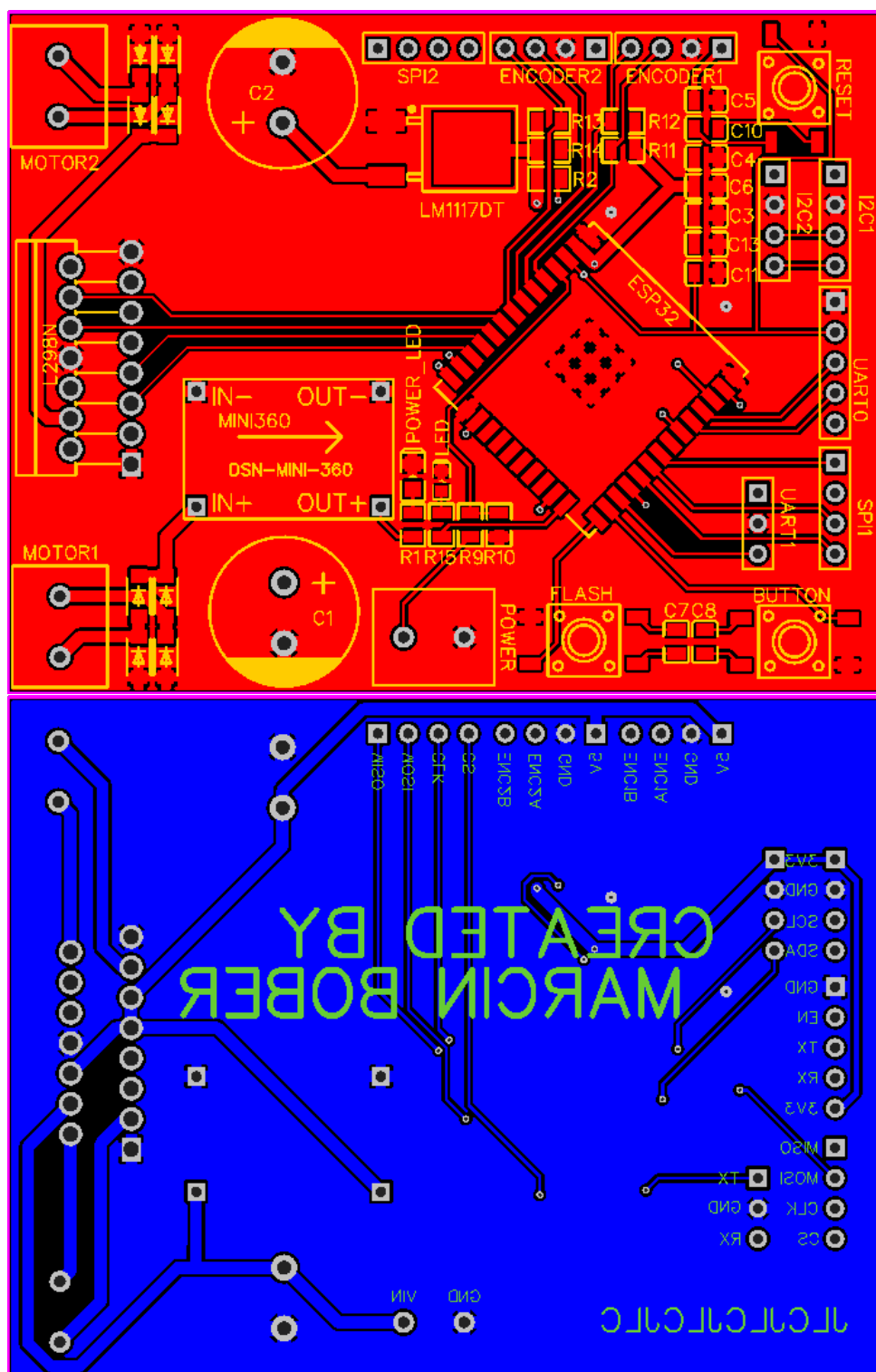
Urządzenie wykonawcze zawiera mikrokontroler, mostek H oraz element wykonawczy w postaci silnika prądu stałego wyposażonego w enkoder kwadraturowy. Poza możliwością zadawania napięcia na silnik, urządzenie dysponuje funkcją pomiaru napięcia zasilania. Poza wymienionymi jednostkami na płytce drukowanej znajdują się także wszystkie elementy niezbędne do prawidłowej pracy procesora takie jak kondensatory i rezystory.

### 3.7.2 Wygląd

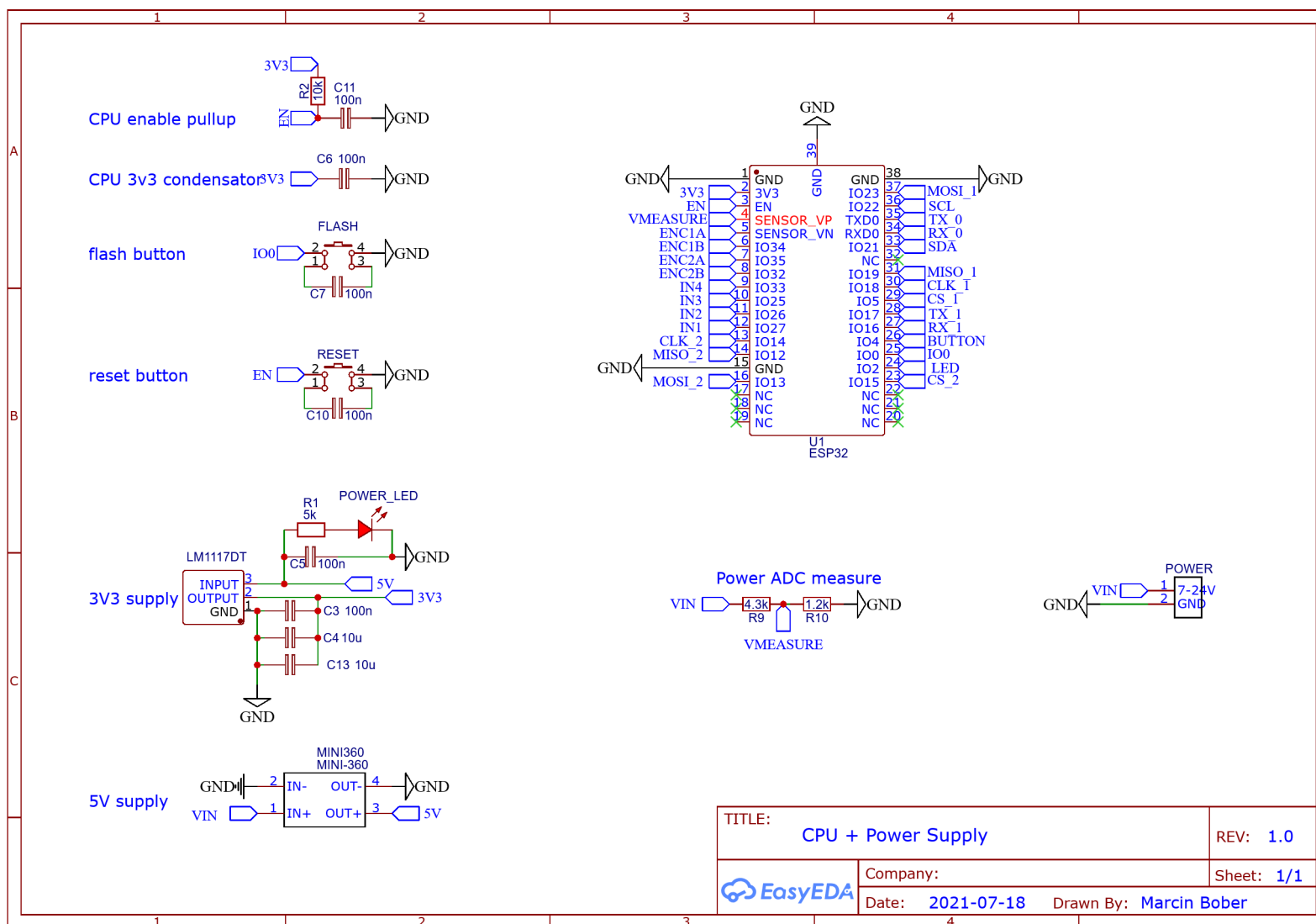
Przygotowany wzór płytki drukowanej z rozlokowanymi elementami i poprowadzonymi ścieżkami został złączony do tego dokumentu. Obie strony płytki zarówno górna jak i dolna dostępne są jako rysunek ??.

### 3.7.3 Schemat

Schemat składa się z trzech stron. Pierwsza strona (rys. ??) dotyczy sekcji zasilania oraz połączeń z SoC wraz z elementami niezbędnymi do jego prawidłowej pracy. Druga strona (rys. ??) opisuje połączenia enkoderów oraz mostka H. Trzecia strona (rys. ??) są to jednie wyprowadzenia interfejsów.

