AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Informatyki, Elektroniki i Telekomunikacji

INSTYTUT ELEKTRONIKI

Projekt dyplomowy

Sterownik procesowy silnika BLDC z funkcjonalnością IoT
BLDC engine process controller with IoT functionality

Autor: *Hubert Kolano*
Kierunek studiów: *Elektronika i Telekomunikacja*
Opiekun pracy: *dr. Inż. Dominik Grochala*

Kraków, *2024*

Spis treści

1. Wstęp

Celem pracy jest zaprojektowanie i zaprogramowanie urządzenia, które będzie służyło jako sterownik silnika BLDC (Brushless Direct Current motor) używanego w pracowni laboratoryjnej. Cały proces projektowania urządzenia oparty jest na podstawie ustalonych wcześniej od niego wymagań. Silnik jest sterowany normalnie za pomocą potencjometru, sterownik ma za zadanie podnieść precyzje tego rozwiązania, umożliwić pomiar jego obrotów i sterować za pomocą zadania mu żądanej wartości obrotów. Sterownik ma również za zadanie umożliwić ustawienie parametrów pracy silnika poprzez zewnętrzne interfejsy, w tym bezprzewodowe nadające urządzeniu funkcjonalność IoT (Internet of Things).

Przed przystąpieniem do projektowania sterownika, zostanie opisany konkretny model silnika BLDC, którego pracę ma regulować sterownik. Zostanie przedstawiony koncept wykonania sterownika i jakie konkretnie musi on spełnić wymagania, aby ułatwić pracę w laboratorium.

Praca będzie zawierała opisany proces tworzenia elektronicznej płytki, gdzie opisane zostaną najważniejsze decyzje związane z doborem komponentów, projekt w programie CAD (Computer-Aided Design) oraz jej manualny montaż.

Zostanie również opisany proces tworzenia oprogramowania, wraz z przygotowaniem wcześniej architektury działania programu i modularnym tworzeniem funkcji na jej podstawie.

Na wykonanym sterowniku wykonane zostaną kalibracje i testy potwierdzające sprawność jego działania.

Na koniec przedstawiona będzie retrospektywa z prowadzenia projektu, możliwy dalszy rozwój prac nad sterownikiem i możliwych poprawek.

2. Założenia projektowe

W tym rozdziale przedstawione zostaną podstawowe założenia projektowe związane z realizacją sterownika procesowego silnika BLDC z funkcjonalnością IoT. Na wstępie określone zostaną cele oraz wymagania techniczne, które będą spełniane w ramach projektu. Następnie zostaną opisane szczegółowe specyfikacje silnika BLDC, do którego tworzony jest sterownik, wraz z kluczowymi parametrami, mającymi wpływ na wybór komponentów i algorytmów sterowania. Na końcu omówiony zostanie koncept projektu, obejmujący wybór architektury systemu, zastosowane technologie oraz opis funkcjonalności, jakie ma realizować sterownik.

2.1. Cel projektu

Celem projektu jest zaprojektowanie, zaprogramowanie i przetestowanie sterownika elektrycznego silnika BLDC. Sterownik ma sterować prędkością obrotową silnika, przy czym ma on spełniać wszystkie wymagane od niego funkcjonalności, wynikające z potrzeb laboratorium jak i samej specyfiki silnika BLDC do którego jest tworzony. Sterownik ma zapewnić stabilną i efektywną pracę silnika i zapewnić możliwie intuicyjną obsługę.

2.2. Wymagania techniczne sterownika

Urządzenie jest projektowane z myślą i przeznaczeniem do sterowania silnikiem BLDC znajdującym się w laboratorium, gdzie do przeprowadzania prac badawczych potrzebny jest do spełnienia szereg następujących wymagań:

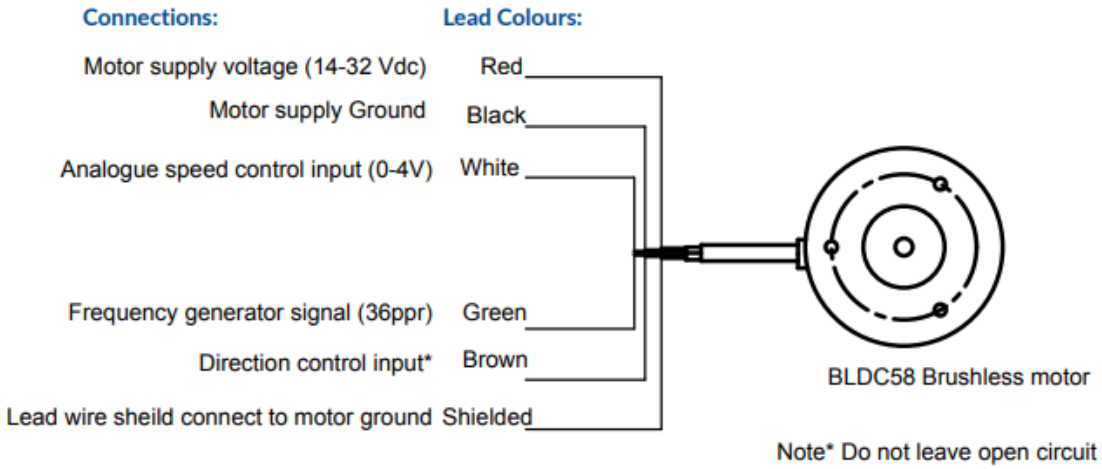
- **dokładność** – silnik powinien być sterowany z dokładnością co do 0,5 RPM (Revolutions Per Minute) przy maksymalnych 60 RPM,
- **interfejs RS-232** – jest to interfejs wykorzystywany do komunikacji z innymi urządzeniami w laboratorium przy pomocy złącza DB9,
- **fizyczny interfejs** – urządzenie ma posiadać ekran i przyciski pozwalające na zmienianie parametrów pracy silnika,
- **funkcjonalność IoT** – możliwość zmiany ustawień za pomocą bezprzewodowej komunikacji,

- **połączenie z silnikiem poprzez gniazdo DB25** – silnik ma wyprowadzony od siebie kabel zakończony wspomnianym konektorem, sterownik ma mieć możliwość jego wpięcia,
- **możliwość zamontowania na szynie DIN** – urządzenie ma się znajdować w szafie elektrycznej, jego wymiary i sposób montażu, mają to umożliwić.

2.3. Specyfikacja silnika BLDC

Silniki bezszczotkowe prądu stałego (BLDC) charakteryzują się wysoką sprawnością, trwałością oraz precyzyjną kontrolą prędkości, co sprawia, że są szeroko wykorzystywane w różnych branżach. Dzięki eliminacji szczotek, stosowanych w tradycyjnych silnikach komutatorowych, silniki BLDC mają mniejsze straty energii oraz zużycie mechaniczne. Znajdują zastosowanie m.in. w elektronice użytkowej (wentylatory, napędy dysków twardych), pojazdach elektrycznych, robotyce, automatyce przemysłowej oraz systemach klimatyzacyjnych i wentylacyjnych (HVAC) [1].

Silnik z do jakiego projektujemy sterownik to model BLDC58-35LEB z serii BLDC58 firmy Mclennan. Wszystkie parametry jakie można znaleźć odnośnie do tego silnika są zamieszczone do pobrania ze strony producenta [2]. Z dokumentacji wiele danych musimy odczytać ze schematów.

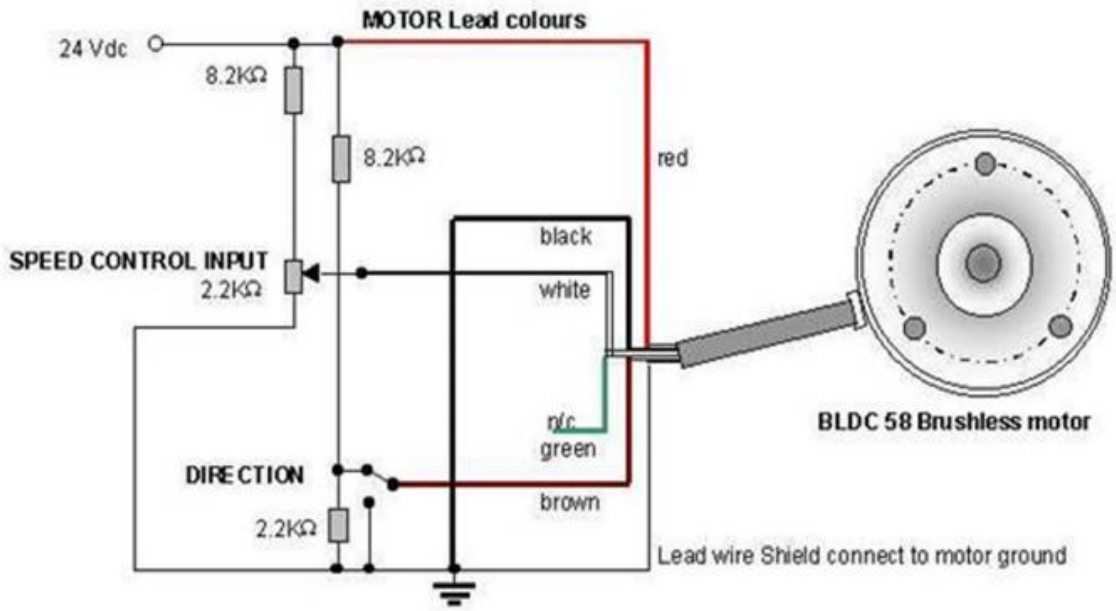


Rys. 2.1 Schemat przedstawiający opis przewodów wyprowadzonych z silnika [2].

Na Rys.2.1 producent zamieścił parę kluczowych informacji, takich jak fakt, że silnik poza zasilaniem, potrzebuje również sterowania analogowym sygnałem napięcia (producent

sugeruje dzielnik napięcia wykonany przy pomocy potencjometru), które przekłada się na prędkość obrotów. Jest również wyprowadzony przewód, którym poprzez podanie wysokiego lub niskiego stanu logicznego steruje się kierunkiem obrotów oraz przewód, który generuje sygnał impulsowy liniowo o częstotliwości liniowo zależnej od obrotów silnika. Znajdują się tam wszystkie potrzebne parametry:

- zasilanie **24 [V]** napięcia stałego,
- zakres sygnału napięcia sterowania: **0-4 [V],**
- maksymalna prędkość obrotowa: **3650 RPM** (obciążenie może ograniczyć do 3000RPM)**,**
- generator sygnału: **36ppr** (pulses per revolution) oznacza ilość pulsów generowaną na obrót



Rys. 2.2 Przykładowy schemat podłączeń dla podstawowej kontroli prędkości [2].

W dokumentacji możemy znaleźć Rys. 2.2, która jest przykładem podstawowego układu do kontroli pracy tego silnika, widzimy, że przewód odpowiadający za kierunek, nie może znajdować się w stanie wysokiej impedancji, dodatkowo wyliczmy sobie jego napięcie stanu wysokiego korzystając z równania na napięcie wyjściowe dzielnika napięciowego:

$$V_{wy} = V_{we} \frac{R_2}{R_1 + R_2} \tag{1}$$

gdzie:

- V_{we} – napięcie wejściowe przyłożone do układu dzielnika napięciowego,
- V_{wy} – napięcie wyjściowe pobierane z węzła pomiędzy rezystorami R_1 i R_2
- R_1 – rezystor znajdujący się pomiędzy napięciem wejściowym a wyjściem,
- R_2 – rezystor znajdujący się pomiędzy wyjściem a masą (punkt odniesienia).

Podstawiając do równania:

$$V_{wy} = 24V \times \frac{2{,}2k\Omega}{8{,}2k\Omega + 2{,}2k\Omega}$$

$$V_{wy} = 5{,}08V$$

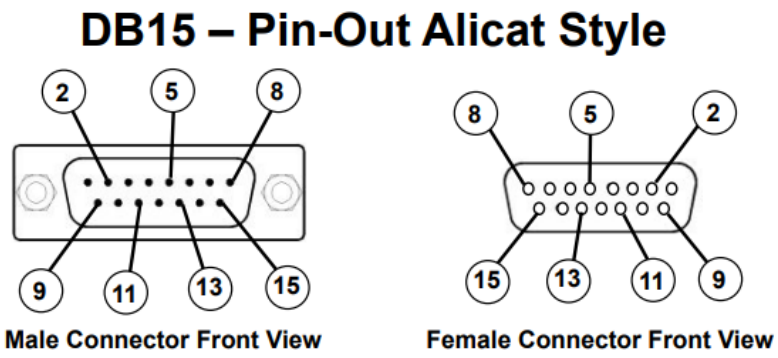
Można zauważyć, że dzielnik napięcia z potencjometrem użytym do sterowania napięcia działa na tych samych wartościach, dla pewności więc autor założył, że jego sterownik będzie w stanie zapewnić napięcie sterujące **w zakresie 0-5 [V].**

Ponadto nie jest sprecyzowane napięcie impulsów generowanych na wyjściu silnika, więc w fazie projektowania warto uwzględnić, że może być to zarówno 24 [V] jak i 5 [V].

Wykonane modyfikacje silnika

Silnik w laboratorium ma dostarczone zewnętrzne zasilanie o wartości 24 [V], jego wszystkie przewody są poprowadzone kablem, który jest zakończony wtykiem DB15.

Do silnika zamontowana jest również przekładnia, która obniża obroty 50 krotnie. Sprawia to, że maksymalna liczba obrotów na wyjściu pod obciążeniem wynosi maksymalnie **60 RPM.** Sterownik na interfejsach powinien wyświetlać informacje o parametrach obrotów na wyjściu z przekładni.



.

Rys. 2.3 Przedstawienie numeracji pinów złącza DB15 [3].

Zakładając numeracje pinów w danym złączu, taką samą jak na Rys. 2.3, można opisać ich poszczególne funkcję.

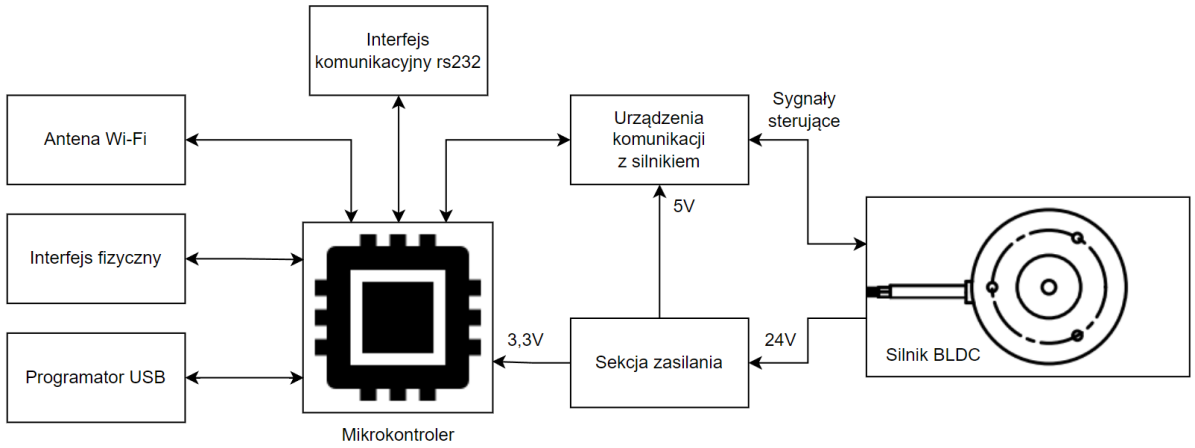
Tabela 1 Opis pinów wtyku DB15 zamontowanego w silniku.

Nr Pinu	Funkcja
1	sterowanie kierunkiem pracy silnika
9	sterowanie prędkością obrotową silnika
12	masa
13	zasilanie 24 [V]
14	odczyt prędkości obrotowej silnika

Tabela 1 zawiera opis pinów, do których zostały przyłączone przewody silnika. Wszystkie nieopisane wyprowadzenia złącza są nie podłączone.

2.4. Architektura projektu sterownika

2.4.1. Schemat blokowy



Rys. 2.4 Schemat blokowy sterownika

Na Rys. 2.4 można zauważyć, że na jednostce mikrokontrolera będzie spoczywała obsługa kilku interfejsów naraz, warto będzie dobrać mikrokontroler mający wbudowaną część z nich. Dodatkowo mikrokontroler monitoruje parametry silnika, przerwania wywołane obsługą interfejsów nie powinny przerywać pomiarów obrotów silnika. Ponadto trzeba zadbać o uzyskanie odpowiedniego zasilania zarówno dla mikrokontrolera (3,3V) jak i dla elementów kontrolujących pracę silnika (ustalone 0-5V) przy dostarczonych z silnika 24V.

2.4.2. Sposób sterowania silnikiem

W projekcie jest potrzeba rozważyć w jaki sposób sterownik będzie wytwarzał analogowe napięcie, które silnik będzie odbierał. W tym celu najlepszym i najprostszym wyborem jest użycie cyfrowo-analogowego przetwornika (DAC), który znajduje się w większości modeli mikrokontrolerów. DAC jest wbudowanym komponentem, umożliwiającym generowanie sygnałów analogowych na podstawie danych cyfrowych. DAC konwertuje cyfrowe dane binarne na ciągłe napięcie lub prąd, proporcjonalne do wartości wejściowej.

Dodatkowo niezbędne będzie wykorzystanie wzmacniacza, gdyż mikrokontroler może jedynie podać na wyjściu przetwornika maksymalne napięcie równe zasilaniu (3,3V), a założone zostało (w rozdziale 2.3), że do kontroli silnika zostanie wykorzystany większy zakres napięć (0-5V).

Przed przystąpieniem do wyboru mikrokontrolera należy sprawdzić jakiej minimalnej dokładności (rozdzielczości) potrzebny jest nam przetwornik DAC.

Przy założeniu liniowej charakterystyki wzmacniacza, za pomocą mikrokontrolera sterujemy napięciem 0-5 V, a silnik liniowo zwiększa swoje obroty w zakresie 0,9 – 5,0 V. Część zakresu przetwornika działa w zakresie, w którym silnik nie rozpoczyna pracy, czyli przetwornik zostanie wykorzystany tylko w pewnej części, a dokładnie: $\frac{5-0,9}{5} * 100\% = 82\%$.

