

POLITECHNIKA POZNAŃSKA
WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI
INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ
ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ



PRACA DYPLOMOWA
INŻYNIERSKA

MIKROPROCESOROWY KASKADOWY UKŁAD REGULACJI PRĄDU,
PRĘDKOŚCI I POŁOŻENIA DLA NAPEDU Z SILNIKIEM BLDC

PROMOTOR:
DR INŻ. DOMINIK ŁUCZAK
POTWIERDZAM PRZYJĘCIE PRACY:

.....
DATA I PODPIS PROMOTORA

MATEUSZ SEMKŁO

2020-11-30

Spis treści

1 Wstęp	3
2 Silnik BLDC	4
2.1 Budowa	4
2.2 Zasada działania	5
2.3 Sposoby sterowania	5
2.4 Model matematyczny	5
2.5 Charakterystyka silnika - wady, zalety	7
3 Sterowanie Zorientowane Polowo	8
4 Modulacja wektorowa - SVPWM	10
5 Realizacja układu napędowego z silnikiem BLDC	12
5.1 Realizacja sprzętowa	12
5.2 Spis urządzeń	12
5.3 Schemat elektryczny	13
5.4 Opis i dane techniczne urządzeń	13
5.5 Konfiguracja ustawień mikrokontrolerów	16
5.5.1 STM32G431CB	16
5.5.2 STM32F746ZG	18
5.5.3 Przedstawienie głównych funkcji w kodzie źródłowym	20
5.6 Implementacja układu regulacji	26
5.7 Komunikacja	29
5.8 Interfejs użytkownika	30
6 Testowanie układu napędowego	31
6.1 Sprawdzenie poprawności działania funkcji SVPWM	31
6.2 Testowanie układu regulacji	32
6.2.1 Regulacja prądu	32
6.2.2 Regulacja prędkości obrotowej	40
6.2.3 Regulacja położenia wirnika	42
7 Podsumowanie	44
Wykaz ilustracji	45

1 Wstęp

Tematem pracy inżynierskiej jest *Mikroprocesorowy kaskadowy układ regulacji prądu, prędkości i położenia dla napędu z silnikiem BLDC*. Zainteresowanie i zdobyta wiedza z zakresu automatyki, mikroprocesorów i maszyn elektrycznych spowodowała chęć stworzenia układu napędowego dla silnika BLDC przy wykorzystaniu możliwości mikrokontrolerów i stawieniu czoła trudnościami przy jego praktycznej realizacji.

Celem jest pokazanie możliwości wykorzystania mikroprocesora z rodziny ARM Cortex wbudowanego w mikrokontroler firmy STM32 jako jednego ze sposobów na sprzętową realizację układu sterowania dla silnika BLDC. Zaimplementowanie Sterowania Zorientowanego Polowo (ang. Field Orient Control) z kaskadową konfiguracją układów regulacji prądu, prędkości, położenia oraz wykorzystanie wektorowej modulacji szerokości impulsów SVPWM (ang. Space Vector Pulse Width Modulation).

Zakres pracy obejmuje:

- dobór i uruchomienie mikroprocesorowej płyty rozwojowej STM32 dla wybranego układu napędowego z silnikiem BLDC,
- dobór parametrów regulatorów PID. Wykorzystanie regulatorów dostępnych w bibliotece CMSIS-DSP,
- opracowanie interfejsu użytkownika pozwalającego na parametryzację i monitoring pracy napędu w czasie rzeczywistym,
- analiza możliwości wykonania procedury automatycznego strojenia układu napędowego.

Silnik BLDC (ang. Brushless Direct Current) należy do rodziny silników z magnesami trwałymi. Charakteryzuje się stałym momentem elektromagnetycznym w szerokim zakresie prędkości obrotowej. Brak komutatora zapewnia większą sprawność i żywotność silnika. Z powodu wielu zalet często wykorzystywane w automatyce i robotyce.

Silnik ze względu na konstrukcję wymaga bardziej złożonego sposobu sterowania w porównaniu z silnikiem prądu stałego. W tym celu wykorzystana została metoda sterowania wektorowego FOC (ang. Field-orient control). Jest to jedna z powszechnie stosowanych technik w układach regulacji trójfazowych silników synchronicznych z magnesami trwałymi. Pozwala ona sprowadzić wartości prądów fazowych stojana z trójwymiarowego układ współrzędnych do wirującego układu o dwóch współrzędnych biegącego synchronicznie z wektorem strumienia wirnika. Uzyskujemy w ten sposób dostęp do dwóch wartości stałych - prądu i strumienia magnetycznego, które można umieścić bezpośrednio w pętlach regulatorów PID i uzyskać możliwość regulacji prądem, wzbudzeniem, prędkością i położeniem.