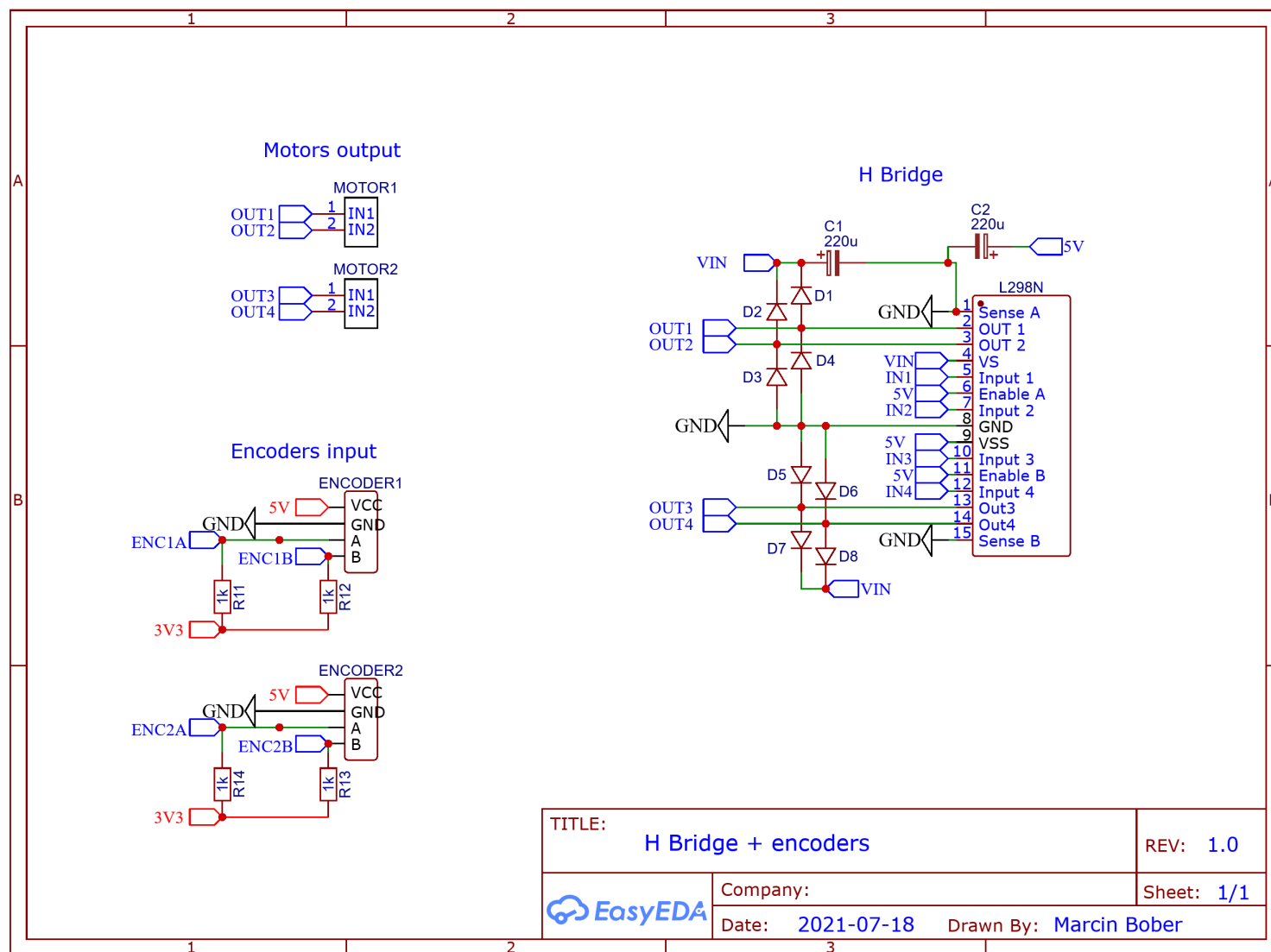


Rysunek 3.8: Wygląd PCB

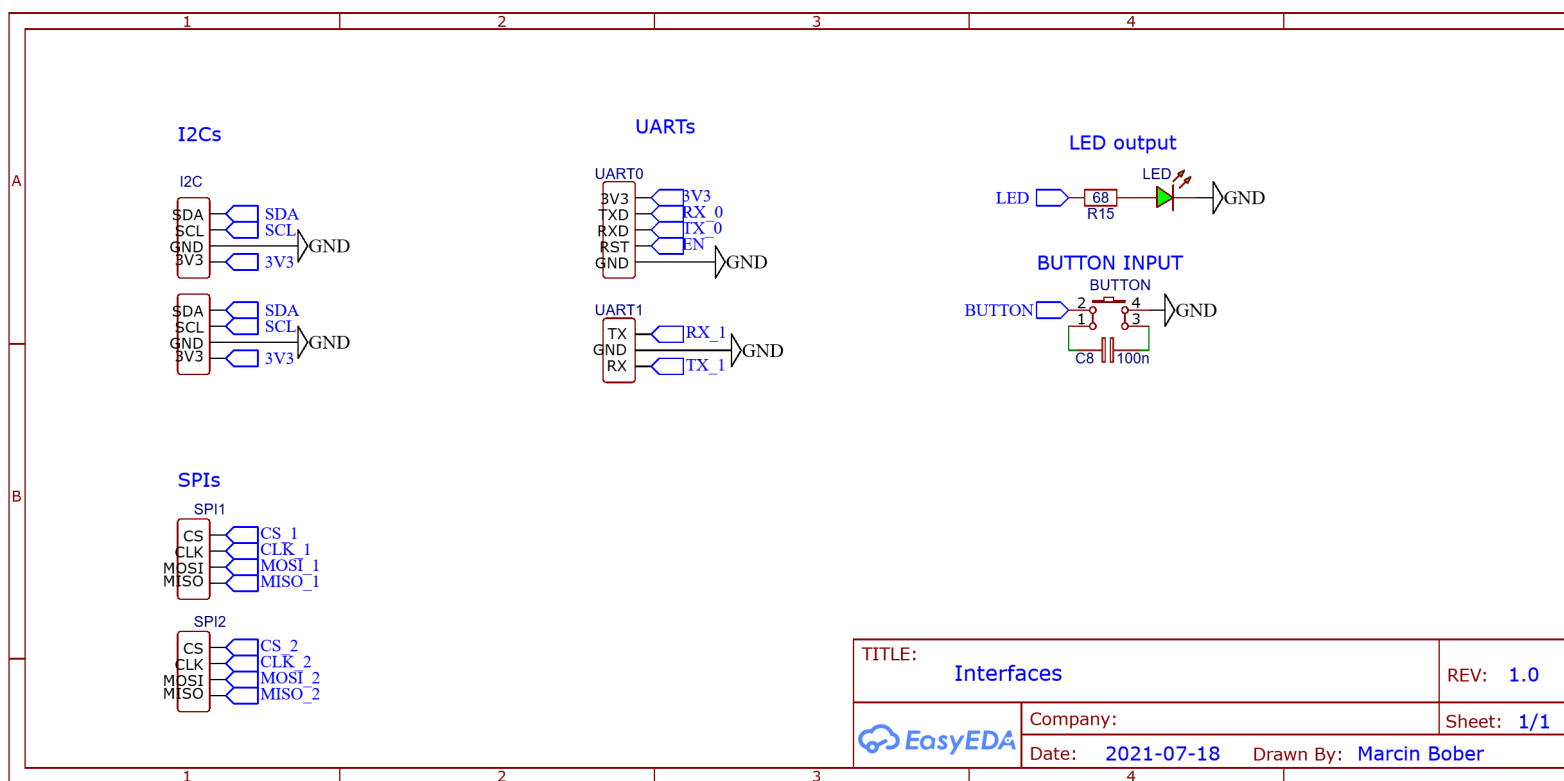


Rysunek 3.9: Schemat płytki PCB (CPU+PSU)





Rysunek 3.10: Schemat płytki PCB (mostek + enkodery)



Rysunek 3.11: Schemat płytki PCB (interfejsy)

## 3.8 Licznik impulsów

### 3.8.1 Opis

Licznik impulsów (Pulse Counter) jest peryferium, które znakomicie ułatwia obsługę enkoderów. Procesory bez takich udogodnień są skazane na odczytywanie przerwań za każdym razem jak enkoder wyśle impuls, aby następnie za pomocą kodu Graya rozpoznawać kierunek obrotu [?]. W takim zastosowaniu użycie enkodera jako pokrętła jest zrozumiałe. Zmieniając enkoder ręczny na taki zamontowany na silniku znacznie zwiększamy częstość generowania impulsów, a co za tym idzie również przerwań, odbieranych w sekundzie pracy procesora. W przypadku użytego w tym projekcie silnika prędkość obrotowa silnika przed przekładnią równa jest:

$$V = \frac{n \cdot k}{t_n} = \frac{300 \cdot 20.4}{60s} = 102RPS$$

Gdzie:

- $n$  - ilość obrotów silnika za przekładnią w ciągu minuty pracy,
- $k$  - współczynnik redukcji przekładni,
- $t_n$  - okres czasu z którego brane są impulsy.

Wiedząc że zintegrowany enkoder wytwarza 11 impulsów na każdy obrót, nietrudno jest wyliczyć że procesor zostanie zasypany ilością 1122 przerwań w ciągu zaledwie jednej sekundy pracy. Taka kolej rzeczy z pewnością odbiłaby się na wydajności urządzenia. W ekstremalnym przypadku użycie większego silnika lub dokładniejszego enkodera mogłoby doprowadzić do sytuacji, gdzie procesor nie byłby w stanie obsłużyć takiego natłoku przerwań. Rozwiązaniem tego problemu jest użycie liczników impulsów, które odciążają mikrokontrolery z takich wyzwań. Posiadają one swoją pamięć w której gromadzą dane na temat zliczonych sygnałów. W takiej sytuacji zadaniem procesora jest jedynie czytanie z tej pamięci. ESP32 posiada aż 8 takich liczników, co stwarza ogromne możliwości wykorzystania tego układu.

Producent układu w dokumentacji procesora informuje że minimalny okres pomiędzy impulsami, który gwarantuje nie pominięcia żadnego odczytu nie może być mniejszy jak  $12.5ns$ . Wykorzystany w projekcie silnik obraca się z prędkością 100 obrotów na sekundę, a przy każdym obrocie enkoder produkuje 11 impulsów. Częstość występowania impulsów jest następująca:

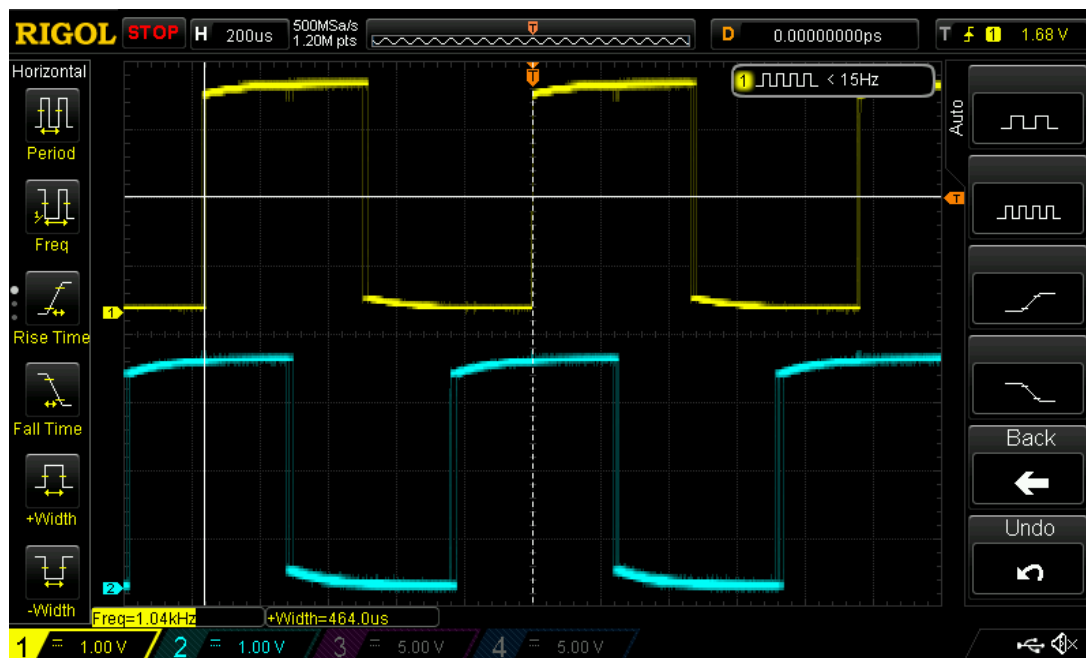
$$f = \frac{n}{t_n} = \frac{100 \cdot 11}{1s} = 1100Hz$$

Gdzie  $n$  oznacza generowaną ilość impulsów przez enkoder w okresie czasu  $t_n$ . Okres pomiędzy impulsami jest odwrotnością częstości:

$$T = \frac{1}{f} = \frac{1}{1100Hz} = 909.09\mu s$$

Obliczenia zostały poparte pomiarami z oscyloskopu. Pomiar został umieszczony na rys. ???. Widać na nim że częstość waha się w okolicy 1040Hz. Delikatne zwiększenie napięcia zasilania spowodowałoby minimalny wzrost obrotów, więc uzyskanie wyliczonej częstości nie jest problemem. Niemniej jednak, okres pomiędzy impulsami enkodera

jest o rzędy wielkości większy niż okres graniczny dla tego licznika. Oznacza to że można wypełnić ufać wskazaniom licznika ponieważ że żadne impulsy nie są pomijane. Abstrahując od okresów pomiędzy impulsami obraz z oscyloskopu wspianiale prezentuje przesunięcie przebiegów pomiędzy kanałami enkodera.