W przypadku, gdyby okazało się, że silnik nie potrzebuje tak szerokiego zakresu (np. wystarczy mu tylko 0,9 – 4 V), wykonanie modyfikacji wzmacniacza, aby zmniejszyć jego zakres będzie bardzo prosta. Jednak taka operacja zmieni nam % wykorzystania przetwornika do $\frac{4-0,9}{4} * 100\% = 77,5\%$

Ustalmy wzór i dzięki któremu oda się policzyć wymaganą rozdzielczość, aby sterować obrotami z potrzebną nam dokładnością. Zakładany jest mniej korzystny % wykorzystania przetwornika:

$$rozdzielczość = \frac{\max RPM}{dokładność\ RPM * 77,5\%} \tag{2}$$

Podstawiając do wzoru (2) ustaloną wcześniej z góry precyzję na poziomie 0,5 RPM oraz max 73RPM otrzymamy działanie:

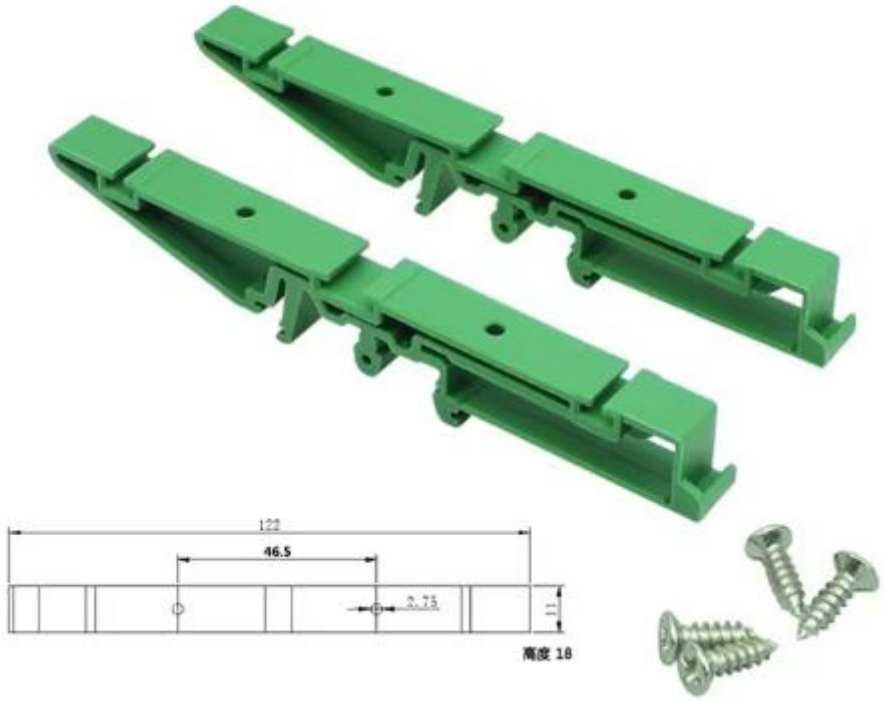
$$rozdzielczość = \frac{73}{0,5 * 77,5\%}$$

$$rozdzielczość \approx 189$$

Rozdzielczość 189 jest pokryta przez DAC o rozdzielczości 8 bitów (rozdzielczość 256 stanów) i taka rozdzielczość będzie wymagana przy wyborze mikrokontrolera.

2.4.3. Możliwość montażu płytki w szafie elektrycznej

Aby zamontować sterownik w szafie elektrycznej, wykorzystane zostaną specjalne uchwyty pozwalające zamontować każde PCB z otworami we właściwym miejscu na szynie DIN (Deutsches Institut für Normung) EN 60715.



Rys. 2.5 Urządzenie do montowania PCB na szynie DIN [4].

Aby umożliwić taki montaż, należy stworzyć PCB, które będzie miało 4 otwory na śruby wysokie na 47mm i dowolnie szerokie.

3. Projektowanie PCB sterownika silnika BLDC

W tym rozdziale opisany zostanie proces realizacji płytki PCB (Printed Circuit Board) dla sterownika silnika BLDC z funkcjonalnością IoT. Projekt PCB zapewnia odpowiednią fizyczną strukturę oraz połączenia dla wszystkich podzespołów sterownika. Proces projektowania płytki PCB dla sterownika BLDC obejmie kilka kluczowych etapów:

1. wybór odpowiedniego oprogramowania,
2. dobór komponentów,
3. projektowanie schematu ideowego,
4. projektowanie płytki PCB,
5. montaż płytki PCB.

Dokładne wykonanie każdego etapu pozwoli na wykonanie wydajnego i niezawodnego systemu sterowania dla silnika BLDC, który będzie w stanie spełnić postawione wcześniej wymagania.

3.1. Wybór komponentów

W procesie projektowania jednym z kluczowych etapów jest dobór odpowiednich komponentów elektronicznych. Przy wyborze elementów do tego projektu brane są pod uwagę trzy główne kryteria: **koszt**, **dostępność** a przede wszystkim **prostota zastosowania.**

Zastosowanie komponentów, które są łatwe do implementacji i mają szeroką dokumentację, może być obrane w projektach, które nie są tworzone z myślą o masowej produkcji, tylko o pojedynczych egzemplarzach. W takim wypadku warto oszczędzić czas projektowania wybierając prostsze do implementacji zastosowania. Prostota zastosowania oznacza również łatwość integracji z innymi elementami układu oraz dostępność gotowych bibliotek wspierających dany komponent w środowiskach programistycznych. W ten sposób możliwe jest szybsze i bardziej niezawodne wdrożenie funkcji sterowania oraz komunikacji bezprzewodowej.

Koszt komponentów odgrywa istotną rolę w każdym projekcie. W projekcie można zastosować popularne komponenty zwykłej klasy, ze względu na podstawowe zastosowanie i podstawowe wymagania (praca w pokojowej temperaturze, w infrastrukturze, która nie jest newralgiczna).

Dodatkowo w kwestii kosztów wyboru układów scalonych, ważne jest sprawdzić, czy wymagają one dodatkowego osprzętu w postaci zewnętrznie przyłączonych kondensatorów, cewek czy rezystorów. Typowo te elementy mają bardzo mały koszt za jednostkę, jednak przez to można je kupić tylko hurtem, więc warto dobierać układy scalone tak, aby potrzebować jak najmniej różnych elementów pasywnych, co ograniczy potrzebną do zamówienia ilość partii.

W kwestii dostępności, warto zobaczyć, czy da się zamówić wszystkie komponenty u jednego dostawcy oraz czy gwarantuje on wysyłkę ich wszystkich w przystępnym terminie – zamawiając wiele komponentów łatwo o pomyłkę i zamówić produkt niedostępny chwilowo na magazynie. Błąd taki może nam zatrzymać pracę nad projektem, a bardzo możliwe jest, że na stanie magazynu znajduje się bliźniaczy element.

Wybór komponentów elektronicznych opartych na powyższych kryteriach znajduje swoje uzasadnienie w literaturze, która wskazuje na konieczność zbalansowania kosztów, dostępności i niezawodności elementów [8].

Komponenty postanowiłem zamawiać ze strony Aliexpress.pl ze względu na dużą dostępność i niską cenę, minusem jest długi czas oczekiwania przesyłki, jednak dłuższy okres i tak będzie nieunikniony oczekiwaniem na produkcję i dostawę PCB.

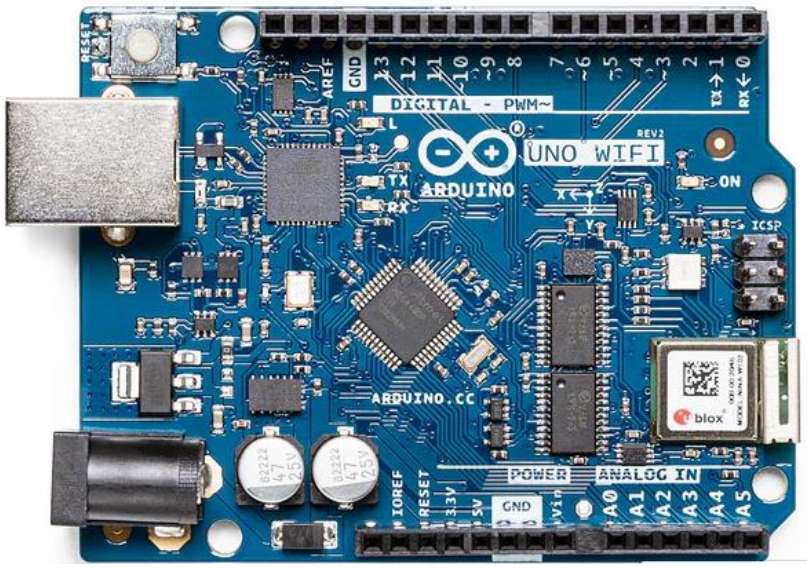
3.1.1. Mikrokontroler

Wybór mikrokontrolera to najważniejszy wybór, dlatego powinno się wykonać go jako pierwszy, może on znacząco wpłynąć na decyzje w wyborze innych elementów ze względu na interfejsy jakie obsługuje i w jaki sposób się go programuje.

W projekcie rozważono wybór trzech różnych mikrokontrolerów z rodzin: Arduino, STM32 oraz ESP32.

Arduino

Jedna z najbardziej znanych platform mikrokontrolerowych, ceniona za prostotę i łatwość użycia, zwłaszcza dla początkujących. Najpopularniejsze mikrokontrolery Arduino to układy oparte na architekturze AVR, takie jak **ATmega328P** [5].



Rys. 3.1 Arduino Uno WiFi REV2 [5].

Istnieją również o wiele bardziej rozbudowane platformy, takie jak na Rys 3.1, jednak są one droższe i dalej oferują dość ograniczoną pinów przeznaczoną dla użytkownika.

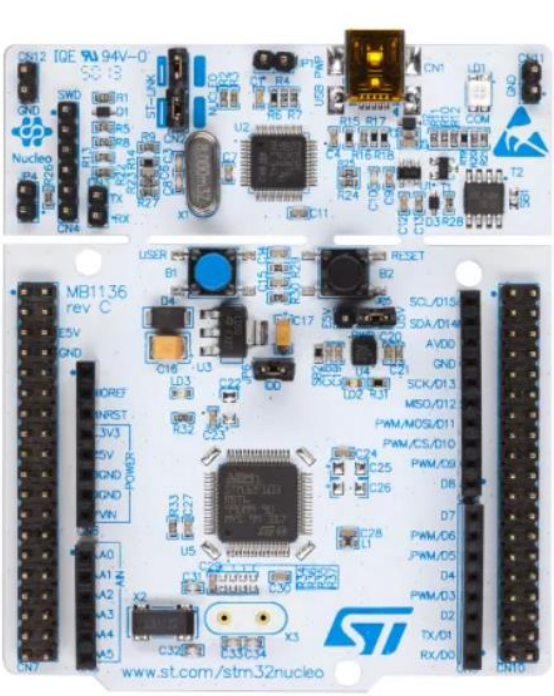
Kluczowe cechy platformy Arduino:

- **Łatwość programowania**: Arduino oferuje prosty w użyciu środowisko IDE oraz ogromne wsparcie społeczności, co czyni go idealnym wyborem dla osób rozpoczynających pracę z mikrokontrolerami.
- **Mniejsza moc obliczeniowa**: typowe mikrokontrolery Arduino (np. ATmega328P) mają całkiem niską moc obliczeniową (16 MHz) oraz ograniczoną pamięć RAM i Flash.
- **Ogromna ilość bibliotek i gotowych projektów**: poza stworzonym przez firmę Arduino środowiskiem, bardzo duża popularność jego wpływa na ogrom gotowych bibliotek open-source, które mogę znacznie ułatwić pracę, nie musząc pisać wszystkiego od zera.

Choć Arduino sprawdza się dobrze w prostych projektach, jego ograniczenia w zakresie wydajności oraz braku wbudowanej łączności bezprzewodowej czynią je mniej atrakcyjnym wyborem w przypadku prowadzonego projektu.

STM32

STM32 to seria mikrokontrolerów produkowanych przez STMicroelectronics, opartych na architekturze ARM Cortex-M. Jest to bardziej zaawansowana platforma w porównaniu do Arduino i oferuje szeroki zakres możliwości. Firma oferuje szereg różnych klas mikrokontrolerów zależnie od wymaganego od nas zastosowania. **Seria L** oferuje rozwiązania jak najbardziej energooszczędne, **seria W** ma możliwość obsługi bezprzewodowych peryferiów, a **seria F** świetnie się sprawdzi jako ogólne zbalansowane rozwiązanie [6].



Rys. 3.2 STM32F334R8 Nucleo-64 płytka prototypowa [6].

Kluczowe cechy STM32:

- **Wysoka wydajność**: STM32, zwłaszcza modele z serii F4 lub F7, oferują znacznie większą moc obliczeniową niż mikrokontrolery Arduino, dzięki wykorzystaniu rdzeni ARM Cortex-M o taktowaniu od 72 MHz do ponad 200 MHz, niektóre produkty posiadają nawet 2 rdzenie.
- **Bogate peryferia**: STM32 oferuje rozbudowane możliwości w zakresie interfejsów peryferyjnych, takich jak DAC, ADC, SPI, I2C, UART, CAN oraz Ethernet. Jest to

bardzo elastyczna platforma dla bardziej wymagających aplikacji przemysłowych i komercyjnych, większość z ich platform posiada bardzo dużą ilość obsługi pinów przeznaczoną dla użytkownika.

- **Trudniejsza implementacja**: STM32, mimo swoich możliwości, jest trudniejsze w implementacji niż Arduino, wymagając bardziej zaawansowanych narzędzi programistycznych (np. STM32CubeMX) oraz wiedzy na temat konfiguracji peryferiów i zarządzania zasobami systemowymi.

STM32 to mocna platforma, szczególnie w projektach, gdzie wymagane są skomplikowane obliczenia i zaawansowane peryferia. Jednak, nie posiada ona wbudowanego modułu Wi-Fi, dodatkowo ich użycie wymaga o wiele bardziej dogłębnego przestudiowania dokumentacji, wraz z większymi możliwościami rośnie trud ich wykorzystania. Ponadto układy tej firmy te są droższe i większość ich możliwości nie znajduje wykorzystana w tym projekcie.

ESP32

Mikrokontroler produkowany przez Espressif, który zyskał popularność dzięki szerokim możliwościom w zakresie łączności bezprzewodowej oraz zaawansowanym funkcjom cyfrowo-analogowym [7].

Rys. 3.3 ESP32-DevKitM-1 [7].

Kluczowe cechy ESP32:

- **Wbudowane Wi-Fi oraz Bluetooth**: ESP32 ma zintegrowane moduły Wi-Fi oraz Bluetooth, co czyni go idealnym rozwiązaniem dla aplikacji IoT,
- **Dwa przetworniki DAC (cyfrowo-analogowe)**: w odróżnieniu od reszty (z wyjątkiem niektórych droższych modeli STM32), ESP32 ma dwa przetworniki DAC.
- **Wysoka wydajność**: ESP32-WROOM jest wyposażony w dwurdzeniowy procesor taktowany z częstotliwością 40 MHz, co zapewnia odpowiednią moc obliczeniową do przetwarzania danych w czasie rzeczywistym.
- **Rozbudowane GPIO i funkcje peryferyjne**: ESP32 oferuje szeroką gamę interfejsów peryferyjnych, takich jak UART, SPI, I2C, co czyni go wszechstronnym narzędziem do obsługi różnych komponentów.
- **Obsługa środowiska Ardunio:** wspieranie tego prostego środowiska przez ESP32 znacznie ułatwia pracę nad projektem.

ESP32 jest bardzo atrakcyjnym rozwiązaniem, zwłaszcza w kontekście aplikacji IoT, dzięki swojej zdolności do integracji bezprzewodowej oraz dużej elastyczności w zarządzaniu wejściami/wyjściami. Dodatkowym atutem jest szerokie wsparcie społeczności oraz dostępność wielu bibliotek i przykładów kodu.

Ostateczny wybór

Z modeli ESP32 zdecydowano się na wybór modelu mikrokontrolera **ESP32-WROOM-E**, który jest najnowszym z najpopularniejszej serii WROOM. Seria ta jest jedną z najpopularniejszych poprzez swoją wszechstronność i zadowalającą moc obliczeniową, a to wszystko zawarte w bardzo niskiej cenie.

Wybór tego mikrokontrolera do projektu sterownika silnika BLDC z funkcjonalnością IoT był uzasadniony przede wszystkim jego wbudowaną funkcjonalnością bezprzewodową, co jest kluczowe dla aplikacji IoT. Mikrokontroler ten oferuje także dwa rdzenie pozwalające wykonywać pomiary ciągłe w czasie nienaruszone wywołaniem przerwań, co znacznie zwiększa dokładność i elastyczność, co jest niezbędne w kontekście precyzyjnego sterowania silnikiem BLDC. W porównaniu do STM32, ESP32 ma przewagę dzięki zintegrowanej łączności Wi-Fi i Bluetooth, wraz z wbudowaną anteną co pozwala na uniknięcie zakupu i montażu dodatkowych komponentów i upraszcza projekt. Posiada również wbudowane dwa 8 bitowe przetworniki DAC, które można połączyć celem uzyskania jeszcze wyższej precyzji. ESP32 jest optymalnym wyborem, zapewniającym odpowiednią wydajność, funkcjonalność IoT oraz prostotę implementacji.

3.1.2. Układy scalone zasilania

W układach elektronicznych konieczne jest odpowiednie zarządzanie napięciami zasilającymi poszczególne moduły. Zbyt wysokie napięcie może doprowadzić do uszkodzenia elementów, podczas gdy zbyt niskie może uniemożliwić ich poprawne działanie. Dlatego stosuje się układy scalone przetworników napięcia, które przekształcają napięcie wejściowe na odpowiednie poziomy zgodnie z wymaganiami danego systemu. W tym celu wybrano dwa popularne układy scalone: **LM2596S-5** oraz **AMS1117-3.3**.