Do modulacji szerokości impulsów wykorzystano metodę SVPWM (ang. Space Vector Pulse Width Modulation). W porównaniu z alternatywną modulacją SPWM (ang. Sine Pulse Width Modulation) cechują się mniejszymi zniekształceniami sygnału spowodowanych przez wyższe harmoniczne i wykorzystuje większy przedział napięcia zasilania. W wyniku modulacji SVPWM generowane są wektory, które przyjmują kształt sześciokąta i dzieli przestrzeń na sześć sektorów. Wektor referencyjny jest generowany w wyniku aktywowania dwóch sąsiadujących wektorów z przestrzeni stanu w określonej sekwencji i czasie. W wyniku tych operacji formowane są trzy sygnały PWM dla trzech par tranzystorów przekształtnika energoelektronicznego.

Głównymi elementami układu jest zestaw prototypowy STM32 B-G431B-ESC1 oraz STM32 Nucleo F746ZG. B-G431B-ESC1 wyposażony w mikrokontroler STM32G431CB, rdzeń ARM Cortex M4, trójfazowy przekształtnik energoelektroniczny jest układem realizującym proces regulacji prądu, prędkości i położenia wirnika. Nucleo F746ZG odpowiada za komunikację między regulatorem a interfejsem użytkownika. Aplikacja użytkownika jest napisana w języku python i łączy się z płytą Nucleo F746ZG za pośrednictwem przewodu sieciowego przy wykorzystaniu protokołu TCP/IP. Informacje przekazywane są w postaci lańcuchów znaków w formacie JSON. Specyfika układu pomiarowego prądu wymaga aby pomiar był wykonywany, gdy wszystkie tranzystory grupy górnej były wyłączone a grupy dolnej włączone. Wyzwolenie następuje w osi symetrii każdego okresu sygnału PWM. W funkcji obsługi zdarzenia przeprowadzane są wszystkie operacje związane z regulacją i modulacją sygnałów.

W układzie zdefiniowane są cztery regulatory PID: prądu, strumienia magnetycznego wirnika skojarzonego ze stojanem, prędkości i położenia wirnika. Są one połączone ze sobą w sposób kaskadowy. Pętle regulacji są zagnieździone jedna w drugiej, gdzie wyjście z regulatora pętli zewnętrznej ustala wartość zadawaną pętli wewnętrznej.

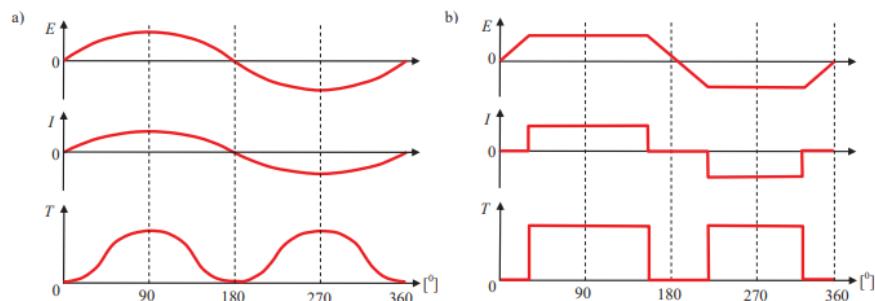
Przy pomocy oscyloskopu i oprogramowania STMStudio do podglądu zmiennych programu w czasie rzeczywistym przeprowadzono pomiary prądu, prędkości i położenia wirnika dla różnych wartości

zadanych przy określonych nastawach regulatorów PID. Zmierzone przebiegi najistotniejszych wielkości zostały przedstawione w rozdziale 6.

2 Silnik BLDC

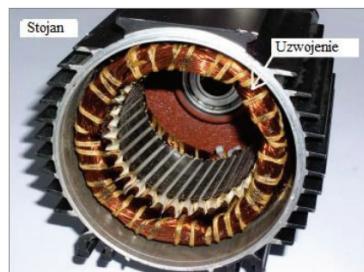
2.1 Budowa

Bezsaczotkowe silniki z magnesami trwałymi można podzielić na silniki prądu stałego BLDC oraz synchroniczne silniki prądu przemiennego PMSM (ang. Permanent Magnet Synchronous Motor). Różnią się one kształtem przebiegu siły elektromotorycznej. Na rysunku 1 przedstawiono przebiegi indukowanego napięcia, prądu i momentu elektromagnetycznego dla silnika BLDC i PMSM. [6]