Rysunek 3.12: Pomiar impulsów z oscyloskopu

### 3.8.2 Konfiguracja

W przypadku tego układu, konfiguracja jest bardzo prosta i sprowadza się do wypełnienia zaledwie dwóch struktur i wysłania ich do peryferium.

Listing 3.8.1: Konfiguracja licznika impulsów

Użycie jednej struktury powoduje włączenie jednego z dwóch kanałów w liczniku. Licznik automatycznie ewaluuje zbocza narastające na obu wyjściach enkodera, co podwaja wykrywaną ilość impulsów. Poprawne skonfigurowanie obu kanałów w liczniku w sposób pokazany na listingu ?? sprawia że licznik poprzez analizę zbocz narastających jak i opadających zwiększa dokładność enkodera aż czterokrotnie. Ten sprytny zabieg sprawił że użyty w projekcie tani enkoder dostarcza nam aż 897 impulsów na każdy obrót wału wyjściowego z przekładni. Skutkuje to impulsem co około  $0.4^\circ$ . Proces ten został opisany w publikacji [?]. Poza konfiguracją licznika musimy dbać jedynie o odczyt zliczonych impulsów oraz zerowanie pamięci po wykonanym odczycie. Funkcje odpowiedzialne za te czynności zostały przedstawione w listingu ??

## 3.9 Modulacja szerokości impulsu - PWM

### 3.9.1 Opis

### 3.9.2 Implementacja

Listing 3.9.1: Inicjalizacja PWM

Listing 3.9.2: Zmiana wypełnienia PWM

## 3.10 Regulator PID

### 3.10.1 Implementacja

Najważniejszą częścią programu jest implementacja regulatora PID. Odpowiednie zaprojektowanie jego działania jest kluczowe w kwestii wydajności oraz poprawności funkcjonowania całego systemu. Schemat funkcjonowania algorytmu jest prosty. Został on przedstawiony na listingu ??.

Czytamy ilość impulsów z układu odpowiedzialnego za ich zliczanie. Przy okazji czyścimy jego rejestry, aby mógł zliczać już kolejne impulsy w trakcie, gdy my będziemy dokonywać obliczeń. Realizacja tego jest przesłonięta przez marko zawarte w ?? w celu zwiększenia czytelności kodu.

Listing 3.10.1: Pobieranie wartości i czyszczenie rejestrów licznika

Warto zaznaczyć że nie zapisujemy momentu zaczytania danych. Musimy więc zagwarantować że funkcja ta będzie wykonywać się cyklicznie w ściśle określonym przedziale czasowym. Każde odstępstwo od tego będzie powodowało błędy w poprawnym działaniu algorytmu.

Listing 3.10.2: Pętla regulatora PID

Kolejnym krokiem jest policzenie uchybu regulacji. Jest to tradycyjnie różnica wartości ustalonej i zliczonych impulsów. Następnie liczymy wartości dla każdego z członów regulacji, jednocześnie gwarantując że mieszczą się one we wcześniej zdefiniowanych przedziałach. To również ukryte zostało pod makrem dostępnym w listingu ??.

Wartości każdego z członów mnożymy poprzez wybrane przez użytkownika współczynniki. Są one zapisane w zmiennych dzięki czemu istnieje możliwość zmiany ich wartości dynamicznie podczas pracy programu. Teraz wystarczy już zsumować wszystkie wartości oraz zapewnić że uzyskana wartość nie przekroczy wartości granicznych możliwych do ustawienia dla PWM. W tym momencie część obliczeniowa jest gotowa. Istnieje jednak spore ryzyko że uzyskana w ten sposób wartość nie będzie mieściła się w zakresie przyjmowanym przez peryferium odpowiedzialnym za modulowanie napięcia. Niezbędne jest więc wykonanie kolejnej normalizacji do akceptowalnych wartości ??.

Listing 3.10.3: Normalizacja wartości regulatora

### 3.10.2 Proces PID

Pętla regulatora PID zaimplementowana jest w swoim własnym procesie. Ułatwia to kontrolę częstości wykonywania obliczeń i umożliwia zatrzymanie procesu gdy jest on niepotrzebny. Na przykład przy braku połączenia z serwerem MQTT.

Implementacja procesu zawarta jest w listingu ??. Pierwsze linie kodu rozpoczynają inicjalizację niezbędnych obiektów. W skład nich wchodzi:

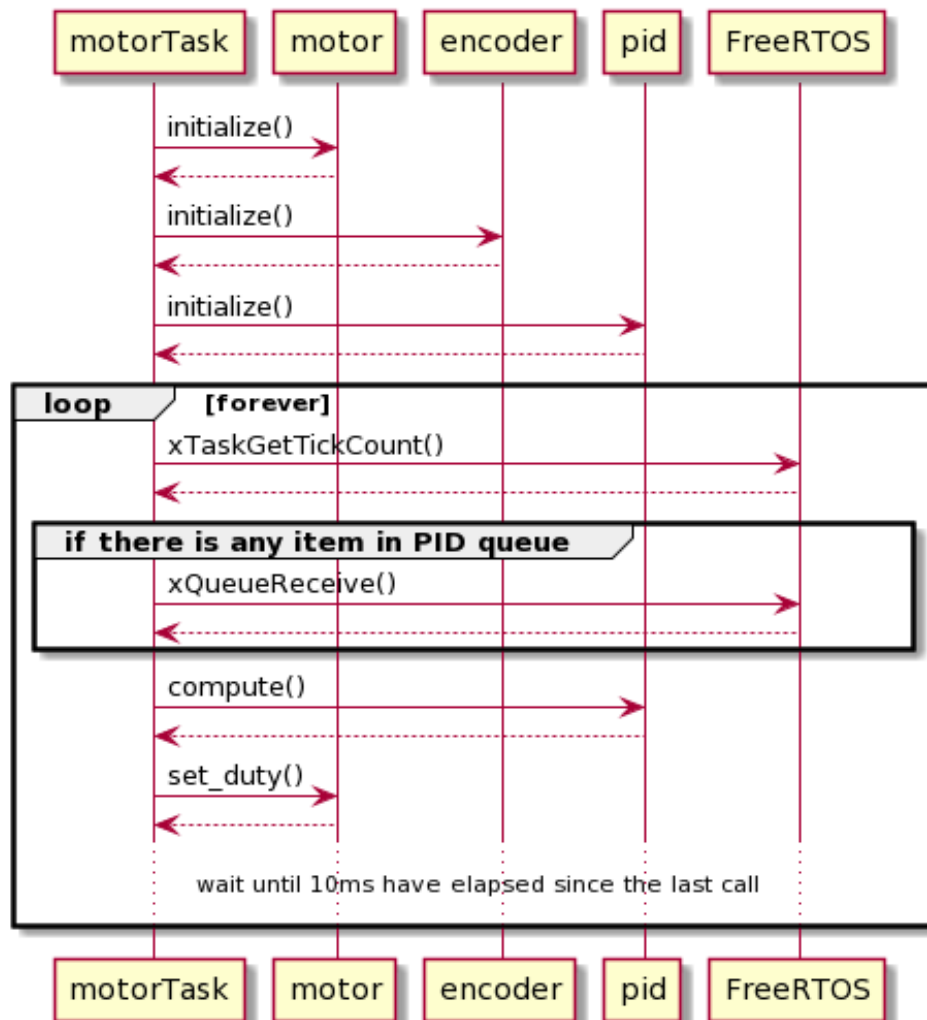
- config - struktura zawierająca konfigurację parametrów regulatora. Niezbędna do dynamicznej zmiany parametrów PID.

- engine - instancja sterująca peryferium PWM,
- encoder - instancja do obsługi enkodera,
- pid - instancja regulatora PID,
- pid\_result - zmienna zawierająca wyliczoną wartość regulacji,

Kolejne linie kodu wykonują się w nieskończonej pętli. Następuje w niej odczyt aktualnego stanu licznika systemowego. Jest on wykorzystywany do wyliczenia jak długo wykonują się operacje w pętli. Następnie sprawdzana jest kolejka FIFO. W przypadku znalezienia się w niej nowej konfiguracji dla regulatora, jest ona natychmiast zastosowana.

#### Listing 3.10.4: Proces wykonywanie PID

Po opuszczeniu instrukcji warunkowej następuje obliczenie wartości regulatora i ustawienie modulacji szerokości impulsu zgodnie z uzyskanym wynikiem. Ostatnim krokiem jest zatrzymanie procesu. Czas zatrzymania jest zależny od prędkości wykonywania pętli i warunkuje częstotliwość wyzwalania PID na 100Hz. Na obrazku ?? znajduje się diagram aktywności opisywanego procesu.



Rysunek 3.13: Diagram aktywności procesu PID



## 3.11 Pomiar napięcia

Pomiar napięcia dokonywany jest przy użyciu przetwornika analogowo-cyfrowego wbudowanego w mikroprocesor [?]. Posiada on bowiem dwa 12 bitowe przetworniki, których wejścia są multipleksowane na aż 18 pinów. Po wczytaniu się w dokumentację okazuje się jednak że użyteczność tych przetworników ma wiele wykluczeń. Najważniejszym z nich jest interferencja przetwornika ADC2 z peryferium odpowiedzialnym za połączenie WiFi. Oznacza to że w projektach od których wymagamy stałego połączenia z siecią jesteśmy ograniczeni jedynie do 8 pinów obsługujących odczyty analogowe.

Kolejnym ciekawym aspektem tego układu jest możliwość programowego wyboru tłumienia sygnału wejściowego. Producent umożliwia nam wybór od 0dB poprzez 2.5dB oraz 6dB aż do 11dB tłumienia. Im wyższy parametr zostanie wybrany tym wyższe napięcia jest w stanie obsłużyć przetwornik. Niemniej jednak wybór wyższych oczek znacznie degraduje precyzję pomiaru.

### 3.11.1 Konfiguracja ADC

Po dokładnym przestudiowaniu dokumentacji przyjęte zostało ustawienie tłumienia na poziomie 2,5dB. Patrz ???. Wyższe ustawienia wprowadzają znaczną niedokładność pomiarów. Według dokumentacji [?] takie ustawienie sprawia że efektywny zakres pomiarowy kształtuje się w zakresie od 100mV do 1250mV na pinie mikrokontrolera. Błędy pomiaru zadeklarowane przez producenta nie powinny przekroczyć  $\pm 30mV$ . Jednakże, zasilanie płytki jest zaprojektowane do obsługi dużo większych napięć, tak aby móc wykorzystywać szeroki zakres silników. Podanie napięcia wyższego jak zasilanie procesora, które w tym wypadku wynosi 3.3V, spowodowałoby trwałe uszkodzenia układu. Do bezpiecznego pomiaru napięcia zasilania niezbędne jest więc zastosowanie dzielnika napięcia.

Rozdzielczość peryferium została ustawiona na 12 bitów. Wynika z tego że istnieje  $2^{12} = 4096$  poziomów kwantyzacji. Nie spodziewam się że mój autorski projekt płytki drukowanej ?? będzie miał na tyle niski szum kwantyzacji, aby móc w pełni wykorzystać taką rozdzielczość, ale w tym przypadku czas pomiaru nie jest tutaj kluczowy. Założenie wykonywania pomiaru zakłada że i tak będzie wykonywane wiele pomiarów pod rząd, aby później móc wyciągnąć z nich średnią.