W projekcie sterownika, kluczowym zadaniem było dobranie odpowiednich układów scalonych, które zapewnią stabilne i efektywne zasilanie dla różnych podzespołów urządzenia. W związku z tym należało zastosować przetworniki napięcia, które obniżą napięcie z dostępnych 24V do poziomów odpowiednich dla mikrokontrolera i innych elementów układu [9].

LM2596S-5

LM2596S-5 to przetwornik typu **buck (step-down)**, stosowany do obniżania napięcia wejściowego na niższe, stabilne napięcie wyjściowe. Jego działanie opiera się na cyklicznym przełączaniu elementów kluczujących (np. tranzystorów MOSFET) oraz magazynowaniu energii w dławiku. Dzięki technice modulacji szerokości impulsu (**PWM**), przetwornik może dostarczać stabilne napięcie wyjściowe nawet przy dużych różnicach napięć wejściowych i wyjściowych.

LM2596S-5 wyróżnia się wysoką sprawnością, która wynosi ponad 90%, co sprawia, że jest to efektywne rozwiązanie do konwersji napięcia z 24V na 5V. Jednocześnie układ ten może pracować przy częstotliwości przełączania wynoszącej 150 kHz, co pozwala na stosunkowo małe rozmiary komponentów pasywnych (dławik, kondensatory). Należy jednak pamiętać, że przetworniki impulsowe generują pewne zakłócenia elektromagnetyczne, co wymaga odpowiedniego filtrowania wrażliwych układów. Dodatkowo układ posiada wbudowane mechanizmy ochrony przed przeciążeniami, co stanowi istotną zaletę przy jego zastosowaniu w zasilaniu urządzeń o zmiennym obciążeniu. Przetwornik ten automatycznie monitoruje prąd wyjściowy i w przypadku przekroczenia dopuszczalnej wartości aktywuje ograniczenie prądu, zapobiegając przegrzewaniu się i uszkodzeniu układu. Funkcja ta działa w sposób ciągły, co oznacza, że w sytuacji przeciążenia przetwornik zmniejsza prąd wyjściowy, aby chronić obwód przed zniszczeniem [10].

AMS1117-3.3

LDO (ang. **Low Dropout Regulator**) to stabilizator liniowy niskiego spadku napięcia, który umożliwia przetwarzanie napięcia wejściowego na niższe napięcie wyjściowe przy minimalnej różnicy między nimi. Przetwornik LDO składa się z tranzystora regulacyjnego (zwykle typu PNP lub MOSFET) oraz układu odniesienia napięcia i sprzężenia zwrotnego.

Przetworniki LDO są proste w budowie, generują niewielkie zakłócenia elektromagnetyczne (brak przełączania impulsowego), łatwo integrują się z innymi układami, a także mogą pracować przy minimalnej różnicy napięć wejściowego i wyjściowego. Jako ich wadę można wymienić niską efektywność, przy dużych różnicach napięć, ponieważ nadmiar energii rozpraszany jest w postaci ciepła. To może prowadzić do przegrzewania się układu i strat mocy [9].

Układ **AMS1117-3.3** to stabilizator liniowy, który przekształca napięcie wejściowe 5V na stabilne napięcie 3.3V. Jest on bardzo tani oraz wymaga minimalnej ilości elementów pasywnych (tylko dwa kondensatory) [11].

3.1.3. Moduł komunikacji RS-232

Jednym z wymagań projektowych jest sposób komunikowania się za pomocą protokołu RS232, jednak wybrany przez nas mikrokontroler nie obsługuje go. Na rynku są gotowe rozwiązania oferujące prostą konwersje protokołu UART (który jest wbudowany w mikrokontroler) na RS232.

UART to uniwersalny asynchroniczny odbiornik-nadajnik stosowany do komunikacji szeregowej między dwoma urządzeniami. UART działa na zasadzie przesyłania danych w formie bitów szeregowych, gdzie każdy bit jest przesyłany jeden po drugim w ustalonym porządku (LSB lub MSB). UART nie wymaga sygnału zegarowego, a synchronizacja nadawcy i odbiorcy jest osiągana przez ustalenie wspólnej prędkości transmisji (baud rate) oraz użycie bitu startu i stopu.

RS232 to starszy standard komunikacji szeregowej stosowany głównie w urządzeniach przemysłowych i komputerowych. RS232 definiuje zarówno sygnały elektryczne, jak i protokół wymiany danych, gdzie sygnały są przesyłane różnicowo między nadajnikiem a odbiornikiem, protokół ten zapewnia prędkość do 20 Kbps [12].

Moduł **MAX3232** to przetwornik napięć, który konwertuje sygnały UART (TTL) na sygnały RS232, dzięki czemu możliwa jest komunikacja między urządzeniami korzystającymi z różnych poziomów napięć. UART działa na poziomach logicznych 0-5V (lub 0-3.3V), podczas gdy RS232 wymaga napięć ±12V. **MAX3232** jest odpowiedzialny za przekształcenie poziomów logicznych TTL na poziomy wymagane przez RS232, co czyni go idealnym rozwiązaniem w projektach wymagających kompatybilności między tymi standardami [13].

3.1.4. Układ napięcia sterującego silnikiem

W wybrany mikrokontroler są wbudowane dwa przetworniki DAC, sumując ich napięcie można uzyskać jeden, dokładniejszy przetwornik. Aby to zrobić potrzeba dwóch wzmacniaczy operacyjnych.

Wzmacniacz będzie zasilany napięciem 5V i również powinien być w stanie takie napięcie podać na wyjściu, wymaga to, żeby był wykonany w technologii rail-to-rail, pozwalającej uzyskać napięcie wyjściowe wzmacniacza bliskie napięciu zasilania (normalnie można spodziewać się maksymalnego napięcia wyjściowego mniejszego o ponad 1 V od napięcia zasilania) [14].

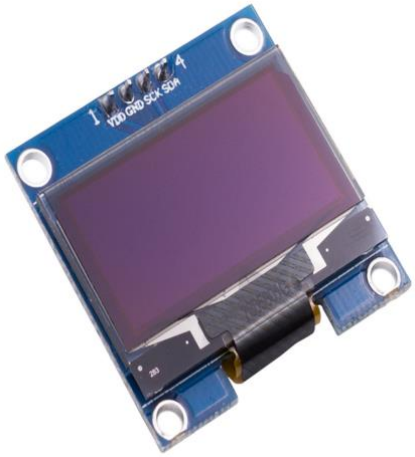
Na schemacie zamieszczonym w dokumentacji silnika (Rys 2.2), można zauważyć, że producent w dzielniku napięcia 24V zastosował rezystory o wartości rzędu 10kΩ. Na podstawie tego przyjęto z zapasem, że wzmacniacz powinien być w stanie osiągnąć prąd o wartości 10mA.

Wszystkie nasze wymagania spełnia układ **LMC6482,** jest to układ scalony wykonany w technologii CMOS zawierający w sobie dwa wzmacniacze operacyjne. Potrafi on przekazywać napięcia na wyjściu bliskie napięciu zasilania i pracuje w wymaganym napięciu [15].

Żeby zbudować układ potrzeba również dostarczyć zasilanie -5V, łatwym rozwiązaniem tego problemu jest zastosowanie układu **ICL7660.** Układ ten jest w stanie wytworzyć napięcie -5V z napięcia 5V. Należy uważać, aby nie wystawiać go na duże obciążenie, nie jest on przeznaczony dla dużych obciążeń. W projekcie będzie użyty tylko do zasilenia części wzmacniacza, którego wyjście trafia do wejścia wysokiej impedancji drugiego wzmacniacza, tak więc prądy płynące w tym układzie będą minimalne [16].

3.1.5. Ekran interfejsu fizycznego

Jako ekran zdecydowano się na gotowy moduł zasilany napięciem 3,3 / 5 V. Obraz w tym module można ustawiać za pomocą komunikacji I2C.



Rys. 3.4 Ekran użyty w projekcie [17].

Wybrano dwukolorowy ekran OLED, o rozdzielczości 128x64. Ekran o małych rozmiarach 1’3 cala, nie zajmie dużo miejsca, przy czym pozwoli wyświetlić wszystkie potrzebne informacje o parametrach silnika, w czytelny sposób (tylko 3 linijki tekstu). Technologia OLED dodatkowo jest bardzo energooszczędna [17].

3.1.6. Programator

Aby zaprogramować na płytce ESP32 mikrokontroler, niezbędny jest układ, który pozwoli przekonwertowanie sygnału USB na UART. Dodatkowo, aby rozpocząć proces programowania należy na chwilę podciągnąć do masy pin EN, co spowoduje reset, a następnie podczas włączania się mikrokontrolera musi przez chwile być podciągnięty do masy pin GPIO0 [18].

Układ **CH340C** jest dwukierunkowym konwerterem USB na UART, a dodatkowo po odpowiednim podłączeniu, jest w stanie sam zresetować do stanu programowania. Użycie go jako programatora bardzo ułatwi prace związane z programowaniem mikrokontrolera.

3.2. Wybór oprogramowania

Do realizacji projektu zdecydowano się na wykorzystanie oprogramowania **KiCad**, które jest jednym z najbardziej popularnych i wszechstronnych narzędzi dostępnych na rynku.

KiCad to oprogramowanie typu open-source, które umożliwia projektowanie zarówno prostych, jak i złożonych układów elektronicznych. Program ten jest szeroko stosowany w środowiskach akademickich, jak również w przemyśle, głównie ze względu na swoją elastyczność, szeroki zakres funkcji oraz brak opłat licencyjnych. KiCad oferuje wiele narzędzi do projektowania schematów ideowych, projektowania PCB, a także wbudowane narzędzia do symulacji oraz generowania plików produkcyjnych, tak zwanych „*Gerber files*”.

Ponadto przez swoją popularność istnieje do niego wiele poradników jak i wiele utworzonych wątków na forum pozwalające na wyszukanie natrafionego potencjalnego problemu i jego rozwiązania.

W czasie pracy pisania była wykorzystywana wersja oprogramowania **KiCad 8.0.**

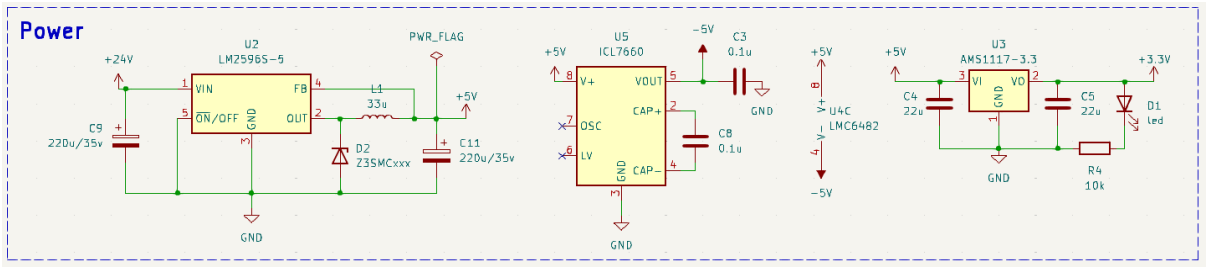
3.3. Projektowanie schematu ideowego

Schematu ideowego używa się do odwzorowania połączeń elektrycznych między komponentami oraz ich funkcji w układzie. Na tym etapie weryfikujemy działanie projektu, zapewniając, że poszczególne elementy będą ze sobą poprawnie współpracować.

Elementy powinny być ułożone w sposób przejrzysty, który odzwierciedla przepływ sygnału i zasilania. Warto również zwrócić uwagę na odpowiednie grupowanie bloków funkcjonalnych – np. sekcji zasilania, komunikacji, sterowania itd. Przemyślany układ schematu ułatwi późniejsze projektowanie PCB [8].

Elementy na schemacie są łączone za pomocą tzw. **netów**, które reprezentują połączenia elektryczne między wyprowadzeniami komponentów. Ważne jest, aby połączenia były jasne i nie mylące. Zastosowanie **etykiet (labels)** może pomóc w identyfikacji poszczególnych sygnałów i zwiększa przejrzystość schematu.

3.3.1. Sekcja zasilania



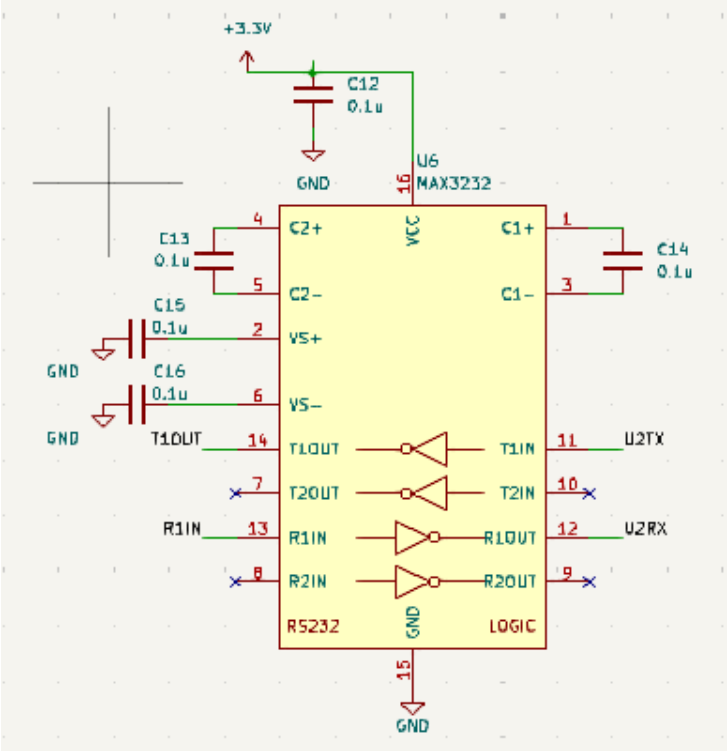
Rys. 3.5 Schemat ideowy przedstawiajacy układy sterownika odpowiadające za zasilanie

Wszystkie układy podłączono do elementów pasywnych zgodnie z wskazówkami producenta, tak jak na Rys. 3.4. W przypadku tego projektu wymogi spełnia zamieszczany często w notach katalogowych *example application*, co znacznie ułatwia pracę.

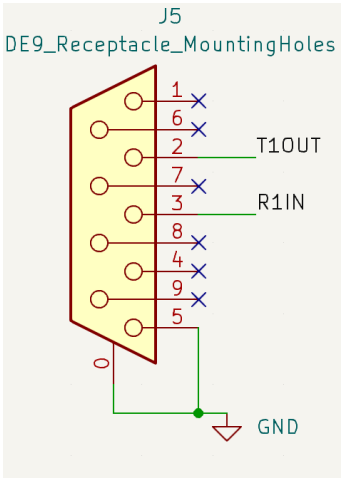
Do wszystkich wyjść napięcia, dołączamy „power flag”, używa go tzw. ERC (Electrical Rule Check), który sprawdza czy aby na pewno wszystkie elementy na schemacie są przyłączone do zasilania.

Dodatkowo do zasilania 3,3 V została podpięta dioda z połączonym szeregowo rezystorem, celem sygnalizowania, że z układem zasilania jest wszystko w porządku.

3.3.2. Układ komunikacji rs-232



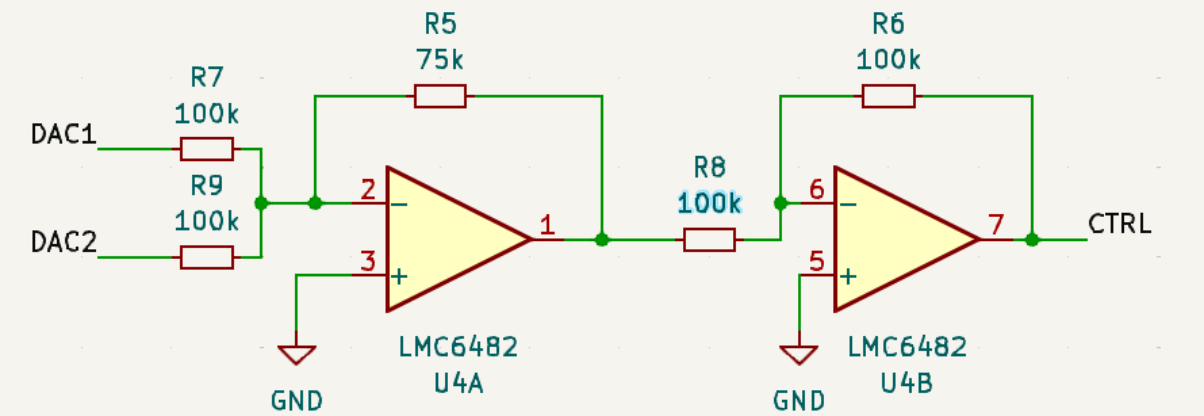
Rys. 3.6 Sposób podłączenia konwertera MAX3232



Rys. 3.7 Wyprowadzenia konektora DB9

Układ MAX3232 (Rys. 3.6) podłączamy zgodnie z instrukcjami producenta, następnie część układu z interfejsem RS232 podłączamy do pinów konektora DB9, które odpowiadają za komunikacje.

3.3.3. Układ wzmacniaczy do analogowego wyjścia napięcia.



Rys. 3.8 Sposób podłączenia wzmacniaczy

W czasie wykonywania projektu zdecydowano się wykorzystać obydwa 8 bitowe przetworniki DAC mikrokontrolera poprzez podłączenie ich do sumującego wzmacniacza operacyjnego. Da to efekt równy sterowaniu z rozdzielczością dwa razy większą niż jednego DAC.

Napięcie na wyjściu wzmacniacza sumującego jest ujemne, dlatego aby na wyjściu otrzymać napięcie dodatnie, użyto jeszcze jeden wzmacniacz operacyjny, który będzie działał jako inwerter.