Rysunek 1: Przebiegi siły elektromotorycznej E, prądu I i momentu T dla jednej fazy dla: a) silnika PMSM, b) silnika BLDC [15]

Istnieje wiele odmian silników BLDC różniących się budową, konstrukcją obwodów magnetycznych, rozkładem indukcji i kształtem sił elektromotorycznych. Uzwojenie twornika jest zazwyczaj w stojaniu i jest uzwojeniem trójfazowym. Kształt magnesów trwałe umieszczonej na wirniku zapewniają stałą wartość indukcji w szczelinie maszyny i trapezoidalny kształt siły elektromotorycznej. Rysunek 2 i 3 przedstawia budowę stojana i wirnika przykładowego silnika bezszczotkowego. [7]



Rysunek 2: Stojan silnika BLDC [6]

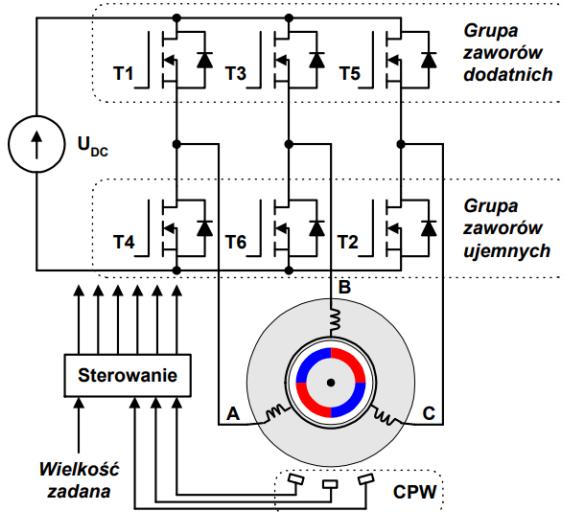


Rysunek 3: Wirnik silnika BLDC [6]

2.2 Zasada działania

Silnik elektryczny przetwarza dostarczoną energię elektryczną w energię mechaniczną. Ruch wirnika występuje, gdy zasilając odpowiednie uzwojenia stojana następuje pojawienie się wirującego pola elektromagnetycznego, które oddziałuje na pole magnetyczne wirnika i wprawia go w ruch. Strumienie magnetyczne wirnika i stojana powinny być względem siebie nieruchome. W klasycznym silniku prądu stałego powyższy warunek jest spełniony przez odpowiednie umiejscowienie szczotek komutatora względem pola stojana. Z powodu braku klasycznego komutatora w silniku BLDC zapewnienie wirowania strumienia stojana synchronicznie z wirnikiem jest realizowane przez odpowiednie przełączanie uzwojeń. W tym celu wykorzystuje się przetwornik energoelektroniczny, którego zadaniem jest odpowiednie usytuowanie strumieni względem siebie, poprzez przełączanie tranzystorami mocy. Aby to zrealizować potrzebna jest wiedza o aktualnym położeniu kąta obrotu wirnika, wykorzystując czujniki Halla lub przetworniki obrotowo-impulsowe.^[7]

Na rysunku 4 pokazano uproszczony schemat sterowania i zasilania silnika BLDC.



Rysunek 4: Uproszczony schemat sterowania i zasilania silnika [3]

2.3 Sposoby sterowania

Silnik bezszczotkowy nie posiada konwencjonalnego komutatora , więc stale należy kontrolować położenie wirnika oraz zastosować odpowiedni sposób załączania uzwojeń. Istnieją trzy podstawowe strategie komutacji [7]:

- Komutacja trapezoidalna – przełączanie uzwojeń odbywa się w sposób dyskretny więc nie jest potrzebny ciągły pomiar kąta obrotu wału. Najczęściej wykorzystuje się czujniki Halla, które są rozmieszczone w odstępach 120 stopni lub bezczujnikowe obliczanie położenia wirnika. Zmiana stanu jednego z czujników inicjuje rozpoczęcie zmiany zasilania uzwojeń stojana. Zasilane są jednocześnie tylko dwa uzwojenia.
 - Komutacja sinusoidalna – przełączanie uzwojeń odbywa się w sposób quasi-ciągły. Prądy fazowe i siły elektromotoryczne powinny być sinusoidalne. Przekształtnik energoelektroniczny pełni funkcję falownika. Do pomiaru położenia wykorzystuje się przetwornik impulsowo-obrotowy lub algorytm do bezczujnikowego obliczania kąta obrotu wirnika. Zasilane są jednocześnie trzy uzwojenia.
 - Zastosowanie algorytmu FOC (ang. Field Orient Control).
- Metoda ta została wykorzystana w niniejszej pracy i opisana w rozdziale 3.