Listing 3.11.1: Konfiguracja przetwornika ADC

### 3.11.2 Dzielnik napięcia

Zadecydowałem że w tym projekcie wystarczający będzie najprostszy dzielnik oparty na dwóch rezystorach. Pozwoli nam on liniowo podzielić napięcie, tak aby nigdy nie przekroczyło ono wartości niebezpiecznej dla mikroprocesora. Do stworzenia dzielnika wykorzystałem rezystory z szeregu E24 [?] o wartościach  $9.1k\Omega$  oraz  $1k\Omega$  i podłączyłem je w sposób pokazany na schemacie ??.

Obliczenie dostępnego zakresu pomiarowego przetwornika adc odbywa się przy pomocy wzoru na dzielnik napięcia [?]. Jest on następujący:

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$

Gdzie napięcie wejściowe  $V_{out}$  jest maksymalnym napięciem dostępnym na wejściu przetwornika. Zgodnie z informacją zawartą w dokumentacji procesora [?] wybranie tłumienia na poziomie 2.5dB skutkuje otrzymaniem 1250mV jako granicy efektywnego zakresu pomiarowego. Przy czym producent zaznacza że ta wartość zależy od konkretnego egzemplarza i może się wahać.  $R_1$  oraz  $R_2$  to użyte rezystory w dzielniku. W tym przypadku są to kolejno  $9.1k\Omega$  oraz  $1k\Omega$ .

Dla maksymalnego obsługiwanego napięcia z przedziału wzór to:

$$1.250V = \frac{1k\Omega}{9.1k\Omega + 1k\Omega} \cdot V_{in}$$

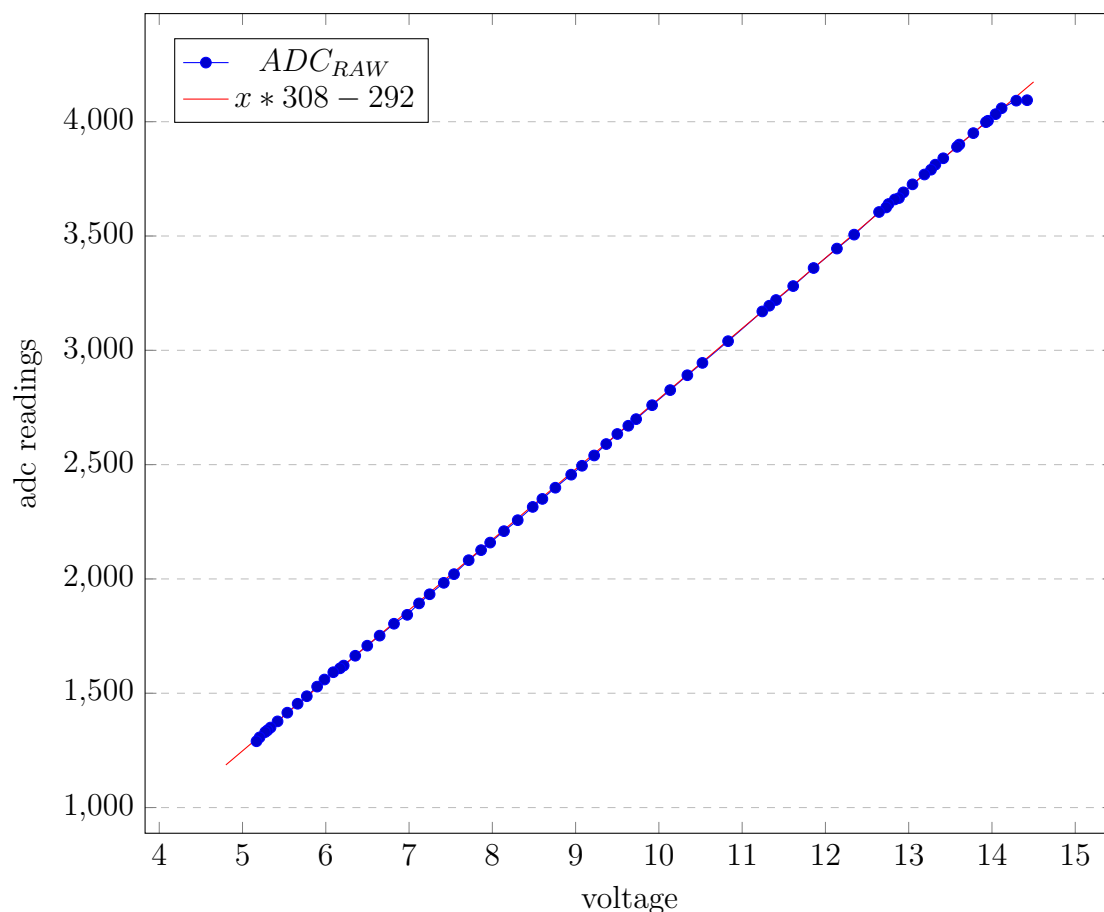
Po przekształceniu go tak aby wyliczyć napięcie przed dzielnikiem otrzymujemy:

$$V_{in} = \frac{1.250V \cdot (9.1k\Omega + 1k\Omega)}{1k\Omega} = 12.625V$$

Dla najniższego napięcia w mieszczącego się w zakresie efektywnego pomiaru (100mV) otrzymamy:

$$V_{in} = \frac{0.100V \cdot (9.1k\Omega + 1k\Omega)}{1k\Omega} = 1.01V$$

Oznacza to że w zakresie zasilania od 1.01V do 12.625V mamy dużą szansę uzyskać wynik pomiaru bliski rzeczywistości.



Rysunek 3.14: Pomiary danych ADC do napięcia zasilania

### 3.11.3 Wyznaczanie współczynnika konwersji

W celu przeliczenia wartości odczytywanych z przetwornika analogowy-cyfrowego na napięcie wyrażone w woltach, niezbędne jest wyznaczenie współczynnika konwersji.

Posiadając wiedzę na temat użytych rezystorów w dzielniku napięcia, jest możliwe wyliczenie tego współczynnika. Aczkolwiek, moje doświadczenie z projektowaniem układów pomiarowych podpowiada że teoria bardzo często mija się z praktyką i zazwyczaj kalibruję swoje układy wykonując serię pomiarów. Tak też zrobiłem w tym przypadku. Wyniki zostały przedstawione na wykresie ??

Uzyskane pomiary potwierdzają że prawie cały zakres przetwornika, nie licząc końców, jest liniowy. Pomiary zakończyłem przy około 5.1V ponieważ poniżej tej wartości stabilizator liniowy, użyty do obniżenia napięcia na procesorze, przestawał poprawnie działać, co kończyło się resetami jednostki centralnej.

Dzięki regresji liniowej byłem w stanie wyprowadzić równanie, które w łatwy sposób umożliwia przeliczanie odczytów z ADC na wartość napięcia.

$$y = \frac{ADC_{RAW} + 292}{398}$$

Za  $ADC_{RAW}$  wstawiamy wartość pomiaru i otrzymujemy napięcie wyrażone w woltach.

### 3.11.4 Rozdzielczość pomiaru

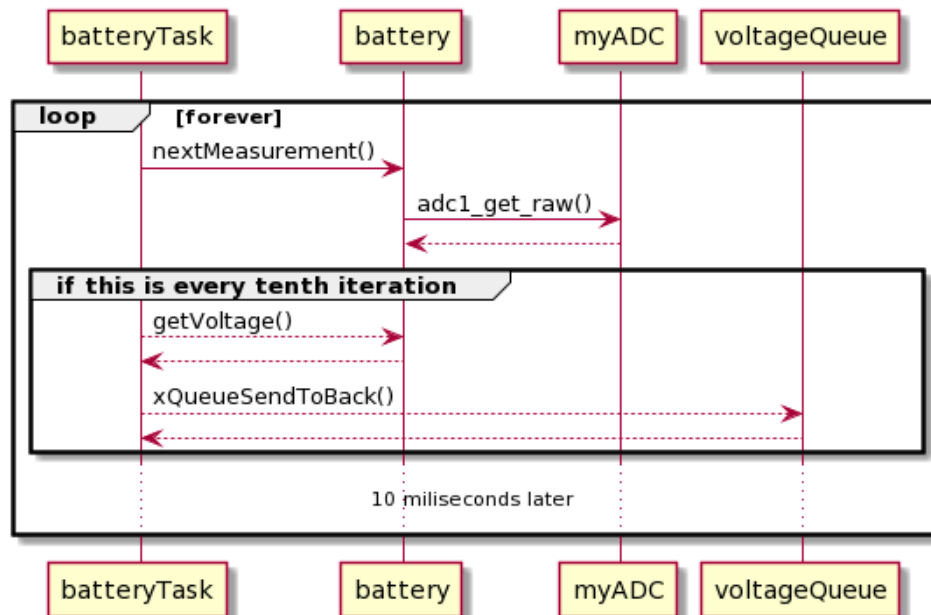
Posiadając już możliwość przeliczania surowych odczytów na napięcie, należałoby wyznaczyć rozdzielczość otrzymanych danych. Do obliczenia rozdzielczości napięciowej, czyli najmniejszego możliwego skoku zdolnego do zapisania przed przetwornik, musimy wykonać dwa pomiary. Następnie należy porównać różnice obliczonych napięć do różnicy surowych danych. Przy pomocy podanego wzoru:

$$\frac{V_1 - V_2}{ADC_1 - ADC_2} = \frac{10.136V - 6.171V}{2826 - 1609} = \frac{3.965V}{1217} = 0.0032V$$

### 3.11.5 Wykonywanie pomiarów

Wykonywanie pomiarów jest wykonywane na osobnym procesie. Dzięki temu można w łatwy sposób kontrolować częstotliwość mierzenia napięcia i wysyłania danych z ADC na MQTT. Pętla wyzwalająca odczyt wykonuje się raz na 20 milisekund. W każdej iteracji następuje pomiar, a odczytana wartość umieszczana jest w buforze. Co dziesiąta iteracja wylicza średnią z pomiarów, konwertuje wartość na wolty i wysyła dane na MQTT. Diagram aktywności umieszczony został w listingu ??.

Listing 3.11.2: Wyzwalanie pomiaru i przeliczanie wartości



Rysunek 3.15: Diagram aktywności dokonywania pomiarów

## 3.12 Łączenie z brokerem MQTT

Jedną z zalet wykorzystania oficjalnego frameworka wydawanego przez producenta jest gotowa implementacja warstw abstrakcji ułatwiających obsługę protokołu MQTT. Umożliwia ona łatwą konfigurację i wykorzystanie tego środka komunikacji za pomocą kilku wysokopoziomowych funkcji.

### 3.12.1 Konfiguracja

Pierwsze co musimy zrobić to wypełnić strukturę konfiguracyjną. Znajduje się w niej aż 165 elementów. Jednak dla prawidłowej pracy wystarczy wypełnić kilka podstawowych pól takich jak dane logowania użytkownika czy adres brokera. Część z nich służy jako wskaźniki do danych przychodzących, bufory, czy flagi niezbędne do funkcjonowania.