Obliczenia do wartości rezystorów:
Wstępnym założeniem jest wykorzystanie rezystorów rzędu 100k Ω oraz z szeregu dokładności E24. Wzór na napięcie wyjściowe wzmacniacza sumującego, mającego identyczne rezystory na wejściu:

$$Uwy = -\frac{Rf}{Rwe}(U1 + U2) \tag{3}$$

Po przekształceniu:

$$Rf = -\frac{Uwy * Rwe}{(U1 + U2)}$$

A by uzyskać interesującą nas proporcje rezystorów, podstawiamy znane nam wartości napięć, oraz zakładamy, że rezystory **Rwe** będą miały wartość 100k Ω.

$$Rf = \frac{5 * 100\,000}{3{,}3 + 3{,}3} = 75{,}75(75) k\,\Omega$$

Przybliżając wynik do wartości z szeregu E24:

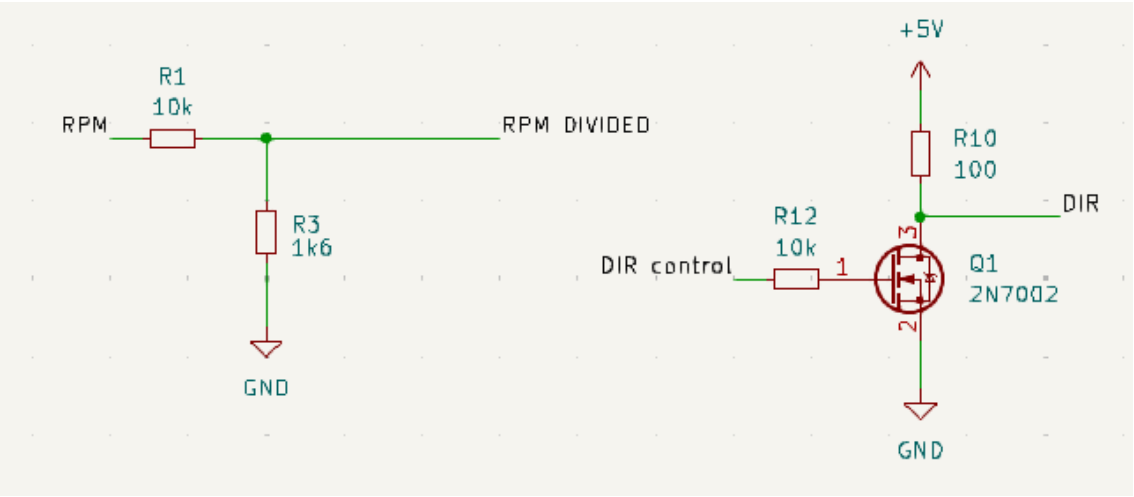
Tabela 2 Wycinek tabeli „Najpopularniejsze szeregi i należące do nich wartości” [19].

E24	10-11-12-13-15-16-18-20-22-24-27-30-33-36-39-43-47-51-56-62-68-75-82-91

Rezystor odpowiadający za sprzężenie zwrotne będzie miał wartość 75kΩ.

3.3.4. Odczyt prędkości i sterowanie kierunkiem

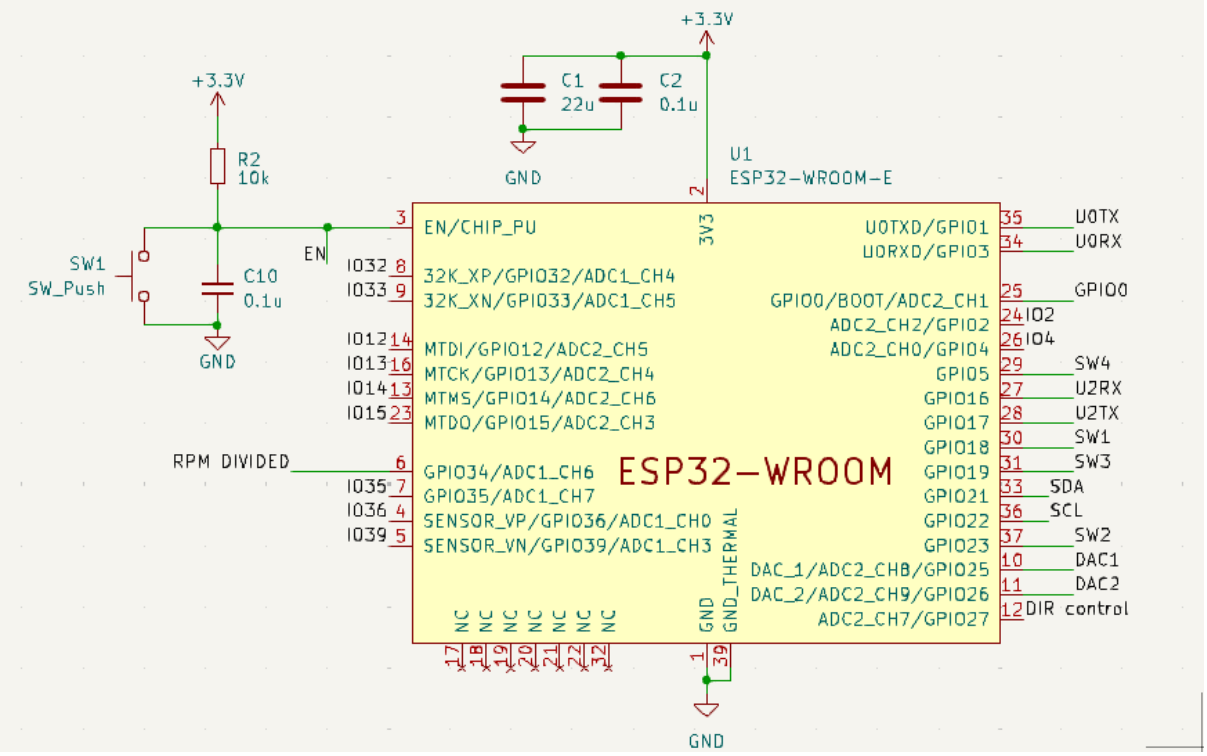
Wartość szczytowa napięcia, jakie silnik może wytwarzać na generatorze impulsów służącym do pomiaru prędkości obrotowej nie została opisana w specyfikacji producenta. W przypadku, gdyby to było 24V w obawie przed uszkodzeniem mikrokontrolera zaprojektowano dzielnik napięcia taki jak na Rys. 3.9. Gdyby jednak okazało się, że napięcie jest podawana w bezpiecznym zakresie, a dzielnik uniemożliwia odczytanie wartości dla mikrokontrolera (za niskie napięcie), można wylutować rezystor podciągający do masy.



Rys. 3.9 Schemat dzielnika napięcia i tranzystora sterującego kierunkiem obrotów

Po prawej stronie Rys. 3.9, widoczny jest układ, który steruje podawaniem napięcia odpowiedzialnego za kierunek obrotów, poprzez mikrokontroler sterujący bramką tranzystora.

3.3.5. Podłączenie mikrokontrolera

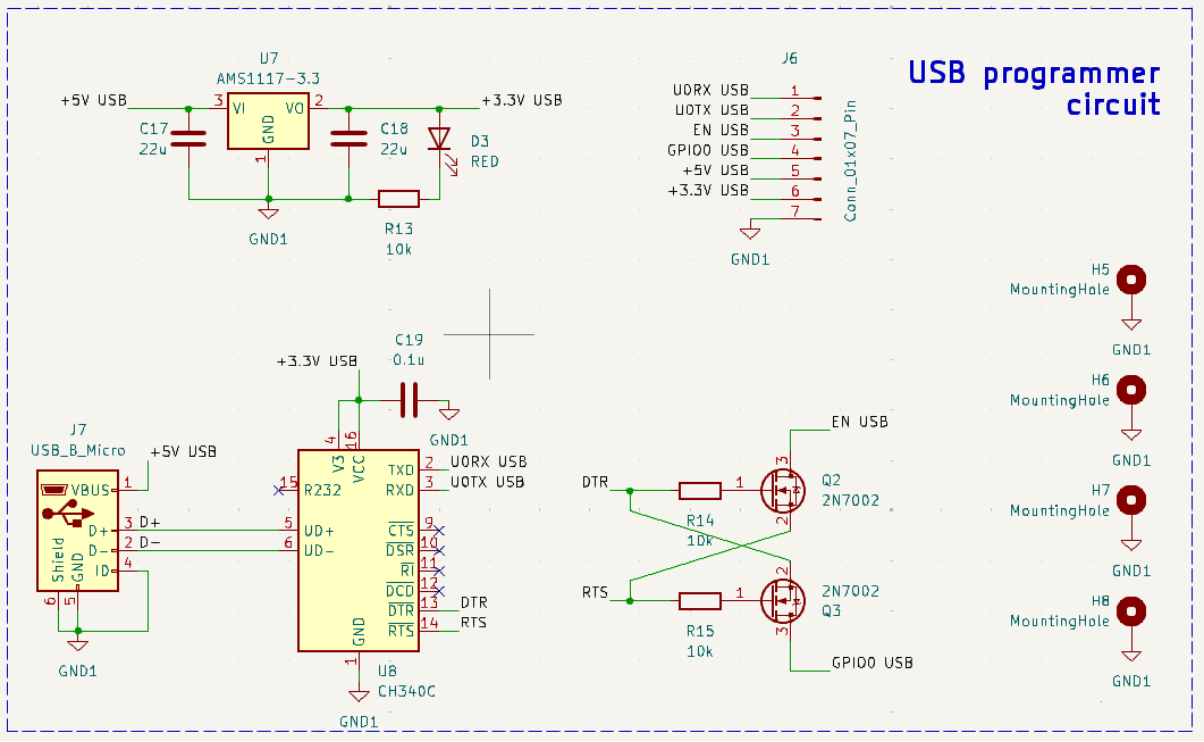


Rys. 3.10 Schemat podłaczeń mikrokontrolera

Mikrokontroler podłączamy zgodnie z zaleceniami producenta oraz wykorzystując interesujące nas piny zgodnie z ich przeznaczeniem. Dodatkowo wyprowadzono nawet nie użyte w projekcie piny, do złącz, na wszelki wypadek nieprzewidzianej konieczności skorzystania z nich. Na Rys. 3.10 możemy też zauważyć, że w razie potrzeby umieszczono przycisk podciągający do masy in EN, odpowiadający za reset mikrokontrolera [18].

3.3.6. Układ programatora

Programator nie jest niezbędną częścią działania sterownika, jednak jest niezbędny do jego wykonania. Z tego powodu zdecydowano potraktować programator jako oddzielny układ, który będzie można odczepić od reszty sterownika, celem przyszłego użycia w innych projektach.



Rys. 3.11 Schemat ideowy programatora ESP32 przy pomocy USB.

Na Rys 3.4 można zauważyć, że programator został zaprojektowany z myślą samodzielnego działania (własny układ zasilania 3,3 V) oraz z wyprowadzeniami pinów, które pozwolą na programowanie ESP32, które również ma wyprowadzone takie piny (a przynajmniej RX i TX).

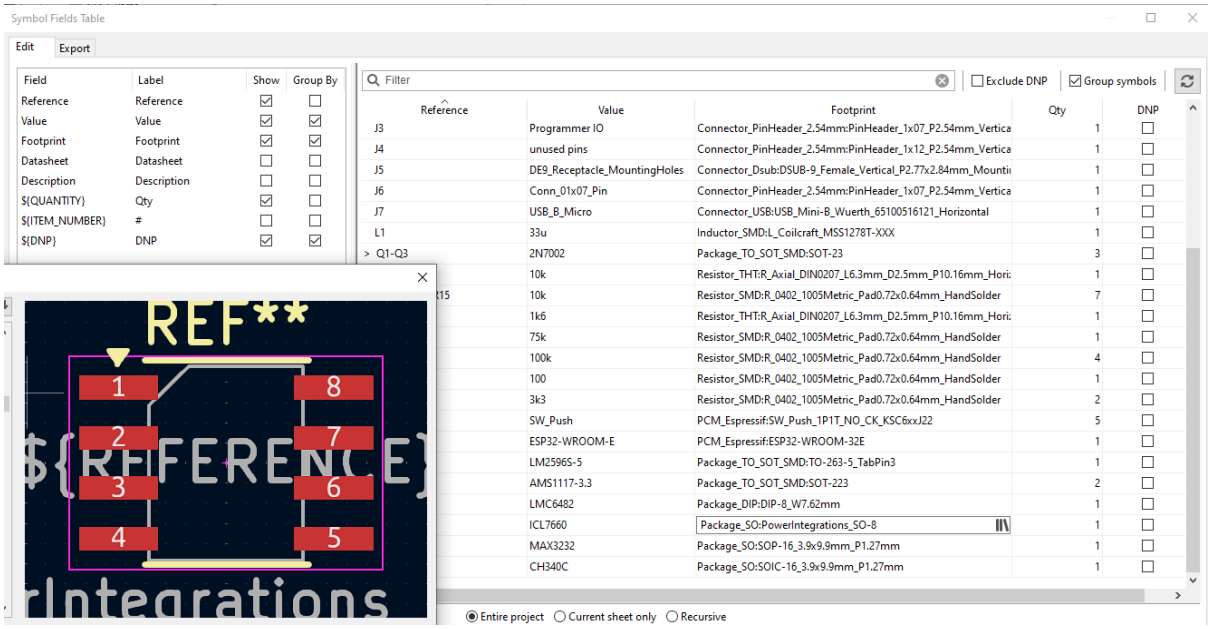
Dodatkowo wykorzystano dwa tranzystory MOSFET pozwalające wykorzystać logikę układu do samoczynnego wejścia mikrokontrolera w tryb bycia programowanym.

Ważne do zaznaczenia jest, że interfejs USB wykorzystuje różnicową parę sygnałów, aby KiCad ją jako tako rozpoznał, co będzie pomocne podczas procesu fizycznego projektowania płytki, należy nazwać parę linii sygnału tak samo, przy czym jedną nazwę zakończyć znakiem „+”, a drugą znakiem „-”.

3.4. Projektowanie PCB

Projektowanie płytki drukowanej to etap w wykonywaniu urządzeń elektronicznych, który przekształca teoretyczny schemat ideowy w fizyczną warstwę, na której zostaną umieszczone wszystkie elementy elektroniczne.

Projektowanie PCB w programie KiCad rozpoczyna się od doboru odpowiednich **footprintów** dla elementów schematu ideowego. **Footprint** to fizyczny układ wyprowadzeń elementu, który zostanie odwzorowany na płytce drukowanej. Każdy element schematu musi mieć przypisany odpowiedni footprint, który odpowiada jego rzeczywistej obudowie



Rys. 3.12 Wycinek z ekranu podczas pracy nad doborem footprintów

Po wybraniu footprintów, należy uwzględnić tolerancje zgodne z możliwościami produkcyjnymi dostawcy PCB. W tym projekcie zdecydowano się na dostawce JLCPCB, a jego możliwości i limity tolerancji można znaleźć na jego stronie internetowej. Tolerancje dotyczą szerokości ścieżek, odległości między nimi, a także wymiarów otworów przelotowych (vias) i padów lutowniczych. Wybieramy także ilośc warstw naszej płytki, do naszych zastosowań wystarczy najtańsza opcja – 2 warstwowa.

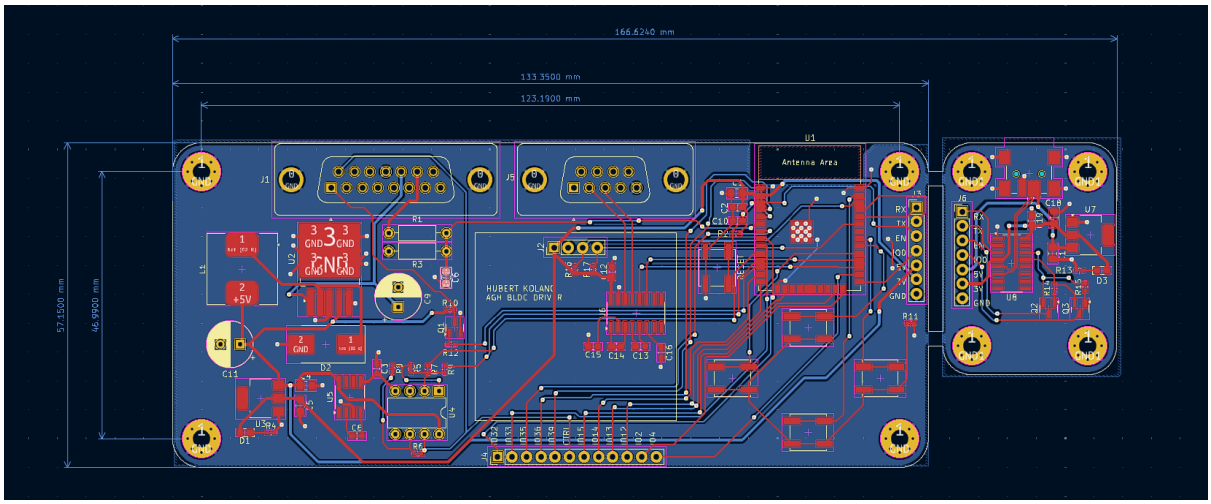
PCB musi mieć 4 otwory na śruby umieszczone na odpowiednich wymiarach (rozdział 2.4.3).

Kolejnym krokiem jest **rozmieszczenie elementów** na płytce PCB. Odpowiednie rozmieszczenie komponentów wpływa nie tylko na estetykę projektu, ale przede wszystkim na jego funkcjonalność i wydajność. Ważne punkty do zapamiętania podczas projektowania PCB:

- **Grupowanie elementów funkcjonalnie powiązanych** - elementy odpowiedzialne za zasilanie powinny być umieszczone blisko siebie, a kluczowe elementy sygnałowe możliwie jak najbliżej mikrokontrolera,
- **Optymalizacja miejsca** - elementy należy umieszczać tak, aby zajmowały jak najmniej miejsca, jednocześnie zapewniając dostęp do padów lutowniczych oraz wygodę montażu. Warto również uwzględnić możliwe zakłócenia elektromagnetyczne (EMI) i oddzielić ścieżki sygnałowe od zasilających.
- **Kondensatory odsprzęgające –** powinny być umiejscowione możliwie blisko wejść/wyjść pinów zasilających układy.