2.4 Model matematyczny

Wartość momentu elektromagnetycznego wytworzonego dla płaskiego fragmentu siły elektromotorycznej przez prąd płynący w jednym uzwojeniu fazowym [7]:

$$Me = p\psi_p i \quad (1)$$

Me - moment elektromagnetyczny

p - liczba par biegunów

ψ_p -strumień magnetyczny wytworzony przez wirnik a skojarzony z uzwojeniem stojana

i - prąd uzwojenia stojana

Chwilowa wartość strumienia skojarzonego z k-tym uzwojeniem zmieniająca się w funkcji obrotu wirnika [7]:

$$\psi_k(\theta_e) = \psi_p f(\theta_e) \quad (2)$$

θ_e - elektryczny kąt położenia wirnika

Łącząc powyższe wyrażenia 1 i 2 uzyskujemy moment elektromagnetyczny dla k-tej fazy [7]:

$$M_{ek} = p\psi_k(\theta_e)i \quad (3)$$

Moment całkowity jest równy sumie momentów elektromagnetycznych[7]:

$$M_e = \sum_{k=1}^3 M_{ek} \quad (4)$$

Równanie ruch ma postać:

$$M_e = J \frac{d\omega}{dt} + M \quad (5)$$

J - moment bezwładności

M - moment oporowy

Wzór dla płaskiego odcinka siły elektromotorycznej[7]:

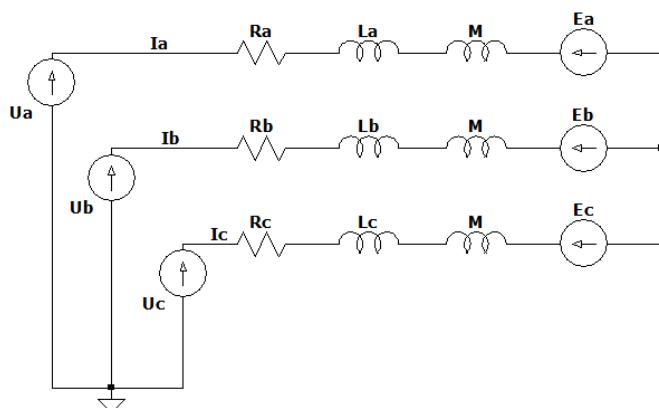
$$E_p = p\psi_p \omega \quad (6)$$

ω - prędkość obrotowa wirnika

Wyrażenie opisujące siłę elektromotoryczną indukowaną w k-tym uzwojeniu[7]:

$$E_k = p\psi_k(\theta_e)\omega \quad (7)$$

Silnik BLDC można traktować jako szczególny przypadek silnika synchronicznego. Schemat zastępczy silnika trójfazowego z magnesami trwałymi przedstawiony na rysunku 5.



Rysunek 5: Schemat silnika synchronicznego z magnesami trwałymi

M - indukcyjność wzajemna
 L - indukcyjność własna
 R - rezystancja uzwojeń
 E - siła elektromotoryczna

Zakładając, że indukcyjności własne i rezystancje są równe:

$$L = L_A = L_B = L_C, R = R_A = R_B = R_C \quad (8)$$

oraz indukcyjności wzajemne również to całkowita indukcyjność jednej fazy jest równa:

$$L_S = L - M \quad (9)$$

Otrzymujemy wyrażenie [7]:

$$\begin{bmatrix} U_A \\ U_B \\ U_C \end{bmatrix} = R \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} + L_S \frac{d}{dt} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} + \begin{bmatrix} E_A \\ E_B \\ E_C \end{bmatrix} \quad (10)$$

oraz

$$\frac{d}{dt} \begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} = -\frac{R}{L} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_A \\ I_B \\ I_C \end{bmatrix} - \frac{1}{L} \begin{bmatrix} E_A \\ E_B \\ E_C \end{bmatrix} + \frac{1}{L} \begin{bmatrix} U_A \\ U_B \\ U_C \end{bmatrix} \quad (11)$$