Kolejnym krokiem jest stworzenie struktury inicjalizacyjnej z pomocą poprzedniej struktury. Posłuży nam do tego specjalna funkcja która sprawdza poprawność wprowadzonych przez nas danych, a dane nie definiowane zostają zastąpione wartościami domyślnymi.

Następne musimy wybrać jakie zdarzenia chcemy rejestrować i gdzie mają one trafiać. W tym przypadku wybieramy pełną pulę dostępnych sygnałów. Zawartość funkcji obsługującej zdarzenia znajduje się w listingu ??.

Pozostaje już tylko uruchomić moduł MQTT przekazując powstałą konfigurację do funkcji startowej. Dane zostaną przesłane do nowego procesu, który zajmie się za nas obsługą komunikacji. Wszystkie niezbędne kroki oraz konfiguracja została przedstawiona na listingu ??.

Listing 3.12.1: Konfiguracja połączenia MQTT

### 3.12.2 Obsługa zdarzeń

Wykorzystana warstwa abstrakcji posiada system udostępniający nam możliwość odbierania sygnałów generowanych przez zdarzenia wynikające z pracy klienta MQTT. Skorzystamy z tej funkcjonalności, aby informować użytkownika o stanie komunikacji, a także aby odpowiednio reagować na zdarzenia. Została ona załączona w listingu ??.

Po otrzymaniu sygnału poprawnego nawiązania połączenia, zaczynamy subskrybować wszystkie tematy które przechowują potrzebne nam dane. Jest to przedstawione na listingu ??.

Listing 3.12.2: Subskrybowanie niezbędnych tematów

Następnym krokiem jest utworzenie nowego procesu przypisanego do rdzenia drugiego (numeracja jest od zera). Jego zadaniem jest transmisja danych z urządzenia do brokera. Utworzenie kolejnego zadania oraz sam fakt pomyślnego uzyskania połączenia obarczony jest komunikatem diagnostycznym.

Otrzymanie informacji o zakończeniu połączenia wiąże się z odwróceniem zmian dokonanych przez poprzedni sygnał. Po wysłaniu komunikatu na strumień wyjścia, dokonujemy zabicia procesu wysyłającego dane, po uprzednim sprawdzeniu czy taki jeszcze istnieje.

Jest to zabezpieczenie przez sytuacją, gdy istnieje więcej niż jeden taki proces. Następnie po odczekaniu jednej sekundy, wyzwalana jest próba ponownego łączenia.

Ostatnim ważnym sygnałem jest informacja o zmianie wartości subskrybowanego kanału. Jest ona obrazu przekazywana do stworzonej przez siebie funkcji. Znajduje się ona w listingu ???. Jej jedynym zadaniem jest dodawanie odebranych wartości do odpowiednich kolejek priorytetowych w zależności od tematu wiadomości.

#### Listing 3.12.3: Odbieranie danych

Pozostałe sygnały służą czysto w celach diagnostycznych. Z tego też powodu nie wykonują one żadnego kodu poza raportowaniem zdarzenia poprzez komunikat na standardowym strumieniu wyjścia.

#### Listing 3.12.4: Obsługa sygnałów

### 3.12.3 Wysyłanie danych

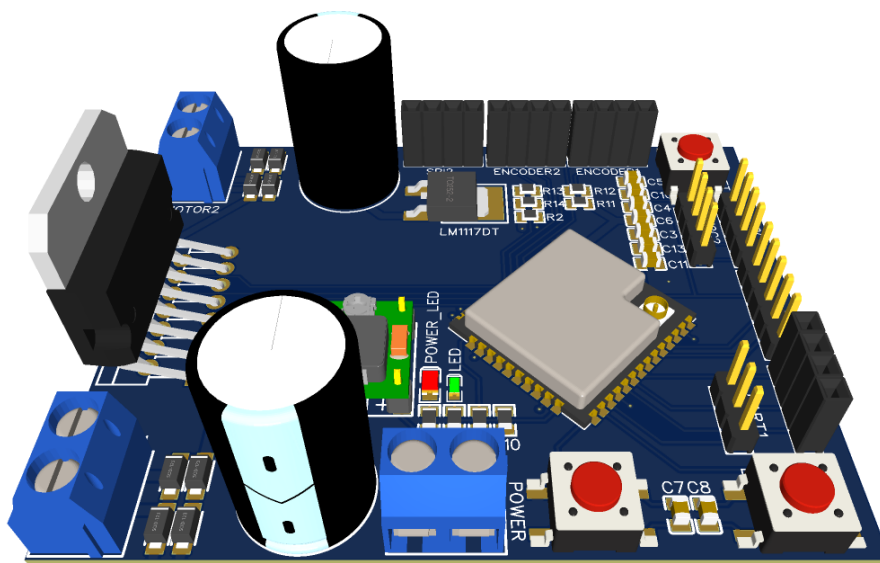
Transmisja informacji odbywa się na osobnym procesie niż reszta modułu MQTT. Logika przekazywania danych jest względnie prosta. Wystarczy odebrać wiadomość z kolejki priorytetowej, a następnie przekazać ją do wysłania z pomocą odpowiedniej funkcji. Na listingu ??? przedstawione jest wysyłanie wiadomości na wszystkich trzech tematach zarządzanych przez mikrokontroler.

#### Listing 3.12.5: Wysyłanie danych

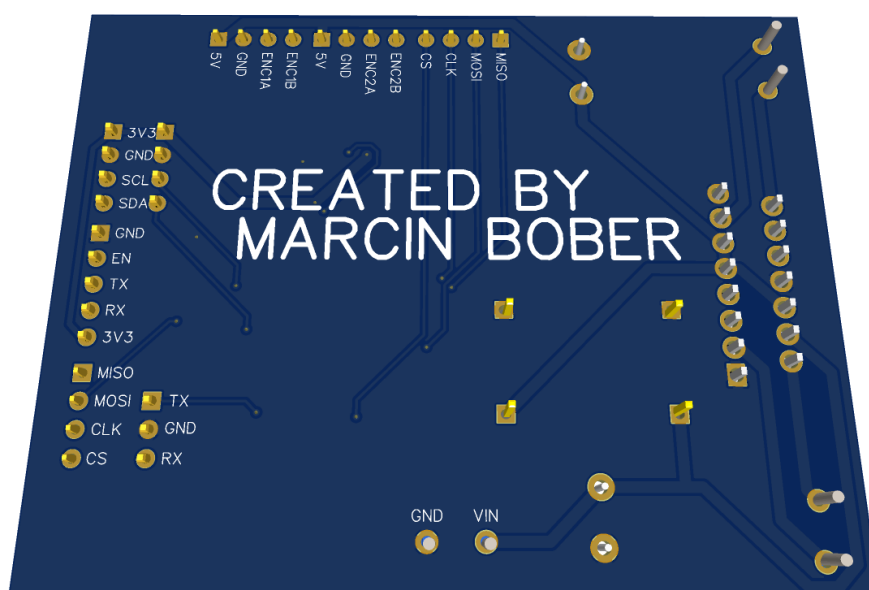
## 3.13 Gotowe urządzenie

### 3.13.1 Symulacja

Symulacje gotowego urządzenia zostały przedstawione na rysunkach ?? oraz ??.



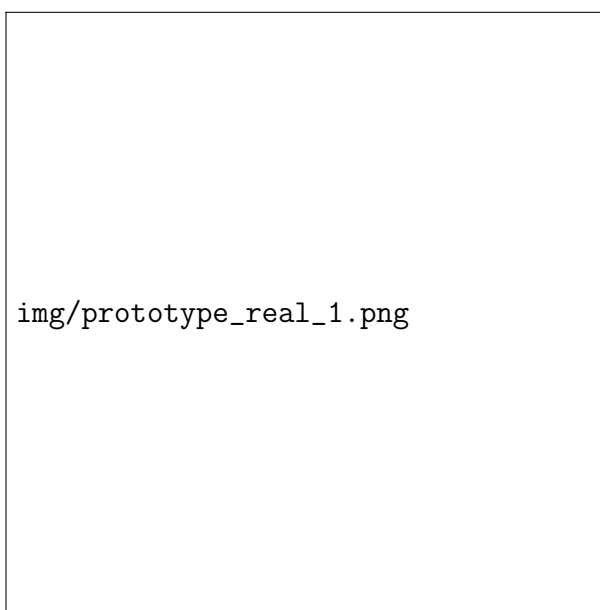
Rysunek 3.16: Symulacja gotowego urządzenia (wierzch)



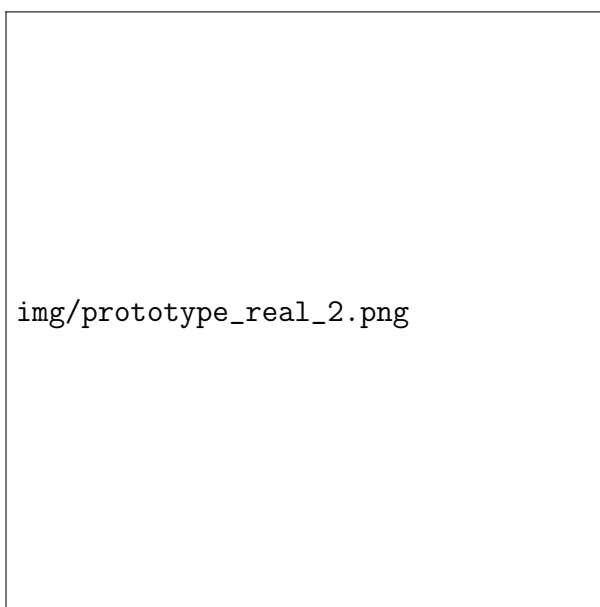
Rysunek 3.17: Symulacja gotowego urządzenia (spód)

### 3.13.2 Prototyp

Zdjęcia gotowego urządzenia zostały przedstawione na rysunkach ?? oraz ??.



Rysunek 3.18: Zdjęcie gotowego urządzenia (wierzch)



Rysunek 3.19: Zdjęcie gotowego urządzenia (spód)



# Rozdział 4

## Szczegółowy opis brokera MQTT

### 4.1 Opis

Broker MQTT jest to pewnego rodzaju serwer który zbiera wiadomości od wszystkich klientów, a następnie przekierowuje je klientom docelowym zgodnie z ich subskrypcjami. Jego zadaniem jest więc być pośrednikiem w wymianie informacji pomiędzy klientami oraz dbanie o odpowiedni przepływ wiadomości.

### 4.2 Protokół

MQTT jest skrótem od MQ Telemetry Transport. Jego głównym założeniem jest nie-samowita prostota implementacji jednocześnie zachowując wybitnie skromne wymagania sprzętowe. Jego zalety szybko zostały zauważone czego najlepszym przykładem jest fakt że znalazł on szerokie zastosowanie w takich branżach jak Automotive, logistyka, czy produkcja. Jednak najczęściej kojarzony jest on z tematami Internetu Rzeczy oraz Inteligentnych Domów. Świetnie nadaje się on do łączenia w jedną sieć małych energooszczędnych urządzeń.

Wiadomości są zorganizowane w hierarchii tematów. Każda z wiadomości przypisana jest do jakiegoś tematu. W przypadku, gdy temat nie istnieje, zostaje automatycznie utworzony wraz z napływem pierwszej wiadomości. Broker po odebraniu wiadomości informuje wszystkich klientów, którzy zapisali się do listy subskrypcji danego tematu. W przypadku, gdy dany temat już istnieje następuje nadpisanie jego wartości nowymi danymi. Dzieje się tak dlatego że broker przechowuje tylko i wyłącznie ostatnią wiadomość z każdego tematu.