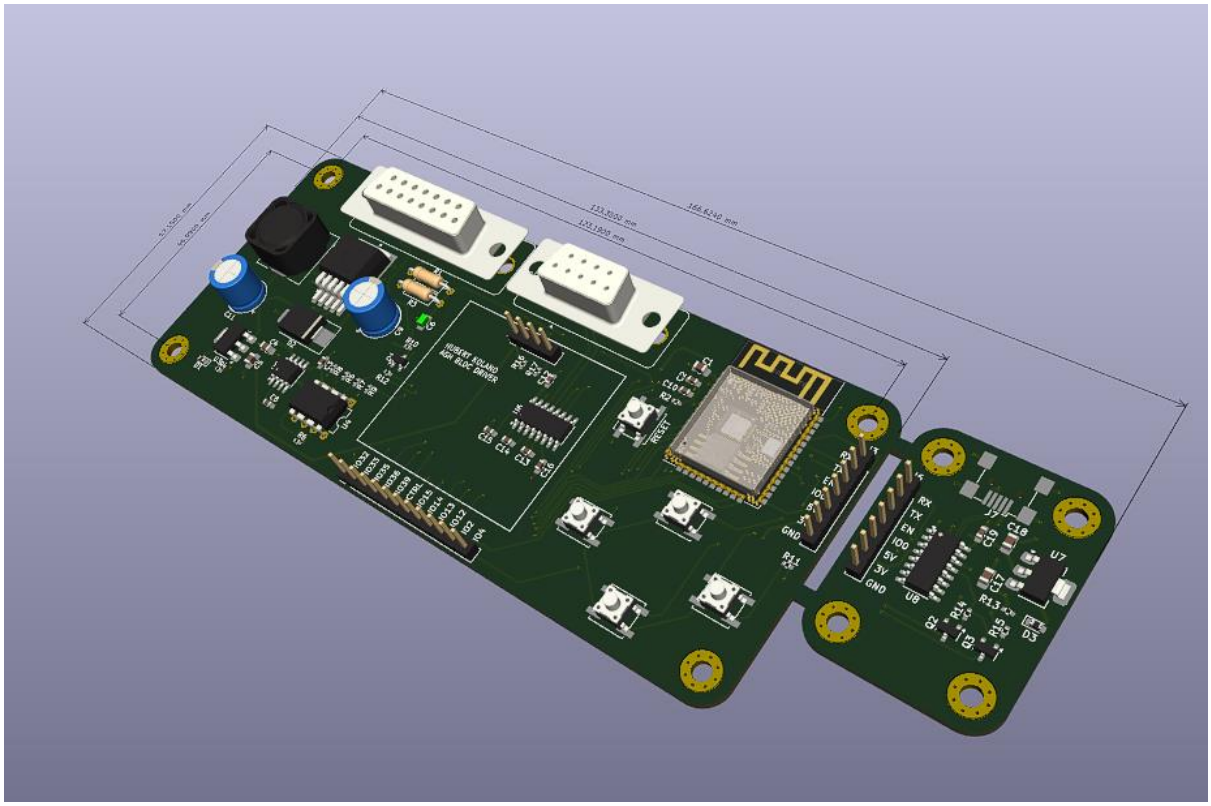
Po umiejscowieniu elementów na płytce, należy przystąpić do **prowadzenia połączeń** (routing) między nimi. Prowadzenie ścieżek to proces, w którym określa się przebieg sygnałów na płytce, łącząc wyprowadzenia poszczególnych elementów zgodnie z następującymi punktami:

- **Właściwa grubość ścieżek –** ścieżki, które wykonują połączenia zasilające, mogą mieć większy przekrój, aby uniknąć przepaleń się ścieżki.
- **Optymalizacja ścieżek -** ścieżki należy prowadzić jak najkrócej, minimalizując rezystancję oraz zakłócenia.
- **Reguły projektowe**: Ważne jest, aby ścieżki sygnałowe nie przecinały się i aby przestrzegać reguł projektowych (DRC – Design Rule Check) dotyczących odległości między ścieżkami, szerokości ścieżek i otworów.



Rys. 3.13 Zaprojektowany PCB

Spełniając wszystkie postawione wymagania powstał widoczny na 3.13 projekt PCB.

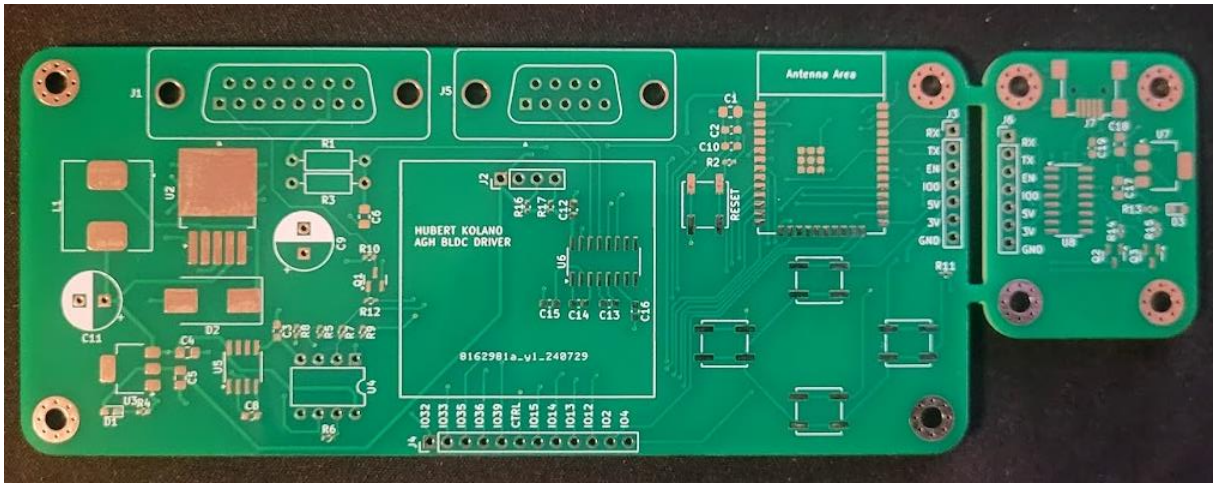


Rys. 3.14 Wizualizacja 3D wykonanego PCB

Program KiCad pozwala podejrzeć wizualizację 3D naszego projektu, tak jak na Rys. 3.14.

3.5. Fizyczne wykonanie PCB

Po zakończeniu projektowania PCB w KiCad, należy wygenerować pliki produkcyjne **gerber files**, które są standardowym formatem akceptowanym przez producentów PCB. Pliki te zawierają wszystkie informacje potrzebne do wyprodukowania płytki, w tym warstwy miedzi, maski lutowniczej i opisów elementów. Tak wykonane pliki wysyłamy do producenta, który sprawdza poprawność plików swoim oprogramowaniem, następnie przyjmuje nasze zamówienie.



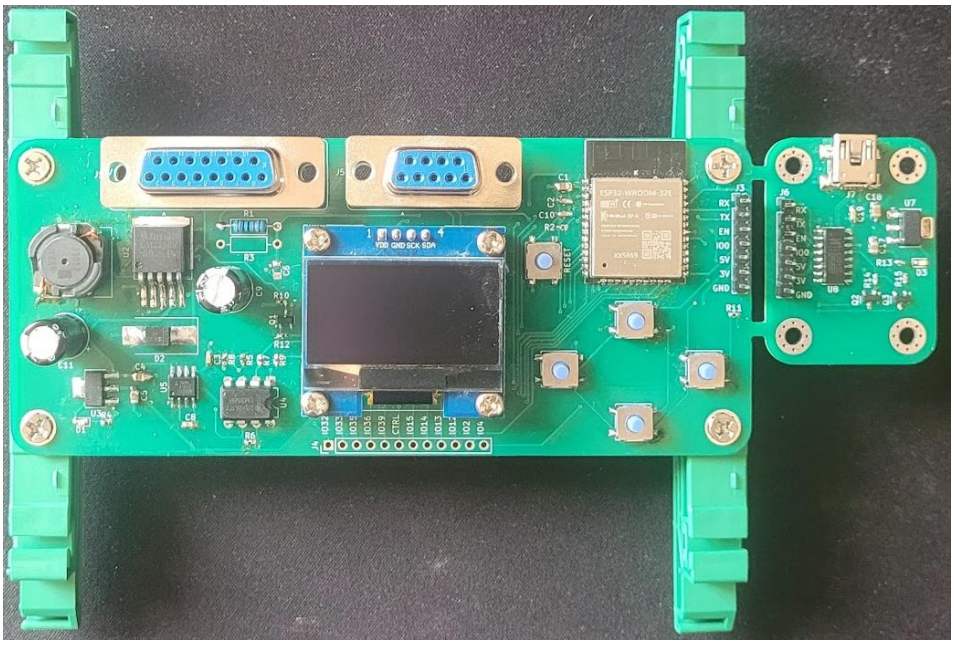
Rys. 3.15 Wygląd PCB po wypakowaniu z przesyłki.

Płytka wygląda bardzo dobrze, mimo to dobrą praktyką jest wykonać manualną inspekcje, czy wszystkie ścieżki są poprawnie wytrawione (nie widać przeskoku miedzi między nimi).

Jeżeli wszystko z nią w porządku, można zacząć wykonywać montaż elementów przy pomocy lutownicy. Elementy powinno się lutować od najmniejszego do największego. Należy zadbać o to, aby lut tworzył błyszczące i równomierne połączenie bez nadmiernej ilości cyny (tzw. zimne luty).

Rys. 3.16 Inspekcja płytki PCB pod szkłem powiększającym.

Po wykonaniu montażu sprawdzono za pomocą szkła powiększającego jakość lutów (Rys 3.16) oraz sprawdzono wyrywkowo miernikiem poprawność lutów.



Rys. 3.17 Wygląd w pełni zmontowanego PCB z przykręconymi uchwytami DIN.

Płytka z zamontowanymi elementami (Rys. 3.17) po inspekcji jest gotowa do programowania. Po podłączeniu do zasilania zapala diody oznaczające poprawność działania sekcji zasilania.

4. Planowanie i implementacja oprogramowania

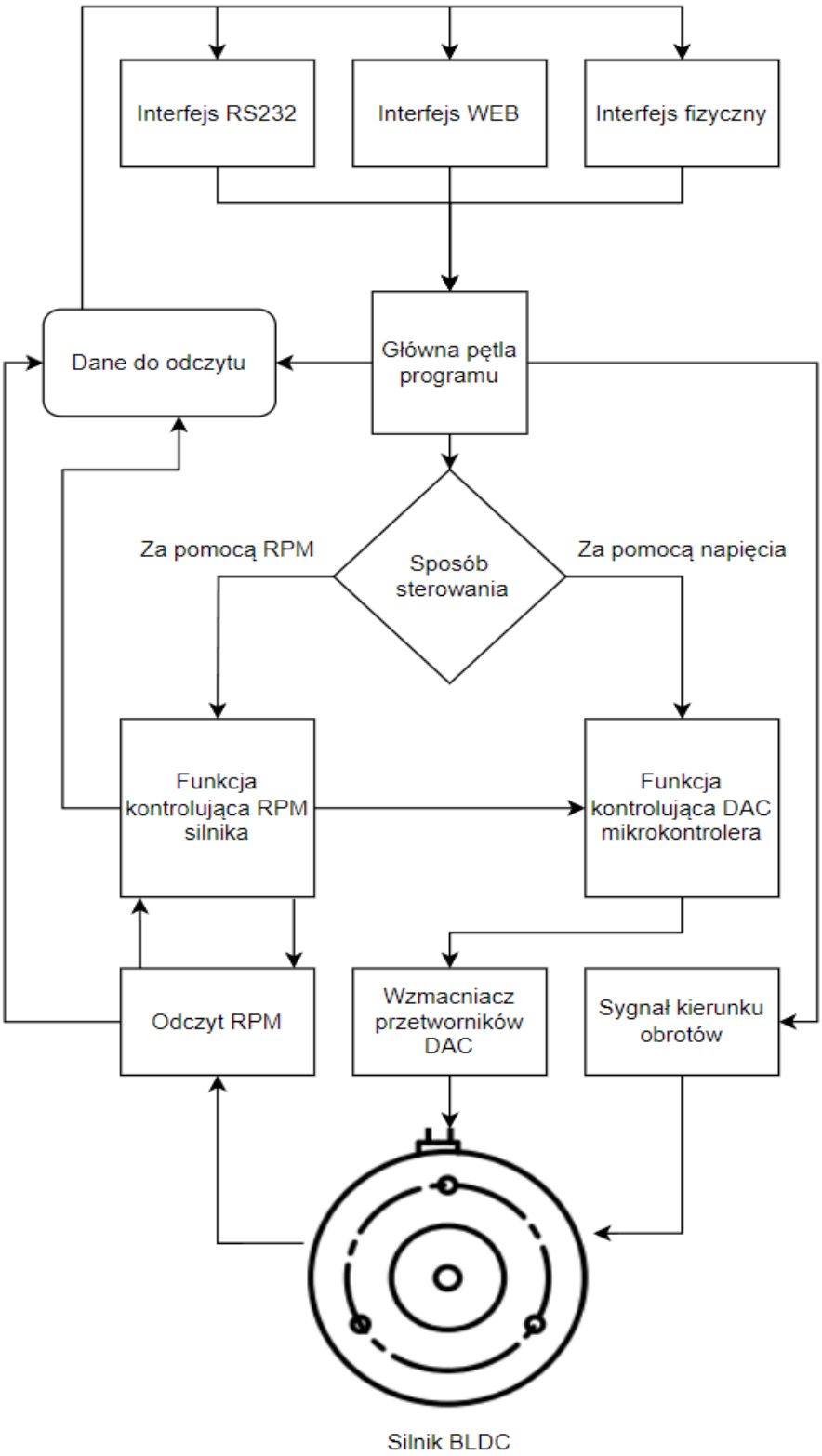
W tym rozdziale omówiony zostanie opisany krok po kroku proces wytwarzania oprogramowania dla mikrokontrolera. Dobrą praktyką przed rozpoczęciem prac nad kodem jest rozpisanie architektury oprogramowania, obejmująca główne moduły i funkcje odpowiedzialne za komunikację z silnikiem, obsługę interfejsów użytkownika oraz realizację funkcjonalności IoT. Przed rozpoczęciem pisania kodu należy również wybrać środowisko programistyczne, które będzie wykorzystywane do pisania i debugowania kodu.

4.1. Architektura oprogramowania

Projektowanie architektury oprogramowania przed rozpoczęciem prac nad jego kodowaniem pozwala na zrozumienie struktury systemu, zarządzanie jego złożonością oraz zapewnia możliwość wykrycia i eliminacji potencjalnych problemów we wczesnych fazach projektu. Dzięki takiemu podejściu zmniejsza się ryzyko wystąpienia poważnych błędów w fazie implementacji i testowania, co znacząco wpływa na jakość i stabilność ostatecznego produktu [20].

Na podstawie schematu blokowego (Rys. 4.1), można wywnioskować ważne wskazówki na temat pisania funkcji kodzie. Każdy z interfejsów będzie kontrolował tą samą funkcjonalność, więc dobrze jest napisać uniwersalne funkcje, który każdy z tych interfejsów będzie mógł wykorzystać. Silnikiem będzie można sterować za pomocą podania wartości RPM oraz napięcia jakie chcemy na wyjściu. Sterowanie za pomocą RPM będzie wymagało użycia pętli zwrotnej odczytu parametrów silnika i poprawienia parametrów zadanych – nie przypuszczamy, że silnik zwiększa liniowo obroty wraz ze wzrostem napięcia sterującego.

Poprzez „Dane do odczytu” rozumiemy zebrane parametry pracy (RPM zadane i odczytane, napięcie, kierunek), które mają być dostępne do wyświetlenia na każdym z interfejsów.

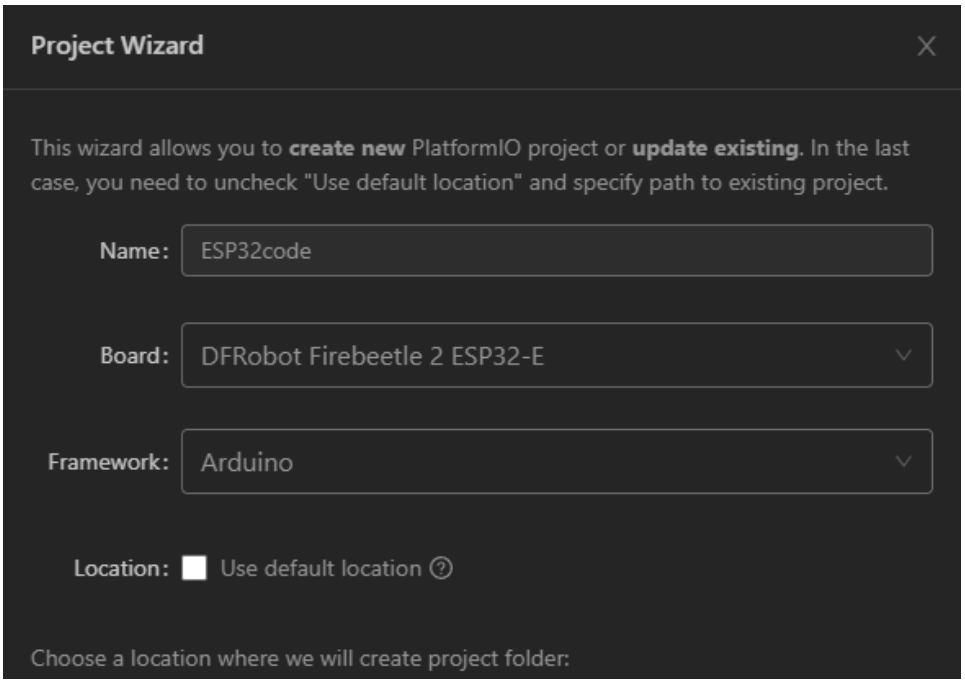


Rys. 4.1 Schemat blokowy działania oprogramowania.