Wykorzystując równanie 3, 5, 7, 11 i zapis 12 w postaci równać stanu uzyska się pełen model idealnego silnika o trapezoidalnej sile elektromotorycznej [7]:

$$\hat{x} = Ax + Bu \quad (12)$$

gdzie:

$$x = [I_A \quad I_B \quad I_C \quad \omega \quad \theta]^T \quad (13)$$

$$u = [U_A \quad U_B \quad U_C \quad M]^T \quad (14)$$

$$A = \begin{bmatrix} -\frac{R}{L} & 0 & 0 & -p\psi_A(\theta_e)/L & 0 \\ 0 & -\frac{R}{L} & 0 & -p\psi_B(\theta_e)/L & 0 \\ 0 & 0 & -\frac{R}{L} & -p\psi_C(\theta_e)/L & 0 \\ -p\psi_A(\theta_e)/J & -p\psi_B(\theta_e)/J & -p\psi_C(\theta_e)/J & -B/J & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (15)$$

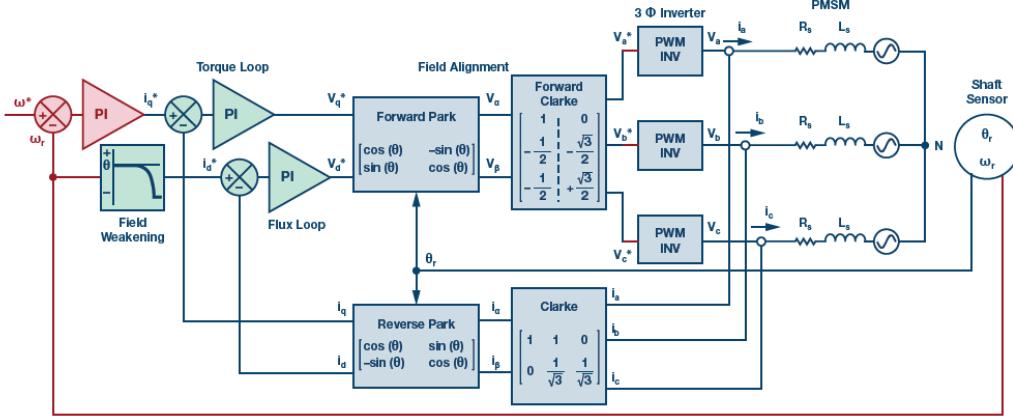
$$B = \begin{bmatrix} 1/L & 0 & 0 & 0 \\ 0 & 1/L & 0 & 0 \\ 0 & 0 & 1/L & 0 \\ 0 & 0 & 0 & 1/L \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (16)$$

2.5 Charakterystyka silnika - wady, zalety

Brak konwencjonalnego komutatora wydłuża żywotność silnika i nie wymaga konserwacji. Brak szczotek powoduje mniejsze tarcie, brak negatywnych skutków iskrzenia podczas komutacji oraz mniejsze zaburzenia elektromotoryczne. W porównaniu ze szczotkowym silnikiem prądu stałego uzwojenia twornika są zazwyczaj umieszczone w zewnętrznej części silnika, co zapewnia lepsze odprowadzanie ciepła do otoczenia. Magnesy trwałe mają bardzo małą przenikalność magnetyczną zbliżoną do przenikalności magnetycznej powietrza. Reaktancja twornika jest pomijalnie mała dzięki czemu osiągany jest duży moment elektromagnetyczny. Do wad należy zaliczyć bardziej skomplikowany sposób sterowania silnikiem przez zastosowanie komutatora elektronicznego w postaci przekształtnika energoelektronicznego do cyklicznego zasilania uzwojeń twornika.

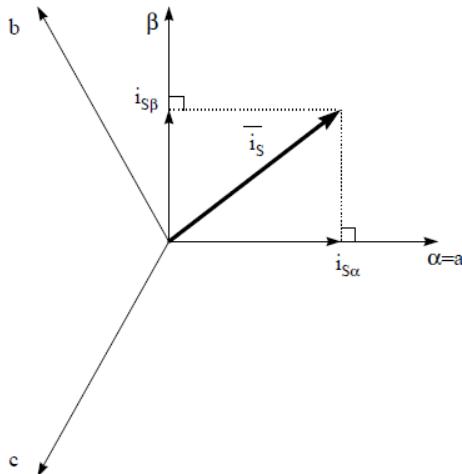
3 Sterowanie Zorientowane Polowo

Algorytm Sterowania Zorientowanego Polowo FOC przedstawiony schematycznie na rysunku 6 jest jedną z powszechnie stosowanych metod sterowania silników trójfazowych z magnesami trwałymi. Polega na bezpośrednim sterowaniu wirującym polem stojana. Pozwala uzyskać dokładną i dynamiczną regulację momentu elektrodynamycznego, prędkości i położenia wirnika.[10]