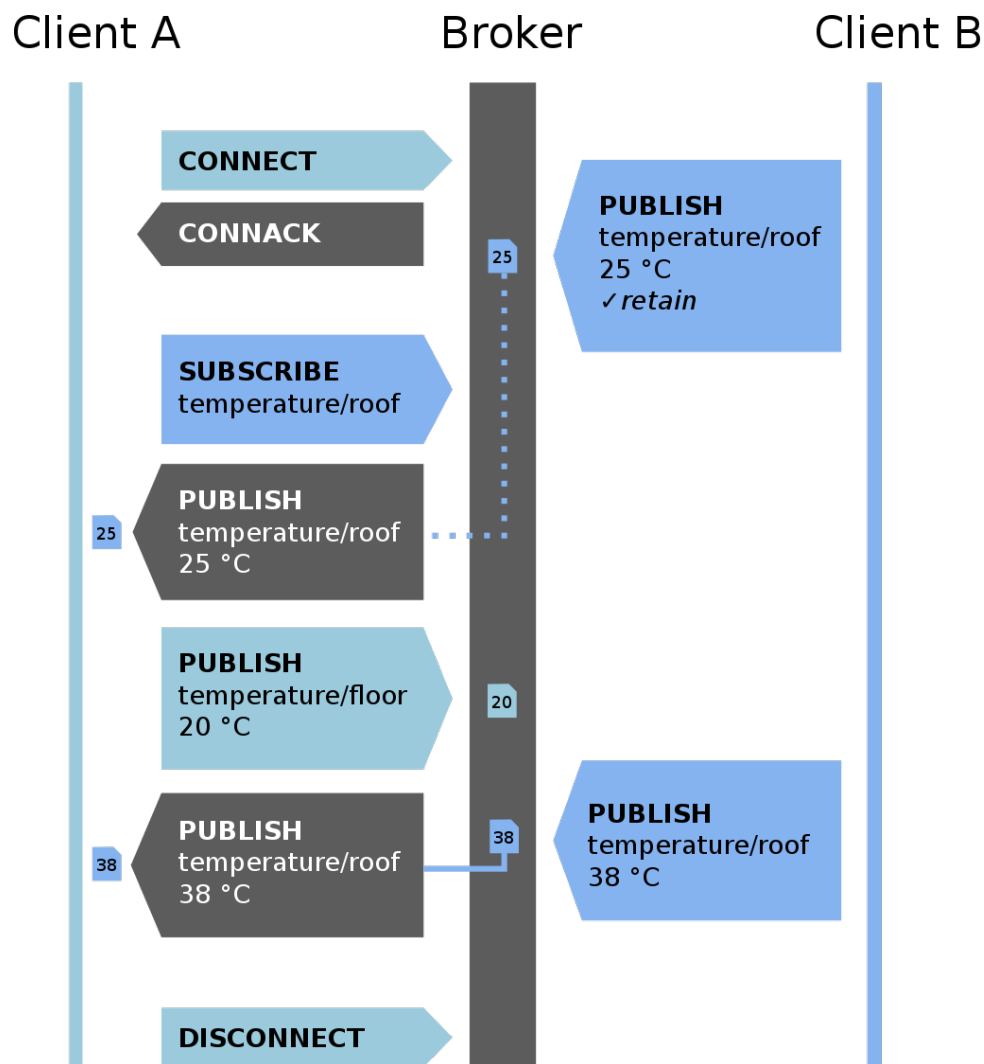
Ciekawą opcją jest ustawienie tak zwanego testamentu. Jest to wiadomość publikowana, gdy klient który sobie taką opcję zażyczył, nieoczekiwanie utraci połączenie z brokerem.

Klienci podłączeni do jednego brokera nie znają się nawzajem. Nie są bowiem udostępniane żadne dane bezpośrednio pomiędzy klientami. Przedstawia to schemat umieszczony na rys. ???. Wszystkie przekazywane wiadomości muszą przejść przez broker.

### 4.3 Typy wiadomości

Istnieje obsługiwanych 14 typów wiadomości.

- CONNECT - Nawiązanie połączenia,



Rysunek 4.1: Schemat działania protokołu MQTT

- **CONNACK** - Potwierdzenie nawiązania połączenia,
- **PUBLISH** - Publikacja wiadomości,
- **PUBACK** - Potwierdzenie publikacji wiadomości
- **PUBREC** - Potwierdzenie otrzymania wiadomości,
- **PUBREL** - Potwierdzenie wysłania wiadomości,
- **PUBCOMP** - Potwierdzenie końca publikowania wiadomości,
- **SUBSCRIBE** - Subskrypcja tematu,
- **SUBACK** - Potwierdzenie subskrypcji,
- **UNSUBSCRIBE** - Anulowanie subskrypcji,
- **UNSUBACK** - Potwierdzenie anulowania subskrypcji,
- **PINGREQ** - Żądanie PINGU,

- PINGRESP - Odpowiedź na żądanie PINGU,
- DISCONNECT - Zakończenie połączenia.

## 4.4 Instalacja i konfiguracja

### 4.4.1 Tworzenie kontenera

Instalacja brokera Mosquitto w kontenerze jest niezwykle prosta. Do uruchomienia instancji brokera wystarczy jedna komenda ?? wpisana w konsoli. W przypadku, gdy Docker nie znajdzie lokalnie danego kontenera, automatycznie pobierze on niezbędne pliki, rozpakuje je, a następnie kontynuuje uruchomienie. Jedynym wymogiem jest poprawnie przeprowadzona instalacja programu Docker.

Listing 4.4.1: Utworzenie instancji brokera MQTT w kontenerze

Do kompleksowego zrozumienia użytego polecenia warto pochylić się nad sensem wykorzystanych opcji:

- *run* uruchamia nowy kontener.
- *-it* włącza interaktywność kontenera, przekierowuje standardowy strumień wejściowy do kontenera.
- *-name* przypisuje kontenerowi przyjazną nazwę.
- *-p* to opcja pozwalająca przekierować port poza kontener. Domyślny port wykorzystywany przy komunikacji MQTT to 1883.
- *-v* montowanie woluminu. W tym przypadku montowany jest on w folderze użytkownika. Taka opcja znacząco ułatwia ingerencję w pliki konfiguracyjne.

Na końcu polecenia znajduje się nazwa obrazu z biblioteki, który ma zostać uruchomiony.

### 4.4.2 Konfiguracja

Po pomyślnej instalacji wymagana jest minimalna konfiguracja programu. Plik "mosquitto.conf" jest głównym plikiem konfiguracyjnym, w którym należy zawrzeć opcje wymienione w listingu ?. Przede wszystkim należy ustawić wykorzystywany port na identyczny z podanym przy tworzeniu kontenera. Kolejno należy wybrać możliwość zapisywania stanów oraz ścieżkę zapisu. Ważnym aspektem jest także wymuszenie autoryzacji podczas połączenia wraz ze wskazaniem na listę dopuszczanych klientów i haseł.

Listing 4.4.2: Minimalna konfiguracja brokera

### 4.4.3 Dodawanie użytkowników

Ostatnim krokiem w przygotowywaniu brokera do przyjęcia klientów jest zdefiniowanie listy akceptowanych loginów wraz z przypisanymi im hasłami. Najprostszym sposobem jest wykorzystanie dedykowanej temu celu komendy widocznej w listingu ??.

#### Listing 4.4.3: Dodawanie pierwszego użytkownika

Tworzy ona nowy plik z hasłami o nazwie *passwordfile* jednocześnie wyzwalając inicjalizację pierwszego użytkownika o loginie *user*. Po wykonaniu polecenia zostaje wystosowane również dwukrotne zapytanie o hasło dla nowego konta.

# Rozdział 5

## Szczegółowy opis aplikacji dostępowej

### 5.1 Grupa docelowa

Aplikacja okienkowa została stworzona, aby zwiększyć dostępność systemu dla osób nie wprawionych w tematy związane z protokołem MQTT. Istnieje pełen przekrój uniwersalnych aplikacji zdolnych do komunikacji za pośrednictwem owego protokołu. Część z nich świetnie naddawałaby się do kooperowania z resztą przygotowanego systemu. Z drugiej strony wszystkie tego typu aplikacje są zazwyczaj nad wyraz rozbudowane oraz wymagają od użytkownika pewnej specyficznej wiedzy i doświadczenia. Nie można wykluczać że z tego rozwiązania miałyby ochotę skorzystać osoba o niższej świadomości technicznej lub chociażby niewdrożona w tematy związane z tą technologią. Z tego też powodu w ramach projektu powstała aplikacja dedykowana opisywanemu systemowi. Gwarantuje ona kompatybilność z resztą elementów zawartych w projekcie jednocześnie posiadając opcje w pełni wykorzystujące wszystkie funkcjonalności systemu. Należy także dodać że projektowana była z myślą o prostocie, intuicyjności w obsłudze i minimalizmie.

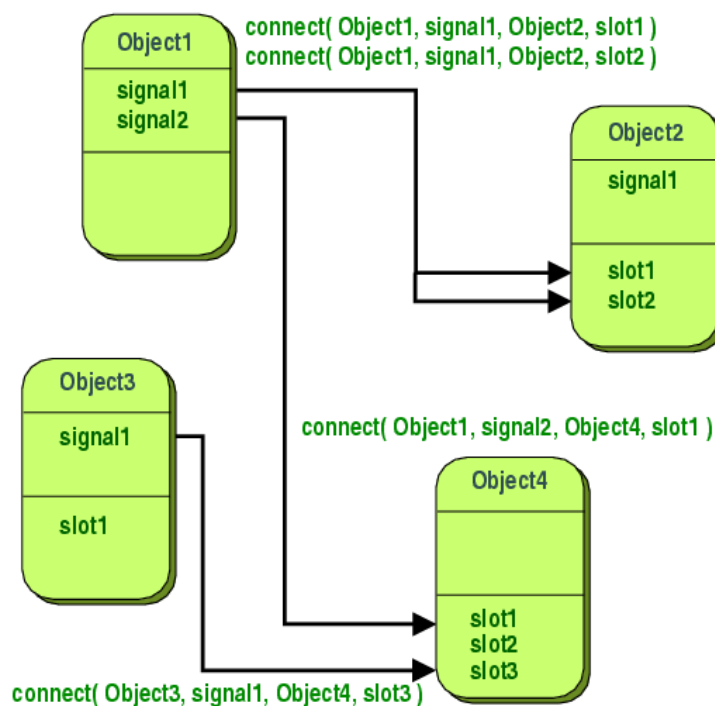
### 5.2 Wykorzystana technologia

Sprostanie założeniom projektu jest nietrywialnym, a wręcz niedorzecznie trudnym zadaniem bazując jedynie na standardowych bibliotekach. Wykonanie tak zaawansowanej aplikacji w krótkim terminie było jednoznaczne z wykorzystaniem wysokopoziomowego frameworka, który znacząco uprościłby rozwój oprogramowania. W tym celu został użyty nowoczesny framework QT w wersji otwarto źródłowej. Jest to niezwykle rozbudowane narzędzie do tworzenia aplikacji okienkowych. Wspiera ono pełen przekrój systemów operacyjnych oraz architektur sprzętowych dzięki czemu program opierający się o tą technologię może być z powodzeniem przenoszony na różne urządzenia. Zdecydowanie najciekawszym aspektem tego oprogramowania jest niecodzienny system sygnałów i slotów. Według wielu początkujących programistów jest on nieintuicyjny. Jednakże wraz z korzystaniem z tego rozwiązania i towarzyszącym temu rosnącym doświadczeniem, pogląd ten jest niejednokrotnie rewidowany.