4.2. Wybór i przygotowanie środowiska

W projekcie zdecydowano się na wykorzystanie **Visual Studio Code (VSCode)** jako środowisko programistyczne do pracy nad oprogramowaniem mikrokontrolera. VSCode to lekkie, szybkie i popularne środowisko z szerokim wsparciem społeczności oraz możliwością łatwego rozszerzania funkcjonalności za pomocą wtyczek.

PlatformIO to jedna z dostępnych wtyczek. pozwala na zarządzanie projektami mikrokontrolerów pozwalająca na łatwe utworzenie nowego projektu (Rys. 4.2). Wybór PlatformIO był podyktowany jego pełną integracją z **frameworkiem Arduino**, co ułatwia programowanie ESP32. Dzięki temu można korzystać z prostych i dobrze udokumentowanych bibliotek Arduino, przy jednoczesnym zachowaniu mocy i możliwości mikrokontrolera ESP32. PlatformIO oferuje również wbudowane narzędzia do kompilacji, debugowania i komunikacji USB - UART, co ułatwia testowanie i weryfikację kodu bez konieczności korzystania z innych narzędzi.



Rys. 4.2 Okno rozpoczynania projektu w rozszerzeniu PlatformIO.

Na urządzeniu, na którym wykonujemy prace wymagane jest zainstalowanie sterownika konwertera USB na UART – CH340C, dostępnego do pobrania ze strony producenta [21].

4.3. Pisanie kodu

Pisanie oprogramowania w frameworku Arduino odbywa się w języku C++, jest to język stworzony do pisania niskopoziomowego (mocno powiązanego ze sprzętem) co pozwala na pełną kontrolę nad peryferiami mikrokontrolera, co przekłada się na wykonywanie wydajnych aplikacji.

W trakcie pisania kodu, kluczowe jest stosowanie dobrych praktyk programistycznych, które zwiększają czytelność i utrzymanie kodu. Dla przykładu zaleca się stosowanie czytelnych nazw funkcji i zmiennych, które jasno określają ich funkcję w programie i są pisane zgodnie z przyjętą dla języka konwencją (np. zmienne i funkcje zawsze zaczynające się małą literą, kolejny wyraz zmiennej piszemy bez odstępu dużą literą). Cały kod jest pisany w języku angielskim.

Ważnym aspektem jest również rozdzielanie kodu na header files (pliki nagłówkowe). C++ jako język kompilowany musi tłumaczyć napisany przez nas kod na maszynowy za pomocą kompilatora. Korzystając z header files, kompilator może podejrzeć, które funkcje zostały dodane bądź zmienione i nie kompilować na nowo reszty kodu, co znacznie skróci proces kompilowania. Dodatkowo taki podział plików pozwala na modularność i lepszą organizację projektu.

W projekcie będziemy również często korzystać z dyrektywy *#define,* pozwala ona na lepszą organizacje i łatwiejszą manipulacje wielokrotnie wykorzystywanymi zmiennymi w projekcie. Dla przykładu obecnie zakładamy zakres sterowania napięciem w zakresie do 5000 mV, podczas pisania kodu często będziemy posługiwać się tą wartością. Istnieje jednak możliwość, że po testach będziemy chcieli zmienić wartość zakresu sterowania napięciem, zamiast zmieniać wszędzie w kodzie wpisaną na sztywno wartość „5000”, wystarczy edytować jedną dyrektywę #define [22].

Cały kod będzie również wykorzystywał zawartą przez producenta ESP32 dokumentacje techniczną dla Arduino framework, która zawiera gotowe biblioteki dla wielu funkcjonalności, co bardzo ułatwi nam pracę [23].

4.3.1. Zmienne globalne

W projekcie zakładamy cztery globalne zmienne, czyli takie, które możemy zmienić bądź odczytać z każdego miejsca w programie:

```
extern int voltageDACS;
extern int engineSetRPM;
extern bool engineDirection;
extern volatile int engineReadRPM;
```

Słowo „extern” oznacza możliwość rozpoznania tej zmiennej przez kompilator w każdym dołączonym pliku (używając #include).

Typ volatile dodajemy do zmiennej, której typ może się zmieniać niezależnie od wykonywanego kodu, jest to informacja dla kompilatora, w przeciwnym przypadku optymalizacja kodu przez kompilator może naruszyć działanie programu [22].

4.3.2. Funkcja sterująca przetwornikami DAC

```
void setCombinedDACOutput(int inputValue) {
    // input check
    if (inputValue < 0 || inputValue > DAC_MAX_VOLTAGE) {
        Serial.printf("Error: Input out of range. Value should be between 0
and %d.\n", DAC_MAX_VOLTAGE);
        return;
    }
    //mapping values, maxes out one dac before using second one
    int dac1Voltage, dac2Voltage;

    if (inputValue <= DAC_HALF_MAX_VOLTAGE) {
        dac1Voltage = inputValue;
        dac2Voltage = 0;
    } else {
        dac1Voltage = DAC_HALF_MAX_VOLTAGE;
        dac2Voltage = inputValue - DAC_HALF_MAX_VOLTAGE;
    }
    int dac1Value = map(dac1Voltage, 0, DAC_HALF_MAX_VOLTAGE, 0, 255);
    int dac2Value = map(dac2Voltage, 0, DAC_HALF_MAX_VOLTAGE, 0, 255);
```

```
    dacWrite(DAC1_PIN, dac1Value);
    dacWrite(DAC2_PIN, dac2Value);

    // Debugging info for UART
    Serial.printf("DAC written, Input: %d [mV] | DAC1: %d bits | DAC2: %d
bits\t\n", inputValue, dac1Value, dac2Value);
}
```

Wykorzystujemy funkcje map(), która skaluje nam podaną do niej zmienną, konwertując ją z pierwszego podanego zakresu, do drugiego. W naszym przypadku konwertujemy wartość mV ustalonych w projekcie na wartość bitów przetwornika DAC mikrokontrolera. Bity DAC są odzwierciedleniem w jakim stopniu na wyjściu DAC zostanie wyprowadzone napięcie zasilania (skala 0-255 dla 8 bitowego przetwornika).

Dla ustalenia wartości przetwornika używamy gotowej funkcji dacWrite() [23].

4.3.3. Funkcja kontrolująca silnik nastawą RPM

Regulacja PID to najpopularniejsza technika sterowania układami ze sprzężeniem zwrotnym. Regulacja ta opiera się na sprawdzaniu odchyłu wartości zadanej od wartości odczytanej i poprzez wyznaczone wcześniej parametry, dobieranie nastawy tak aby skutecznie osiągnąć wartość zadaną.

PID Controller

Command to Device		Proportional		Integral		Derivative		Bias (to prevent output being 0)
$output(t)$	$=$	$(K_P * e(t))$	$+$	$(K_I * \int_0^t e(t)d_t)$	$+$	$(K_D * \frac{d}{dt}e(t))$	$+$	$bias$
$output$	$=$	$(K_P * e)$	$+$	$(K_I * (K_I\_prior + e * iteration\_time))$	$+$	$(K_D * \frac{e - e\_prior}{iteration\_time})$	$+$	$\cdot bias$

Rys. 4.3 Wzór na nastawę parametrów regulatora PID [24]

Znając wzór na sposób działania regulatora PID (Rys. 4.3), można napisać kod funkcji, która odzwierciedli jego działanie:

```
void regulateRPMWithPID() {
    error = engineSetRPM - engineReadRPM;

    integral += error * DELTA_TIME_PID/1000;

    derivative = (error - previousError) / DELTA_TIME_PID/1000;

    output = (PID_Kp * error) + (PID_Ki * integral) + (PID_Kd * derivative);
    Serial.printf("error: %f integral: %f deriative: %f", error, integral,
derivative);
    previousError = error;

    output = constrain(output, DAC_START_ENGINE_VOLTAGE, DAC_MAX_VOLTAGE);

    voltageDACS = output;
    setCombinedDACOutput(voltageDACS);
   }
```

Podczas testów na silniku, należy będzie dobrać odpowiednie nastawy Kp, Ki oraz Kd, tak, aby jak najszybciej osiągnąć sygnał zadany i stabilnie go utrzymywać.

4.3.4. Prowadzenie ciągłego odczytu obrotów

W projekcie mikrokontroler wykonuje ciągły pomiar sygnału impulsowego wysyłanego przez silnik. Ważnym jest niedopuszczenie, aby wywołanie przerwań, czy inne czynności wykonywane na sterowniku (np. związane z obsługą interfejsów) nie naruszyły ciągłego pomiaru sygnału impulsowego. Z tego powodu, zdecydowano się dedykować jeden rdzeń mikrokontrolera wyłącznie w celu odczytu obrotów.

Główna pętla programu w środowisku Arduino wykonuje się na rdzeniu nr.1 (licząc od zera). Istnieje możliwość zaimplementowania drugiej pętli, która będzie działała niezależnie i mogła korzystać z tych samych peryferiów mikrokontrolera. W tym celu została wykorzystana biblioteka ESP32 SoC, która umożliwia korzystanie systemu FreeRTOS. Za jego pomocą można utworzyć proces, który będzie działa jako pętla wykonująca nasz kod [25].

Wyznaczony odpowiedni pin, jest ustawiony jako aktywujący przerwanie wysokim zboczem, następnie za każdym zliczeniem inkrementuje on wartość, która co sekundę jest kopiowana do zewnętrznej zmiennej, używanej w reszcie projektu, a następnie resetowana.

```
 TaskHandle_t Task1;

void IRAM_ATTR onPulse() {
    pulseCount++;
}
void setupEngineRPM() {
    pinMode(34, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(34), onPulse, RISING);

    xTaskCreatePinnedToCore(
        engineRPMTask,          // Task function
        "EngineRPMTask",        // Task name
        4096,                   // Stack size (bytes)
        NULL,                   // Parameters
        1,                      // Priority
        &Task1,                 // Task handle
        0                       // Core ID (Arduino main loop runs on core 1)
    );
}

void engineRPMTask(void *pvParameters) {
    unsigned long previousMillis = 0;

    while (true) {
        unsigned long currentMillis = millis();
        TIMERG0.wdt_wprotect=TIMG_WDT_WKEY_VALUE;
        TIMERG0.wdt_feed=1;
        TIMERG0.wdt_wprotect=0;

        if (currentMillis - previousMillis >= INTERVAL_READOUT) {
            previousMillis = currentMillis;
            // Update engineRPM with the count of pulses in the last second
taking engine PPR into consideration
            engineReadRPM = pulseCount / ENGINE_PPR;
            pulseCount = 0;
        }
    }
}
```

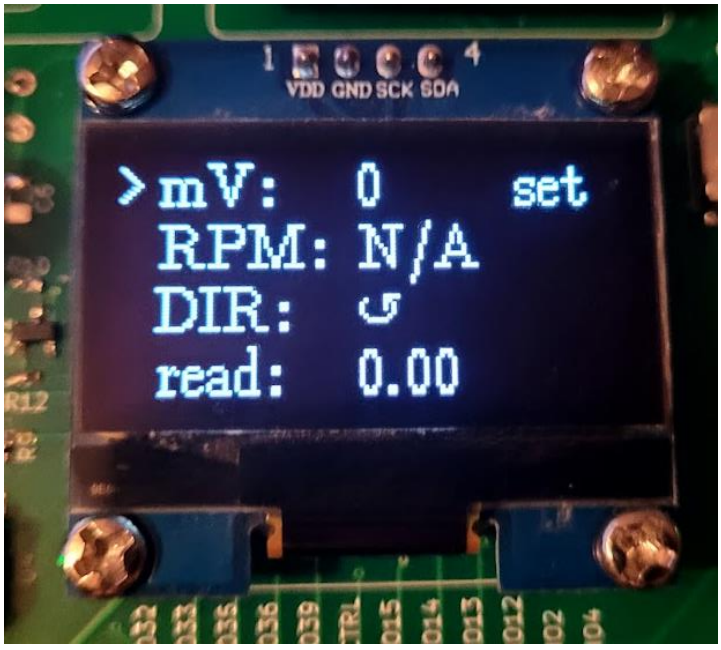
4.3.6. Interfejs UART oraz RS232

W projekcie w celach realizacji komunikacji RS232 wykorzystano moduł **MAX3232**, co pozwala na wykorzystanie UART do wymiany danych z urządzeniami laboratoryjnymi lub innymi systemami zewnętrznymi. Mikrokontroler pozwala na wykorzystanie aż trzech interfejsów UART naraz, można więc w łatwy sposób zaimplementować również komunikacje po USB, wysyłając i zaczytując wszystkie wiadomości na obydwu interfejsach UART. Komunikacja z mikrokontrolerem będzie przypominała wybór opcji z menu, a następnie możliwość wpisania/odczytania wybranej przez nas wartości.

```
void initSerial(){
    Serial.begin(115200);
    Serial2.begin(115200, SERIAL_8N1, 16, 17);
    sendToBothUarts("Welcome to the BLDC interface!\n");
}
void sendToBothUarts(String message) {
  Serial.print(message);  // UART0
  Serial2.print(message); // UART2
}
void serialMenuMessage(){
    sendToBothUarts("Please choose an option:\n");
    sendToBothUarts("1 - Input voltage value\n");
    sendToBothUarts("2 - Input RPM set value\n");
    sendToBothUarts("3 - Change engine direction\n");
    sendToBothUarts("4 - Check voltage\n");
    sendToBothUarts("5 - Check RPM read from engine\n");
    sendToBothUarts("6 - Check Wi-Fi connection\n");
    sendToBothUarts("7 - Enter SSID and password\n");
}
String readFromBothUarts() {
  String input = "";

  if (Serial.available()) {
    input = Serial.readStringUntil('\0');
  }
  if (input.length() == 0 && Serial2.available()) {
    input = Serial2.readStringUntil('\0');
  }
  input.trim();
  return input;
}
```

4.3.7. Zaprogramowanie obsługi ekranu OLED

W celu wyświetlania odpowiednich napisów na ekranie, posłużono się gotową biblioteką u8g2 [26]. Biblioteka pozwala na zrobienie bufora na polu 128 x 64 pikseli, następnie na załadowanie go. Bufor generowany jest przy pomocy konwersji tekstu na piksele przy pomocy gotowych bibliotek i wpisaniu koordynatów startu zapisywania buforu. Wybrana została czcionka, która idealnie wypełni ekran przy 4 linijkach tekstu, oraz pozwoli na dodanie znaku specjalnego kierunku obrotów. Zmienianie tekstu na ekranie, można wykonywać za pomocą nadpisanie bufora pustym tekstem i kolejnym jego nadpisaniu tekstem docelowym.



Rys. 4.4 Wygląd zaprogramowanego ekranu OLED.

Ekran OLED komunikuje się z mikrokontrolerem za pomocą interfejsu I2C. Odświeżane są wyświetlane przez niego nastawy od razu po ich zmianie, natomiast odczyt prędkości obrotowej co sekundę. Ekran z wszystkimi nastawami na 0 (domyślne podczas uruchomienia) prezentuje się tak jak na Rys. 4.4 Wygląd zaprogramowanego ekranu OLED.

Kod pokazujący załadowanie bufora przywitalnego oraz kod zmiany wyświetlanej wartości napięcia:

```
void welcomeOLED(int voltage, int rpmSet, bool direction, int rpmRead) {
    u8g2.clearBuffer();
    u8g2.drawStr(0 , LINE1 , ">");
    u8g2.drawStr(COLUMN1 , LINE1 , "mV: ");
    u8g2.drawStr(COLUMN1 , LINE2 , "RPM: ");
    u8g2.drawStr(COLUMN1 , LINE3 , "DIR:");
    u8g2.drawStr(COLUMN1 , LINE4 , "read:");
    char voltageStr[5];
    sprintf(voltageStr, "%d", voltage);

    u8g2.drawStr(COLUMN2 , LINE1, voltageStr);

    u8g2.drawStr(COLUMN2 , LINE2, rpmSet == 0 ? "N/A" : formatRPM(rpmSet));

    u8g2.drawUTF8(COLUMN2 ,LINE3, direction ? "↻" : "↺");
    u8g2.drawStr(COLUMN2 , LINE4, formatRPM(rpmRead));

    u8g2.drawStr(COLUMN3 , LINE1, "set");

    u8g2.sendBuffer();
}

void OLEDvoltage(int voltage){
    char voltageStr[5];
    sprintf(voltageStr, "%d", voltage);
    u8g2.drawStr(COLUMN2 , LINE1, "       ");
    u8g2.drawStr(COLUMN2 , LINE1, voltageStr);
    u8g2.sendBuffer();
}
```

4.3.8. Implementacja obsługi przycisków

Przyciski typu tact switch, przy zmianie stanu, przez maksymalnie parę milisekund wpadają w oscylacje, co mikrokontroler może odebrać jako kilka wciśnięć przycisku, skutkując w nieprzyjemnej obsłudze interfejsu (więcej naciśnięć przycisku zarejestrowanych niż faktycznie wykonanych). Przeciwdziała się temu zjawisku, poprze równoległe dolutowanie kondensatora, co stłumi bardzo gwałtowne zmiany napięcia lub programując nieczułość na wciśnięcia kilka milisekund po zarejestrowanym wciśnięciu i puszczeniu przycisku. W projekcie sterownika zdecydowana się na drugie rozwiązanie. Programując detekcje wciśnięcia, wykonano również detekcje przytrzymania przycisku, co pozwoli na o wiele szybsze zmiany nastaw za pomocą interfejsu fizycznego.

```
struct Button {
    uint8_t pin;                    // Button GPIO pin
    void (*PressCallback)();    // Function pointer for short press action
    bool isPressed;                  // Button press status
    bool longPressTriggered;         // Has the long press been triggered?
    unsigned long lastPressTime;     // Last time the button was pressed
    unsigned long pressStartTime;    // When the button was first pressed
    unsigned long lastLongPressActionTime; // Last time the long press action
was performed
};
void handleButtonPress(Button &button) {
    unsigned long currentTime = millis();
    bool buttonPressed = digitalRead(button.pin) == LOW;
    if (buttonPressed && !button.isPressed && currentTime -
button.lastPressTime > DEBOUNCE_TIME) {
        button.isPressed = true;
        button.pressStartTime = currentTime;
        button.longPressTriggered = false;
        button.lastLongPressActionTime = currentTime;
        button.lastPressTime = currentTime;
            button.PressCallback();
    }
```

```
    if (buttonPressed && button.isPressed) {
        if (!button.longPressTriggered && currentTime - button.pressStartTime
> LONG_PRESS_TIME) {
            button.PressCallback();
            button.longPressTriggered = true;
        } else if (button.longPressTriggered && currentTime -
button.lastLongPressActionTime > HOLD_PRESS_INTERVAL) {
            button.PressCallback();
            button.lastLongPressActionTime = currentTime;
        }
    }
}
```

W programie napisany został typ danych specjalny dla przycisków, pozwalający dla każdego z osobna mierzyć jego czas wciśnięcia oraz przypisać do niego wykorzystywaną funkcję.