Poniżej przedstawione jest kilka fragmentów kodu użytego przy budowie aplikacji dostępowej wraz z opisami objaśniającymi wszystkie znajdujące się w nim zawilości.

### 5.3 System sygnałów i slotów

Wprowadzenie tego rodzaju funkcjonalności rozegrało niebagatelne znaczenie w popularności frameworka QT. W przygotowanej na potrzeby opisywanego projektu aplikacji, system sygnałów i slotów pełni kluczową rolę w jej funkcjonowaniu. Na listingu ?? został umieszczony fragment programu odpowiadającego za przypisanie sygnałów produkowanych przez liczne elementy interfejsu użytkownika do specjalnie przygotowanych na tę potrzeby funkcji. Nie trudno sobie wyobrazić jak wiele pracy jest w stanie oszczędzić proste łączenie sygnałów ze slotami w porównaniu ze żmudną implementacją karkołomnych rozwiązań na własną rękę. Pozwala to także na omijanie sytuacji, w której nie jeden programista sięgnąłby po rozwiązanie jakim jest wielowątkowość. Sposób działania tego rozwiązania przedstawiony jest na rysunku ??.



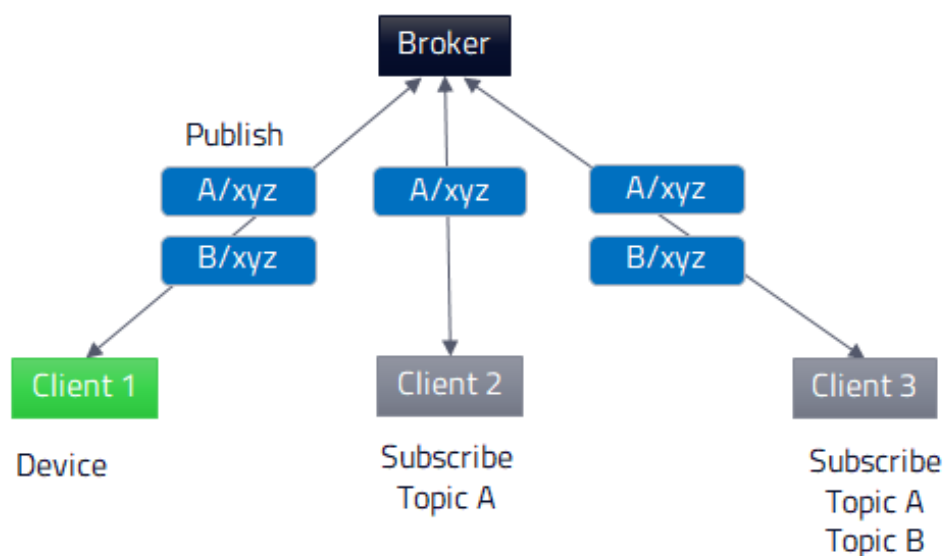
Rysunek 5.1: System sygnałów i slotów

Listing 5.3.1: Użycie systemu sygnałów i slotów

## 5.4 Implementacja komunikacji

### 5.4.1 Łączenie z brokerem

Podobnie jak w framework'u użytym na poczet przygotowania wsadu do sterownika, tutaj również jest zdefiniowana fenomenalnie prosta w obsłudze abstrakcja. Niemniej jednak w celu utrzymania wysokiej czytelności produkowanego kodu została przygotowana nowa autorska klasa. Dziedziczy ona po klasie zawartej w module QtMqtt. Jej celem jest rozszerzenie klasy bazowej o dodatkową funkcjonalność, która sprawi że inicjalizacja i obsługa połączenia będą jeszcze wygodniejsze. Jako argumenty wywołania należy podać adres i port serwera, a następnie nazwę użytkownika oraz hasło. Cała reszta procesu została zgrabnie ukryta pod warstwą abstrakcji. Omawiany kod przedstawiony został w listingu ??.



Rysunek 5.2: Schemat działania modułu Qt MQTT

Listing 5.4.1: Nawiązanie połączenia z brokerem

### 5.4.2 Obsługa zdarzeń

Kolejną analogią znaną z frameworka ESP-IDF jest obsługa zdarzeń przychodzących z modułu MQTT. Wysyła on sygnał połączony z funkcją widoczną w listingu ?? zdający raport o stanie połączenia. Jest on wykorzystywany do aktywowania interfejsu użytkownika, który domyślnie przyjmuje status nieaktywnego. Ponadto jest to wyzwalacz do subskrybowania uprzednio zdefiniowanych tematów.

Listing 5.4.2: Obsługa zdarzeń

### 5.4.3 Subskrybowanie tematów

Subskrybowanie tematów jest dość długą funkcją. Wbrew pozorom nie jest ona skomplikowana. Jej zadaniem jest zasubskrybować jedynie cztery temat. Z tego też powodu

wszystko jest powielone czterokrotnie. To właśnie wpływa w znaczący sposób na jej objętość. Dla każdego tematu wywoływana jest metoda na obiekcie MQTT, która przyjmuje temat oraz parametr QoS.

QoS z angielskiego "Quality of Service" określa poziom istotności wiadomości. W tym przypadku użyta została wartość 0. Oznacza to że nie ma gwarancji otrzymania wiadomości. Taka kolej rzeczy nie wydaje się być z żaden sposób problematyczna ze względu na cykliczność z jaką otrzymywane są ramki. Zgubienie jednej wartości nie musi być istotne w momencie, gdy są one transmitowane z ponadprzeciętnie dużą częstotliwością. W takim systemie pierwszoplanowa jest wydajność i przepustowość. Pozostałe wartości które są możliwe do ustawienia to 1 i 2. Pierwsza z nich sprawia że mamy pewność otrzymania wiadomości, natomiast druga gwarantuje jednokrotność tego zdarzenia [?].

Produktem wywołania tej metody będzie obiekt typu `QMqttSubscription`, który następnie należy przy pomocy systemu sygnałów i slotów, połączyć z odpowiednią funkcją. Funkcja ta jedynie skleja otrzymaną wartość z uprzednio przygotowanym ciągiem znaków i bezpośrednio wyświetla ją w interfejsie użytkownika. Przykład takiej funkcji dostępny jest w listingu ??.

Na samym końcu znajduje się jeszcze wywołanie funkcji ustawiającej wartości domyślne dla wyżej wymienionych tematów, tak aby można było je umieścić w interfejsie. Cała ta procedura powtórzona jest cztery razy dla czterech różnych tematów. Odnosi się do niej listing ??.

Listing 5.4.3: Przetwarzanie odebranych danych

Listing 5.4.4: Subskrybowanie tematów

#### 5.4.4 Wysyłanie danych

Wysyłanie danych polega na odbieraniu wartości prosto z interfejsu i przekazywaniu ich do metody obiektu MQTT. Połączone są więc konkretne sygnały elementów interfejsu z daną lambda. W roli przykładu, sygnał suwaka ustawiającego wartość zadaną łączymy z lambda w której wyłuskujemy wartość suwaka. Następnie jest ona zapisywana jako tablica bajtów i publikowana. Dodatkowo jest wyświetlana w postaci liczbowej w interfejsie. Przykład implementacji znajduje się w listingu ??.

Listing 5.4.5: Wysyłanie danych



## 5.5 Wykresy

### 5.5.1 Definicja klasy

Jedną z najważniejszych funkcjonalności aplikacji jest estetyczna prezentacja danych pobranych ze sterownika. Podjęta została decyzja o wykorzystaniu w tym celu prostych wykresów. Framework QT zapewnia moduł QtCharts, który jest zestawem prostych w użyciu komponentów graficznych niezbędnych do stworzenia estetycznych wykresów. Zostało to uskutecznione poprzez wykonanie nowej klasy dziedziczącej po klasie QChart dostępnej w opisywanym module. W swoim konstruktorze tworzy ona wcześniej zdefiniowany wykres z kilkoma seriami danych. Oprócz tego stworzona klasa posiada zdefiniowane metody do czyszczenia zawartości wykresu, dodawania nowych punktów oraz ukrywania nieużywanych serii danych. Kod jest dostępnym w listingu ??.

Listing 5.5.1: Definicja klasy wykresów

### 5.5.2 Dodawanie punktów do wykresu

Implementacja rysowania wykresów opublikowana przez twórców frameworka QT pozostawia wiele do życzenia. W szczególności kwestia wydajności nie została należycie rozpatrzona czego skutkiem jest niewymiernie wysokie zużycie zasobów komputera podczas renderowania tych obiektów. Zastosowana strategia przeciwdziałająca temu zjawisku zakłada skończoną liczbę punktów obecnych jednocześnie na wykresie. Z tego też powodu został zaimplementowany trywialny algorytm pozbywający się nadmiarowych obiektów.

Kod przedstawiony w listingu ?? ma za zadanie włączać kolejne dane do wykresu. Upřednio musi zostać wyznaczona pozycja punktu na osi X. Następnie dla każdej z 3 serii sprawdzana jest ilość znajdujących się na niej pomiarów. W razie przekroczenia ustalonego limitu, nadmiar elementów jest usuwany z początku kolejki, aby następnie dodać na jej końcu dodać nowy punkt. Przed zakończeniem tej metody wykonywana jest kalkulacja położenia najbardziej oddalonych od siebie obiektów, aby móc dopasować zakresy wyświetlania wykresu.

Listing 5.5.2: Dodawanie danych do wykresu

### 5.5.3 Czyszczenie wykresu

Usuwanie danych w wykresów nie należy do najbardziej skomplikowanych. Wystarczy jedynie na każdej z serii wywołać metodę pozbywającą się wszystkich punktów z bufora. Kod znajduje się w listingu ??.

Listing 5.5.3: Usuwanie wszystkich danych z wykresu

### 5.5.4 Ukrywanie serii

Metoda zmiany widoczności danej serii jest jedynie makrem korzystającym z odziedziczonych metod. Dostępna jest pod listingiem ??.

Listing 5.5.4: Zmiana widoczności serii

## 5.6 Abstrakcja silnika

### 5.6.1 Definicja klasy silnika

W celu ujednoliconego zarządzania danymi odbieranymi i wysyłanymi do sterownika, została utworzona specjalna klasa, która przechowuje wartości z nim związane. Zdecydowanie zwiększa to czytelność kodu ponieważ zmienne dotyczące na przykład prędkości obrotowej nie są porzucane po całym programie. Co więcej, zostały stworzone odpowiednie metody które niwelują niebezpieczeństwo przypadkowego nadpisania jakiejś zmiennej. Implementacja została przedstawiona w listingu ???. Przykładowe użycie metody można zobaczyć w listingu ???.

Listing 5.6.1: Klasa abstrakcji silnika

### 5.6.2 Przeliczanie wartości obrotów

Konwersja liczby otrzymanej impulsów na ilość wykonanych obrotów wymaga od nas posiadania odpowiednich informacji na temat enkodera. Częstotliwości pomiaru impulsów przez sterownik to 100Hz. Oznacza to czas pomiędzy pomiarami równy:

$$T = \frac{1}{f} = \frac{1}{100Hz} = 10ms$$

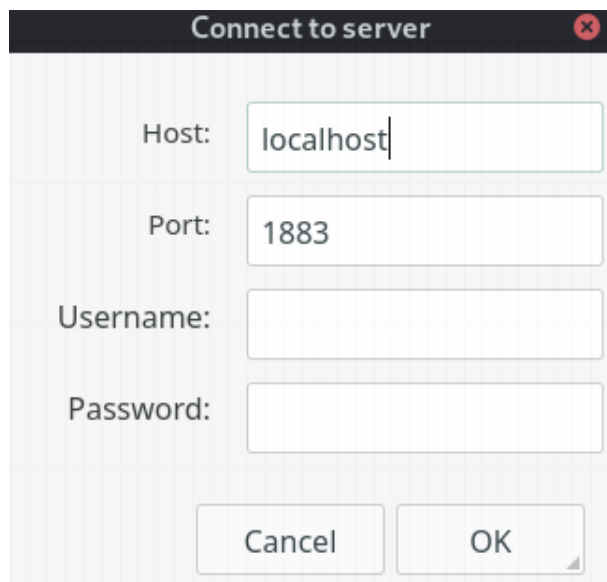
Wynika z tego że ilość obrotów należy pomnożyć stukrotnie, aby uzyskać ilość impulsów na sekundę. Następne mnożenie przez 60 doprowadzi do otrzymania ilości impulsów w okresie jednej minuty. Nie można zapomnieć że wartość impulsów jest czterokrotnie większa ze względu na konfigurację liczników (patrz ???). Z tego też powodu niezbędne jest podzielenie wyniku przez 4. Ostatecznie całość należy podzielić przez ilość impulsów przypadających na jeden obrót wałka wychodzącego z przedkładani (224.4PPR/RPM). W ten sposób możliwe jest uzyskanie odpowiedniego przelicznika.

Listing 5.6.2: Przeliczanie impulsów na obroty

## 5.7 Interfejs użytkownika

### 5.7.1 Górna belka

W górnej części aplikacji znajduje się menu kontekstowe. Pozwala ono na wywołanie okna inicjalizującego połączenie z serwerem (patrz ??). Do nawiązania pomyślnego połączenia wymagane jest umieszczenie poprawnego adresu oraz portu brokera MQTT. W przypadku dezaktywowanej opcji przyjmowania anonimowych użytkowników wymagane jest także podanie ważnych danych autoryzacyjnych.



The image shows a 'Connect to server' dialog box. It has a title bar with a red close button. The dialog contains four input fields: 'Host' with the text 'localhost', 'Port' with the text '1883', 'Username' (empty), and 'Password' (empty). At the bottom are 'Cancel' and 'OK' buttons.

Rysunek 5.3: Okno nawiązywania połączenia

### 5.7.2 Panel parametrów

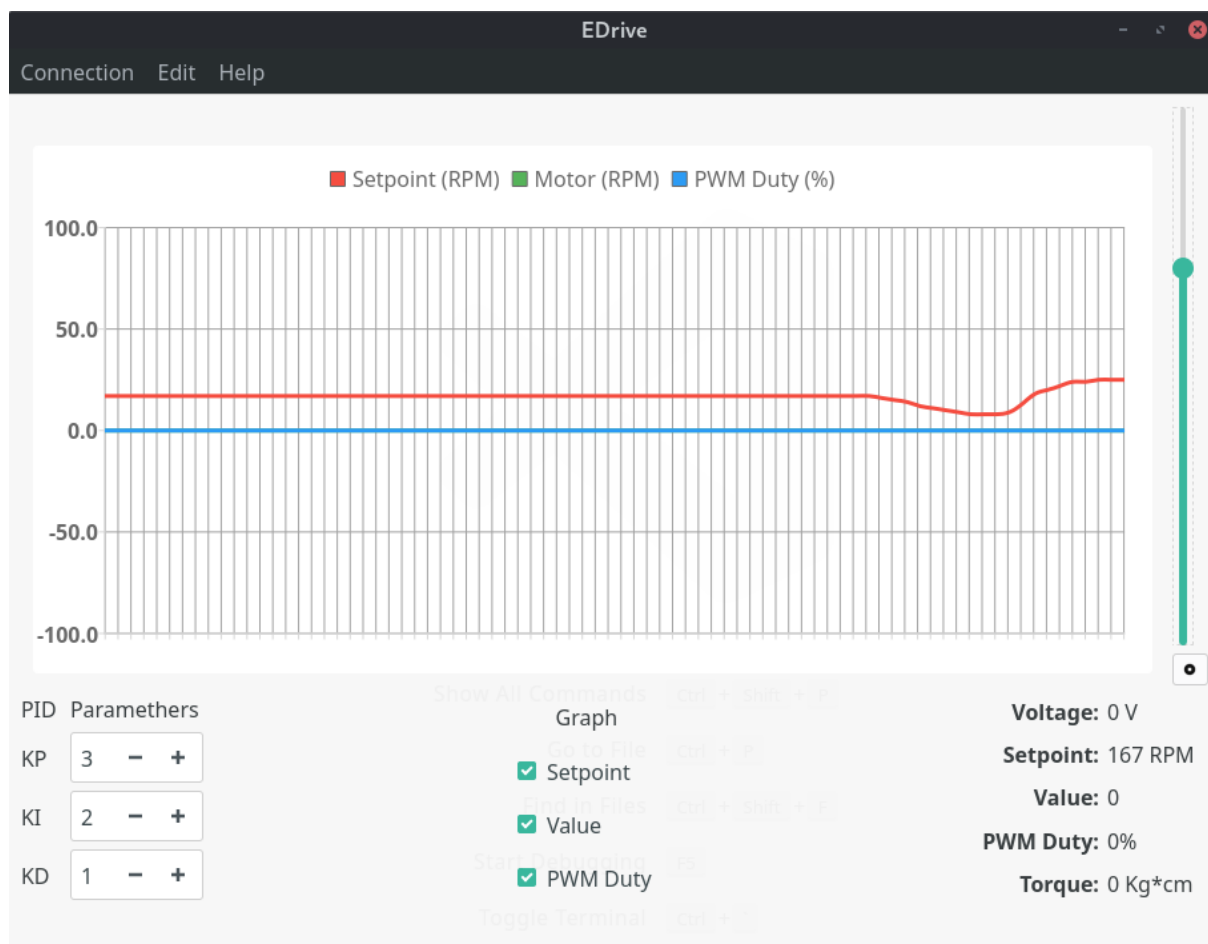
Dolna część okna podzielona jest na trzy części.

- Panel parametrów regulatora.
- Panel widoczności wykresów.
- Panel z aktualnymi danymi.

Pierwsza z nich udostępnia interfejs do dynamicznej zmiany nastaw regulatora

### 5.7.3 Graf

Interfejs zawiera wszystkie elementy niezbędne do wygodnej obsługi urządzenia.



Rysunek 5.4: Interfejs użytkownika

# Rozdział 6

## Instrukcja obsługi

# Rozdział 7

## Podsumowane

# Literatura

- [1] Qt Development Frameworks: Dokumentacja biblioteki Qt  
<https://doc.qt.io/qt-5/>  
Dostęp 14.09.2021
- [2] STMicroelectronics: Dokumentacja L298H  
[https://www.sparkfun.com/datasheets/Robotics/L298\\_H\\_Bridge.pdf](https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf)  
Dostęp 14.09.2021
- [3] Espressif Systems: Dokumentacja mikrokontrolera ESP32 [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)  
Dostęp 14.09.2021
- [4] Espressif Systems: Dokumentacja modułu ESP32WROOM32UE [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)  
Dostęp 14.09.2021
- [5] OASIS: Dokumentacja standardu MQTT 5.0  
<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf>  
Dostęp 14.09.2021
- [6] V.P. Eloranta, J. Koskinen, M. Leppanen, V. Reijonen: Designing Distributed Control Systems: A Pattern Language Approach, Wiley, 2014.
- [7] Kenneth Flamm: Measuring Moore's Law: Evidence from Price, Cost, and Quality Indexes  
<https://www.imf.org/-/media/Files/Conferences/2017-stats-forum/session-6-kenneth-flamm.ashx>  
Dostęp 15.09.2021
- [8] Espressif Systems: Dokumentacja ESP-IDF  
<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>  
Dostęp 15.09.2021
- [9] Amazon Web Services: Dokumentacja FreeRTOS  
<https://www.freertos.org/a00106.html>  
Dostęp 15.09.2021
- [10] Wikipedia: Szereg wartości  
[https://pl.wikipedia.org/wiki/Szereg\\_warto%C5%9Bci](https://pl.wikipedia.org/wiki/Szereg_warto%C5%9Bci)  
Dostęp 17.09.2021
- [11] Wikipedia: Dzielnik rezystorowy  
[https://en.wikipedia.org/wiki/Voltage\\_divider](https://en.wikipedia.org/wiki/Voltage_divider)  
Dostęp 17.09.2021

- [12] Robert W. Doran: The Gray Code  
[http://www.jucs.org/jucs\\_13\\_11/the\\_gray\\_code/jucs\\_13\\_11\\_1573\\_1597\\_doran.pdf](http://www.jucs.org/jucs_13_11/the_gray_code/jucs_13_11_1573_1597_doran.pdf)  
Dostęp 27.09.2021
- [13] Siemens: Enkodery inkrementalne  
<https://publikacje.siemens-info.com/pdf/56/Motion%20Control%20-%20Uk%C5%82ad%20pomiarowy.pdf>  
Dostęp 17.09.2021
- [14] Schemat działania modułu Qt MQTT  
<https://doc.qt.io/QtMQTT/mqtt-overview.html>  
Dostęp 29.09.2021
- [15] Topologia procesora z użyciem FreeRTOS  
<https://www.freertos.org/2020/02/simple-multicore-core-to-core-communication-using.html>  
Dostęp 29.09.2021
- [16] Schemat podłączenia mostka H  
<https://forbot.pl/forum/topic/16-teoria-mostek-h-h-bridge-kompendium-dla-robotyka>  
Dostęp 29.09.2021
- [17] Schemat wykorzystanego silnika  
<https://www.dfrobot.com/product-1619.html>  
Dostęp 29.09.2021
- [18] Schemat wykorzystanego mostka  
[https://www.sparkfun.com/datasheets/Robotics/L298\\_H\\_Bridge.pdf](https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf)  
Dostęp 29.09.2021
- [19] Schemat działania protokołu MQTT  
<https://en.wikipedia.org/wiki/MQTT>  
Dostęp 29.09.2021
- [20] System sygnałów i slotów  
<https://doc.qt.io/qt-5/signalsandslots.html>  
Dostęp 29.09.2021
- [21] Wykorzystana jednostka centralna  
<https://www.digikey.pl/product-detail/pl/espressif-systems/ESP32-WROOM-32U-16MB/1904-1028-1-ND/9381737>  
Dostęp 29.09.2021



# Spis rysunków

# Spis listingów