4.3.9. Łączność bezprzewodowa za pomocą strony WEB.

W projekcie zastosowano mikrokontroler ESP32 do realizacji łączności bezprzewodowej za pomocą sieci Wi-Fi. Do obsługi tej komunikacji wykorzystano popularne biblioteki ESP32 przygotowane przez Espriff (domyślnie pobierane wraz z frameworkiem): WiFi., ESPAsyncWebServer, oraz AsyncTCPktóre umożliwiają stworzenie serwera webowego obsługującego żądania HTTP w sposób asynchroniczny.

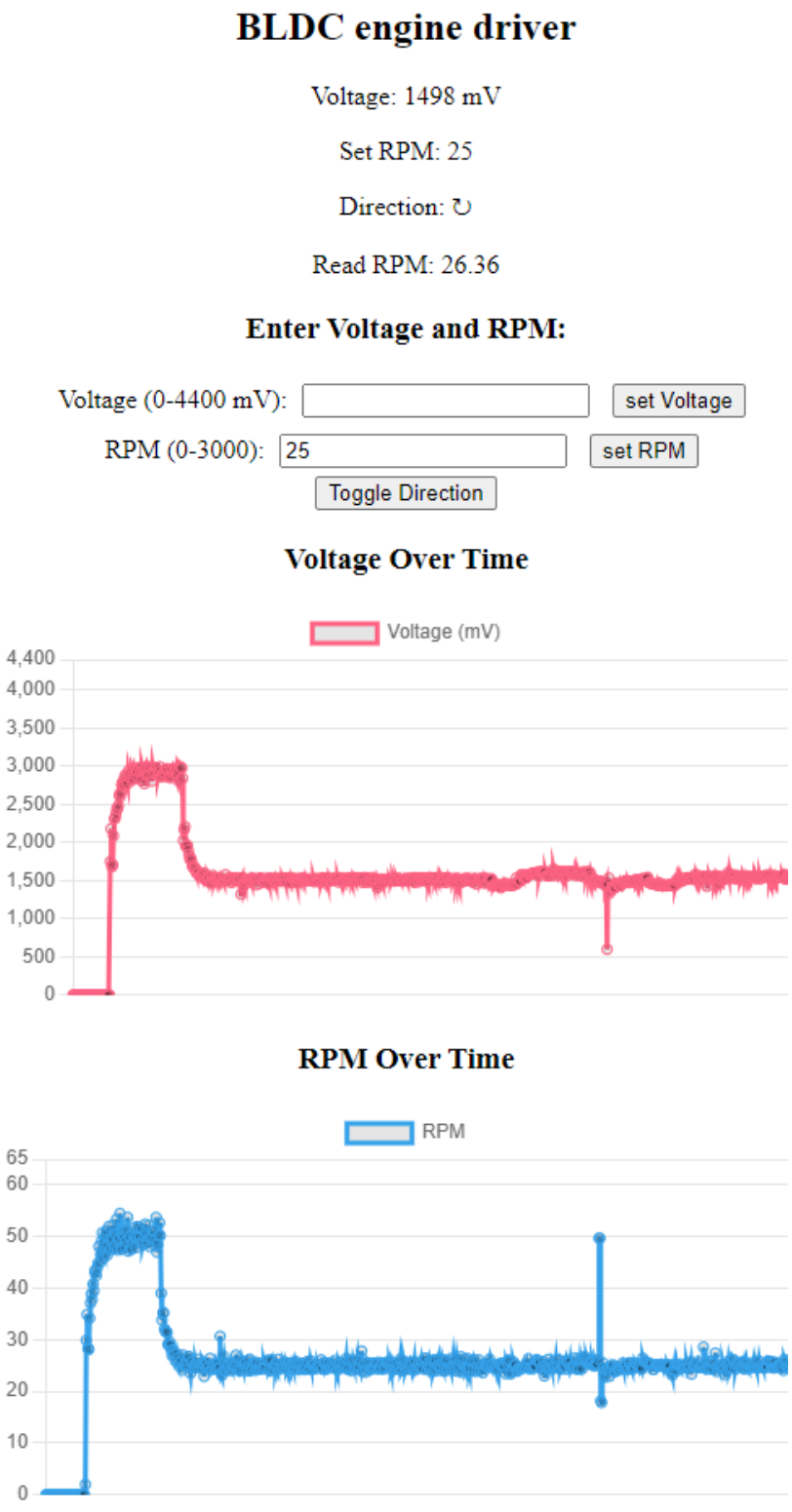
Implementacja rozpoczęła się od skonfigurowania połączenia Wi-Fi za pomocą biblioteki WiFi, która pozwala na połączenie ESP32 do lokalnej sieci bezprzewodowej. Następnie, wykorzystując bibliotekę ESPAsyncWebServer.h, stworzono asynchroniczny serwer webowy, który umożliwia odbieranie i przetwarzanie żądań HTTP od klientów, takich jak przeglądarki internetowe. Serwer webowy pozwala na kontrolowanie i monitorowanie pracy sterownika za pomocą przeglądarki poprzez interfejs użytkownika (UI) dostępny w domenie web.

Biblioteka AsyncTCP.h wspiera asynchroniczne zarządzanie połączeniami TCP, co zapewnia szybką i niezawodną komunikację przy minimalnym wpływie na zasoby mikrokontrolera. Dzięki zastosowaniu tej metody, możliwe było stworzenie responsywnego i stabilnego rozwiązania do zdalnego sterowania i monitorowania parametrów pracy silnika BLDC w ramach projektu sterownika z funkcjonalnością IoT.

```
void initWIFI(String ssid, String password) {
    if (ssid.length() > 0 && password.length() > 0) {
    Serial.print("Connecting to ");
    Serial.println(ssid);
    }
    else{
        Serial.print("No WiFi password detected \n");
        return;
    }
  // Connect to Wi-Fi
   WiFi.begin(ssid, password);
   int i = 0;
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
    if (i++ > 5){
        Serial.println("Failed connecting to WiFi");
        return;
    }}
  Serial.println("Connected to WiFi");
  Serial.println(WiFi.localIP());
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });
  server.on("/getData", HTTP_GET, [](AsyncWebServerRequest *request){
     String json = "{\"voltage\":" + String(voltageDACS) +
                   ",\"setRPM\":" + (turnEngineControlPID ?
formatRPM(engineSetRPM) : "\"voltage control\"") +
                   ",\"direction\":\"" + (engineDirection ? "↻" : "↺") +
                   "\",\"readRPM\":" + formatRPM(engineReadRPM) +
                   "}";
    request->send(200, "application/json", json);
  });
  // Endpoint to handle form submissions
 server.on("/submitRPM", HTTP_POST, [](AsyncWebServerRequest *request){
    if (request->hasParam("rpm", true)) {
      int enteredRPM = request->getParam("rpm", true)->value().toInt();
      if (enteredRPM > ENGINE_MAX_RPM/ENGINE_TORQUE || enteredRPM < 0 ){
            request->send(200, "text/plain", "Enter correct RPM");
      }
      else {
          turnOnRegulationPID(enteredRPM * ENGINE_TORQUE);  // Only update the
RPM
```

```
      }
    }
  });
  server.on("/submitVoltage", HTTP_POST, [](AsyncWebServerRequest *request){
    if (request->hasParam("voltage", true)) {
      float enteredVoltage = request->getParam("voltage", true)-
>value().toFloat();
      if (enteredVoltage > DAC_MAX_VOLTAGE || enteredVoltage < 0 ){
         request->send(200, "text/plain", "Enter correct voltage");
      }
      else {
        voltageDACS = enteredVoltage;
        setCombinedDACOutput(enteredVoltage);  // Only update the Voltage
        turnOffRegulationPID();
      }

    }
  });

  server.on("/toggleDirection", HTTP_GET, [](AsyncWebServerRequest *request){
    changeDirection();
  });

  server.begin();
}
```

Do powyższego kodu należało również wykonać stronę w języku HTML i przekazać ją do funkcji send_P(); Aby rozpocząć łączność bezprzewodową, należy również przekazać jednej z metod klasy Wifi, SSID oraz hasło sieci bezprzewodowej do której ma się połączyć mikrokontroler. W tym celu napisano również funkcje, które są odpowiedzialne za zapisywanie oraz odczytywanie tych danych z pamięci EEPROM, co umożliwi łatwą zmianę hasła za pomocą np. interfejsu UART. Łączenie z siecią Wi-Fi następuje automatycznie przy każdym rozruchu. Aby połączyć się z mikrokontrolerem wystarczy dowolne urządzenie obsługujące przeglądarkę internetową, następnie należy być podłączonym do tej samej sieci i wpisać jako domenę adres IP przydzielony mikrokontrolerowi (jest również podawany po UART w trakcie rozruchu). Znajdziemy stronę na której możemy odczytać wszystkie parametry pracy silnika, samemu je zmienić, oraz zobaczyć na wykresie jak zmieniały się one w czasie przez ostatnie 15 min, tak jak na Rys. 4.5



Rys. 4.5 Widok strony internetowej sterownika BLDC podczas testów.

5. Testowanie sterownika

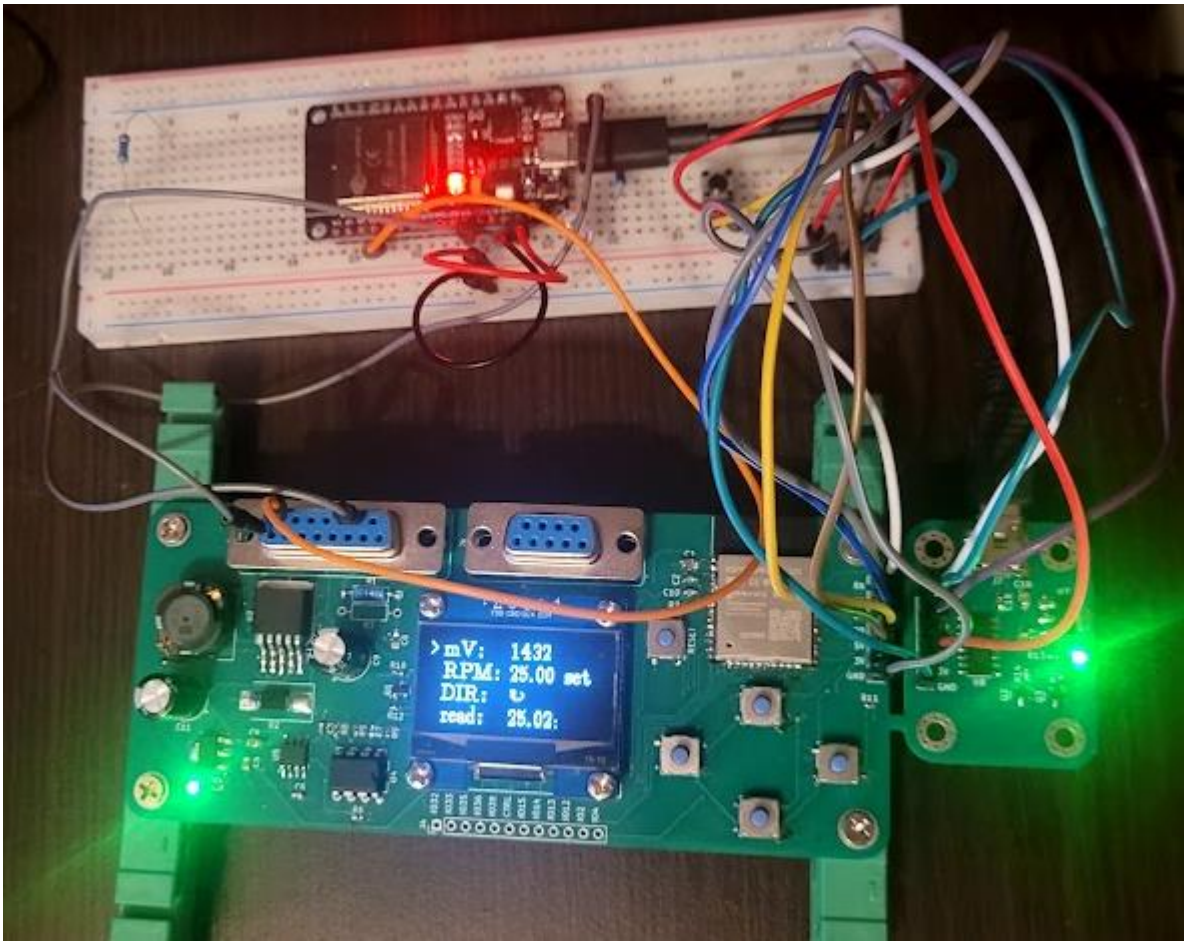
Testowanie sterownika było kluczowym etapem projektu, mającym na celu weryfikację poprawności działania zarówno hardware'u, jak i software'u. Proces testowania obejmował sprawdzenie komunikacji przez wszystkie zaimplementowane interfejsy, jak i poprawność sterowania silnikiem BLDC po wykonaniu nastaw odpowiadających za regulację pracy.

5.1. Testy wstępne

Jako sposób sprawdzenia poprawności działania płytki postanowiono na wykorzystanie innego mikrokontrolera, w tym przypadku płytki prototypowej ESP32, która została tak zaprogramowana, żeby symulować pracę silnika.

```
void setup() {
  if (!ledcSetup(PWM_CHANNEL, PWM_BASE_FREQ, PWM_RESOLUTION)){
    Serial.println("Failed to setup PWM");
  }
  ledcAttachPin(PWM_PIN, PWM_CHANNEL);
  ledcWrite(PWM_CHANNEL, 64);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= TIME_INTERVAL) {
    previousMillis = currentMillis;
    // Read the analog value (range from 0 to 4095)
    int analogValue = analogRead(ANALOG_PIN);

    int mappedFreq = map(analogValue, 0, 4095, 0, PWM_BASE_FREQ);

    if (mappedFreq > 0) {
    ledcChangeFrequency(PWM_CHANNEL, mappedFreq, PWM_RESOLUTION);
    ledcWrite(PWM_CHANNEL, 64);
    } else {
    ledcWrite(PWM_CHANNEL, 0);
    }
  }
}
```

Mikrokontroler na płytce prototypowej został zaprogramowany w ten sposób, aby otrzymane napięcie analogowe na przetworniku ADC (analogowo – cyfrowym) zapisywał. Na podstawie zapisu mapuje częstotliwość z jaką podaje sygnał PWM na jednym z wyjść. Cykl powtarza się co sekundę. Odpowiednie piny naszego sterownika podpięte zostały za pomocą płytki stykowej do płytki prototypowej, wraz z zasilaniem



Rys. 5.1 Płytka podłączona w procesie wstępnych testów.

Płytka podłączona tak jak na Rys. 5.1 jest w stanie symulacji pracy z silnikiem.

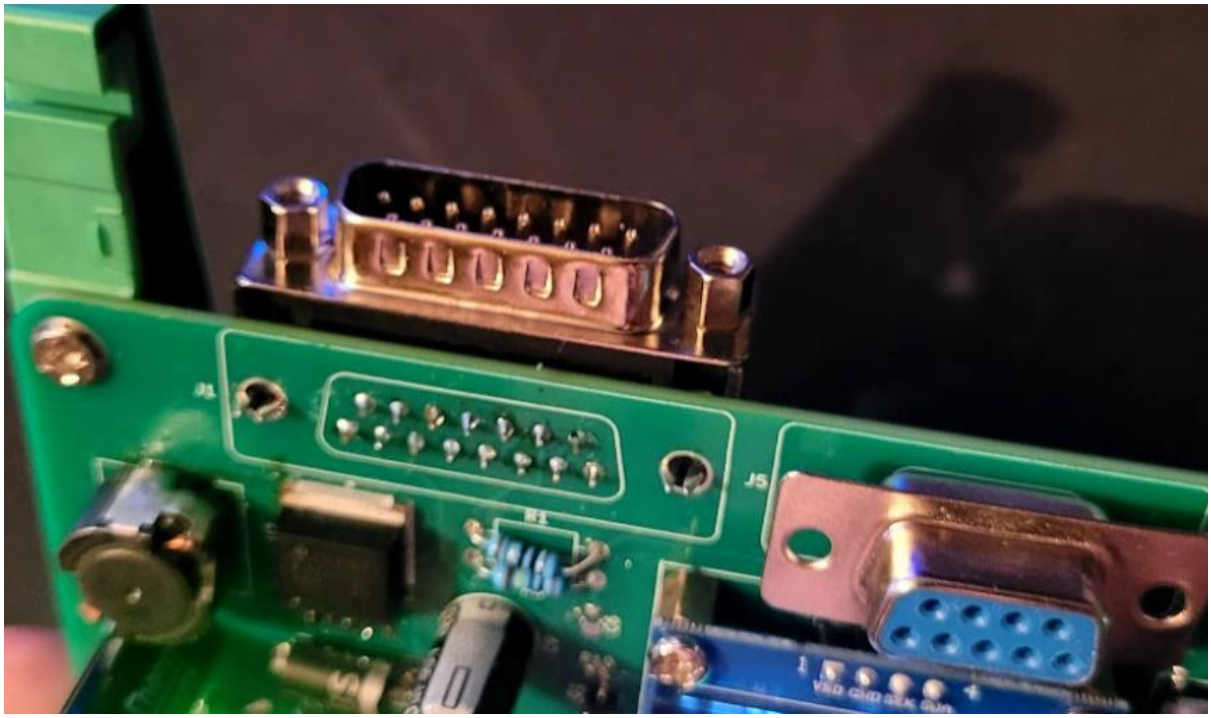
```
Connecting to WiFi...
Connected to WiFi
192.168.1.100
Please choose an option:
1 - Input voltage value
2 - Input RPM set value
3 - Change engine direction
4 - Check voltage
5 - Check RPM read from engine
6 - Check Wi-Fi connection
7 - Enter SSID and password
8 - Show Menu
error: 750.000000 integral: 750.000000 deriative: 0.000750DAC written, Input: 525 [mV] | DAC1: 60 bits | DAC2: 0 bits
error: 523.000000 integral: 1273.000000 deriative: -0.000227DAC written, Input: 516 [mV] | DAC1: 59 bits | DAC2: 0 bits
error: 328.000000 integral: 1601.000000 deriative: -0.000195DAC written, Input: 484 [mV] | DAC1: 56 bits | DAC2: 0 bits
error: 332.000000 integral: 1933.000000 deriative: 0.000004DAC written, Input: 552 [mV] | DAC1: 63 bits | DAC2: 0 bits
error: 318.000000 integral: 2251.000000 deriative: -0.000014DAC written, Input: 609 [mV] | DAC1: 70 bits | DAC2: 0 bits
error: 269.000000 integral: 2520.000000 deriative: -0.000049DAC written, Input: 638 [mV] | DAC1: 73 bits | DAC2: 0 bits
```

Rys. 5.2 Widok wpisów komunikacji komputera ze sterownikiem po USB.

Następnie sterując dowolnym z interfejsów wybieramy różne nastawy i obserwujemy czy nasze pomiary są zgodne z zakładanymi. Rys. 5.2 przedstawia wpisy sterownika wysyłane za pomocą interfejsu UART na USB, przedstawiają one zmianie nastaw parametrów i odczytów. Identyczne odczyty możemy znaleźć na ekranie OLED.

5.2. Testy w laboratorium

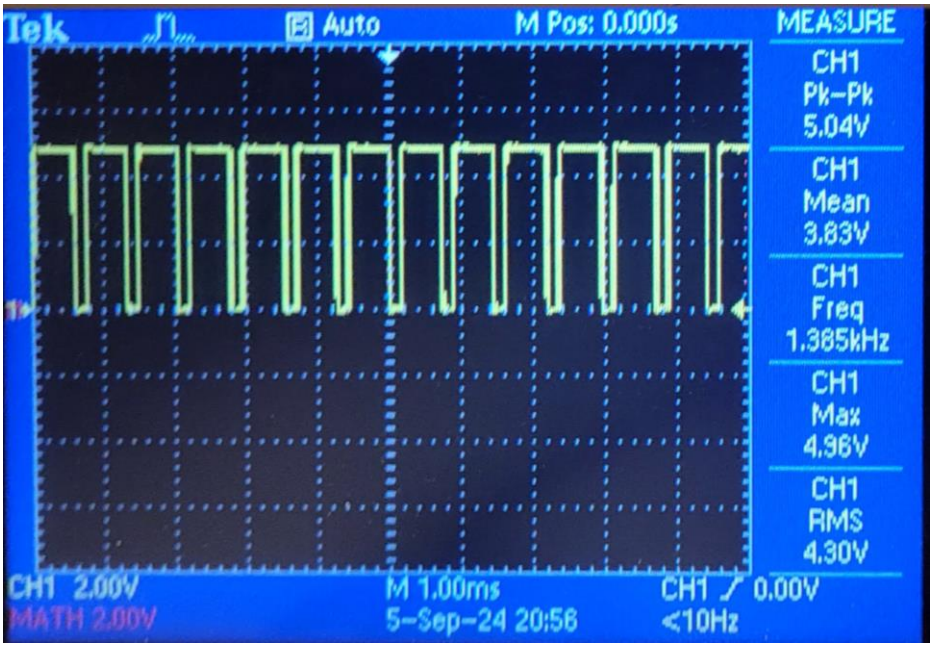
Przed podłączeniem sterownika do wtyczki silnika, sprawdzono wszystkie wyprowadzenia we wtyczce, w obawie przed umiejscowieniem w innym niż deklarowanym miejscu napięcia 24V, co mogłoby uszkodzić sterownik. Wyprowadzenia wtyczki pokryły się z założeniami, jednak zarówno kabel jak i sterownik posiadały gniazdo DB15 typu żeńskiego. Wykonano modyfikację zamiany gniazda w sterowniku na typ męski. Taka zamiana daje na wyprowadzeniu wtyczki odbicie lustrzane wyprowadzeń pinów. Gniazdo dolutowano z tyłu sterownika, tak jak na Rys. 5.3 celem ponownego odwrócenia pinów i uzyskania tego samego układu wyprowadzeń.



Rys. 5.3 Zamontowane nowe gniazdo DB15

Następnie wykonano pomiary dotychczasowego układu sterowania silnikiem, w celu potwierdzenia sposobu sterowania nim oraz napięcia sygnału impulsowego.

Rys. 5.4 Sposób podłączenia probówki oscyloskopu pod układ sterowania silnikiem.



Rys. 5.5 Pomiary sygnału impulsowego silnika za pomocą oscyloskopu.

Poprzez rozebranie dotychczasowego układu i podłączenie się do niego oscyloskopem, w sposób taki jak na Rys. 5.4 można było uzyskać kluczowe informacje, takie jak napięcie szczytowe sygnału impulsowego o wartości 5V, oraz zakres napięcia pracowania silnikiem o wartościach 0 – 4V, przy czym rozpoczyna on prace od 0,9V.

Na Rys. 5.5 widnieje sygnał silnika, kiedy jest wysterowany maksymalnym napięciem. Okazuje się, że przy maksymalnych obrotach silnik generuje częstotliwości o wartości 2,1kHz, zamiast zakładanych 108kHz. Wykonano modyfikacje w kodzie wartość dyrektywy #define ENGINE_PPR z 36 na 0.7, celem zachowania funkcjonalności sterownika.

W dzielnik napięcia przygotowany pod większe napięcie sygnału (rozdział 3.3.4 Odczyt prędkości i sterowanie kierunkiem) modyfikujemy zamieniając pierwszy rezystor na zworę. Rezystor podciągający do masy postanowiono zastąpić rezystorem o wartości 1MΩ, ze względu na możliwość odprowadzenia szumów, które generują fałszywe dane, kiedy np. silnik jest wyłączony i może wpływać na odczyt podczas pracy silnika.

Dodatkowo zauważono też, że układ ILC7660 generuje napięcie ujemne na poziomie -4,5V a nie zakładanych -5V (prawdopodobnie wywołane za dużym obciążeniem przy dużych wzmocnieniach). Zakłóca to pracę wzmacniacza, gdyż wzmacniacz odwracający sumujący, nie może podać do inwertera niższego niż -4,5V. Postanowiono rozwiązać ten problem zmieniając rezystor sprzężenia zwrotnego inwertera na inny o dwa razy większej wartości (200kΩ) powoduje to dwukrotne wzmocnienie i odwrócenie sygnału, teraz nie ma potrzeby podawać niższego napięcia niż -2V, gdyż maksymalne jakie jest potrzebne to 4V. Korzystając ponownie ze wzoru (3), dobrano rezystor 34kΩ. Co skutkuje sterowaniem na wyjściu zakresem 0-4V.

Wykonanie pomiarów silnika przed rozpoczęciem projektu pozwoliłoby na uniknięcie wykonywania modyfikacji, a planowanie samego projektu nie wymagałoby rozważania potencjalnych potrzeb, tylko te uzasadnione. Tabela 3 zawiera zestawienie parametrów rzeczywistych oraz tych zakładanych na podstawie dokumentacji.

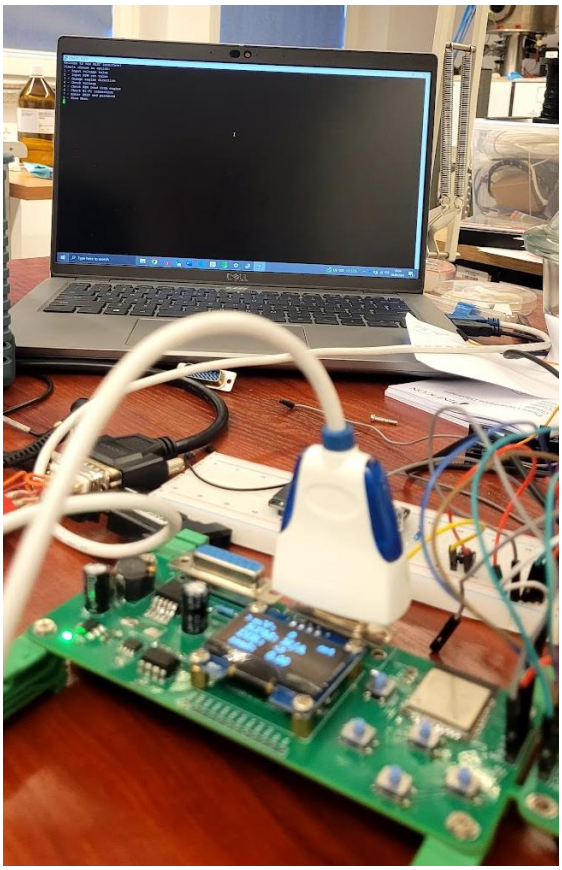
Tabela 3 Zestawienie parametrów silnika wywnioskowanych na podstawie dokumentacji i faktycznie zmierzonych

Wartość	Wywnioskowana z dokumentacji	Faktycznie zmierzona
Zakres napięcia sterowania	**0 - 5V**	**0,9V - 4V**
Maksymalna częstotliwość sygnału impulsowego	**108kHz**	**2,1kHz**
Napięcie szczytowe sygnału impulsowego	**24V lub 5V**	**5 V**

Po wykonanych wszystkich pomiarach i modyfikacjach podłączono wtyczkę silnika do sterownika i zaczęto kalibracje nastaw układu PID. Optymalne nastawy członów regulatora proporcjonalno-całkująco-różniczkującego wyznacza się w procesie jego strojenia. W projekcie wykonamy strojenie metodą prób i błędów

Zaczęto od wyzerowania członów całkującego i różniczkującego, następnie wzmocnienie członu proporcjonalnego stopniowo zwiększano się, aż do zauważenia oscylacji. W miarę zwiększania wzmocnienia członu proporcjonalnego regulacja będzie szybsza, lecz równocześnie łatwo jest doprowadzić do niestabilności, na co nie można sobie w tym przypadku pozwolić.

Po ustawieniu wzmocnienia członu P trzeba zwiększać wkład członu I, aby zmniejszyć oscylacje. Ten składnik równania regulatora opisującego algorytm PID zmniejsza błąd stanu ustalonego, ale też zwiększa przeregulowanie. Przeregulowanie jest zazwyczaj warunkiem szybkiej reakcji. Człon całkujący należy powoli modyfikować, aż wyzeruje się błąd ustalony. Jako ostatnie dobiera się parametry członu D tak, by zmniejszyć przeregulowanie i zapewnić stabilność układu regulacji [24].



Rys. 5.6 Komunikacja z mikrokontrolerem za pomocą RS232.

Przy okazji ustalania nastaw sprawdzono również poprawność komunikacji interfejsu RS232, tak jak na Rys. 5.6. Następnie zaczęto programować kontroler, za każdym razem wpisując mu inną nastawę regulatora PID, zgodnie z ustalonymi założeniami i testowano pracę silnika, unikano dużych przeregulowań i celowano w jak największą stabilność nastawy. Silnik okazał się bardzo responsywny na zadane mu napięcie, co sprawiło, że regulacja była szybka i prosta. Po zakończonym procesie regulacji, pracę nad sterownikiem zostały skończone.

6. Podsumowanie

Praca inżynierska dotyczyła zaprojektowania i implementacji sterownika procesowego silnika BLDC z funkcjonalnością IoT, który umożliwia zdalne sterowanie i monitorowanie parametrów pracy silnika. W ramach projektu zaprojektowano płytkę PCB, dobrano odpowiednie komponenty oraz opracowano oprogramowanie działające na mikrokontrolerze **ESP32-WROOM**, który dzięki wbudowanej łączności Wi-Fi pozwala na integrację z siecią bezprzewodową.

Projekt był nie tylko okazją do praktycznego zastosowania wiedzy z zakresu elektroniki, automatyki i programowania, ale również do rozwijania umiejętności projektowania systemów wbudowanych z funkcjonalnością zdalnego sterowania.

Projekt można dalej udoskonalać, pozwalając np. na lepsze monitorowanie parametrów przez Wi-Fi. Użytkownik mógłby mieć możliwość wyświetlenia danych z pracy silnika w wybranym przez siebie odstępie czasowym, obecnie wykresy są rysowane dla użytkownika tylko kiedy wejdzie on na stronę. W kontekście komunikacji bezprzewodowej, warto by tez przemyśleć sposób komunikacji z innymi urządzeniami elektronicznymi.

Dodatkowo został źle zaimplementowany układ wzmacniacza sterującego napięciem. Przy projektowaniu nie przemyślano sposobu na ominięcie „martwej strefy” zakresu, w której nie pracuje silnik. Można by do wzmacniacza dodać trzeci sygnał, sterowany pinem High/Low, który pozwoliłby na lepsze wykorzystanie rozdzielczości przetworników DAC mikrokontrolera.

Pomocne w projekcie byłoby również wyprowadzenie punktów testowych na płytce i podłączeń nie tylko nieużywanych pinów mikrokontrolera, ale i tych użytych. Dodatkowo, warto było by również zmierzyć parametry pracy silnika i spisać je, przed rozpoczęciem pracy.

Finalnie sterownik spełnił wszystkie swoje założenia, oferuje precyzyjną i prostą obsługę silnika BLDC, pomimo możliwości kilku usprawnień i konieczności wprowadzenia modyfikacji po pełnym zmontowaniu.

BIBLIOGRAFIA

[1] Budziłowicz A., „Zastosowanie Silników BLDC (ang. BrushLess Direct-Current motor) we współczesnych napędach elektrycznych i w motoryzacji”, Instytut Naukowo-Wydawniczy "SPATIUM", Słupsk, 2015

[2] Mclennan, specyfikacja techniczna: *bldc58-50l-50-watt-datasheet*, dostępna na stronie: https://www.mclennan.co.uk/product/bldc58-50l-50-watt

[3] Lesker, opis złącza DB15 ze strony https://www.lesker.com/newweb/process_instruments/pdf/kjlc-datasheet-alicat-15-pin.pdf

[4] Sklep internetowy Aliexpress https://www.aliexpress.com/w/wholesale-pcb-mount-din.html?spm=a2g0o.productlist.search.0

[5] Strona internetowa Arduino https://store.arduino.cc/

[6] Strona internetowa STM32 https://www.st.com/en/evaluation-tools

[7] Strona internetowa Espressif https://www.espressif.com/en/products/devkits

[8] Zieliński Cezary, „Podstawy projektowania układów cyfrowych”, Wydawnictwo Naukowe PWN, Warszawa, 2003.

[9] Wrotek Witold, "Elektronika bez oporu. Praktyczne układy elektroniczne," Wydawnictwo Helion, 2022.

[10] Nota katalogowa LM2596S-5 firmy Texas Instruments, http://www.ti.com/lit/ds/symlink/lm2596.pdf

[11] Nota katalogowa AMS1117-3.3 firmy Advanced Monolithic Systems, http://www.advanced-monolithic.com/pdf/ds1117.pdf

[12] Artykuł EATON, „RS-232 Explained”, https://tripplite.eaton.com/products/rs-232-serial-to-usb-adapters

[13] Nota katalogowa MAX3232 firmy Maxim Integrated, https://datasheets.maximintegrated.com/en/ds/MAX3222-MAX3241.pdf

[14] Artykuł ze strony Toshiba, „What does rail-to-rail mean (Rail-to-Rail Op amp) ?”, https://toshiba.semicon-storage.com/us/semiconductor/knowledge/faq/linear_opamp/what-does-rail-to-rail-mean.html

[15] Nota katalogowa LMC6482 firmy Texas Instruments https://www.ti.com/lit/ds/symlink/lmc6482.pdf?ts=1725304168877&ref_url=https%253A%252F%252Fsearch.brave.com%252F

[16] Nota katalogowa ICL7660 firmy Analog Devices, https://www.analog.com/media/en/technical-documentation/data-sheets/icl7660-max1044.pdf

[17] Nota katalogowa ekranu OLED firmy Az-Delivery https://hw101.tbs1.de/sh1106/doc/1_3_inch_OLED_Datenblatt_4bcd023d-b1d6-4297-a022-71c523c952fd.pdf

[18] Nota katalogowa ESP32-WROOM-32E firmy Espressif, https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf

[19] Strona hifi.pl, tabela „Najpopularniejsze szeregi i należące do nich wartości”, https://www.hifi.pl/slownik/szeregi.php

[20] Instytut Technologii Informatycznych Wydział Studiów Międzynarodowych i Informatyki, „Paradygmat programowania wizualnego w inżynierii oprogramowania”.

[21] WCH-IC, sterownik CH340C, https://www.wch-ic.com/downloads/CH341SER_ZIP.html

[22] Bogusław Cyganek, „Programowanie w języku C++”, Wydawnictwo Naukowe PWN, Warszawa 2023.

[23] Dokumentacja techniczna Espressif dla Arduino, https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/

[24] Artykuł Robots for Robotics, „PID Control (with code), Verification, and Scheduling”, David Kohanbash, 2014, https://www.robotsforroboticists.com/pid-control/

[25] Poradnik do zaprogramowania pętli na drugim rdzeniu: https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/

[26] Biblioteka u8g2 https://github.com/olikraus/u8g2