Rysunek 6: Algorytm FOC [5]

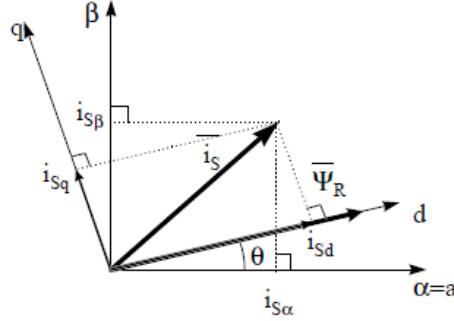
Metoda FOC steruje prądami fazowymi stojana reprezentowanymi w postaci wektora. Czujniki prądu dostarczają informację zwrotną o aktualnych prądach fazowych. Transformata Clark'a, wyrażona za pomocą wzoru 17 i pokazana na rysunku 7, przekształca trójwymiarowy układ współrzędnych, w którym opisane są prądy fazowe I_a, I_b, I_c na układ ograniczony do dwóch współrzędnych otrzymując wirujący wektor na płaszczyźnie. Znając aktualne położenie wirnika i korzystając z transformaty Park'a, wyrażona we wzorze 18 i pokazana na rysunku 8, uzyskujemy wirujący układ współrzędnych. Operacja ta pozwala na przekształcenie wirującego wektora prądu na dwie składowe stałe: strumienia magnetycznego I_d oraz momentu elektromagnetycznego I_q . Daje to możliwość sterowania w sposób rozdzielny prądem odpowiadającym za moment obrotowy oraz wzbudzeniem wytwarzającym strumień magnetyczny skojarzony z twornikiem. Minimalizując wartość I_d zapewniamy kąt prosty między polem magnetycznym stojana i wirnika oraz uzyskujemy największą sprawność silnika.[10] [4]



Rysunek 7: Transformata Clark'a [4]

Wyrażenie opisujące transformatę Clark'a [10]:

$$\begin{cases} I_\alpha = I_A \\ I_\beta = \frac{1}{\sqrt{3}}I_A + \frac{2}{\sqrt{3}}I_B \end{cases} \quad (17)$$



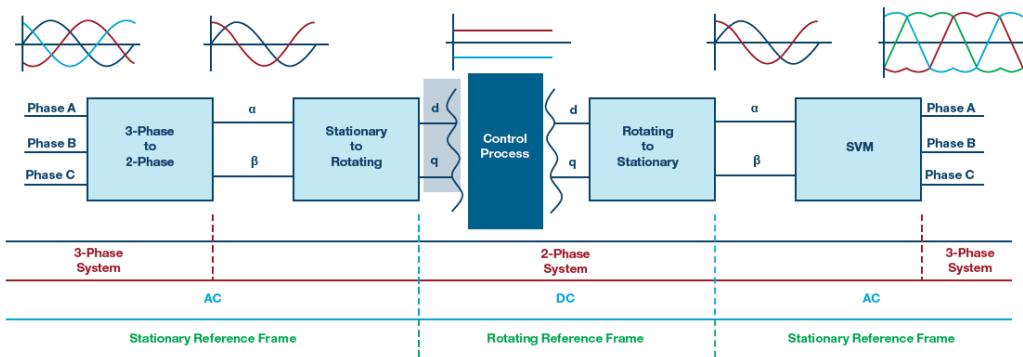
Rysunek 8: Transformata Park'a [4]

Wyrażenie opisujące transformatę Park'a [10]:

$$\begin{cases} I_d = I_\alpha \cos(\theta) + I_\beta \sin(\theta) \\ I_q = I_\alpha \sin(\theta) + I_\beta \cos(\theta) \end{cases} \quad (18)$$

Przekształcenie przebiegów prądów przemiennych w dwie składowe stałoprądowe pozwala na zastosowanie regulatorów PID dla składowej I_d oraz I_q uzyskując regulator prądu i wzbudzenia. Regulacje prędkości obrotowej i położenia wirnika jest zrealizowana przez dodanie kolejnych regulatorów PID do gałęzi ze stałą I_q .

Zastosowanie odwrotnych transformat pozwala na sprowadzenie składowych stałych do trzech przebiegów sinusoidalnych i wprowadzenie ich wartości na kolejny blok algorytmu FOC zwany SVPWM (ang. Space Vector Pulse Width Modulation) opisany w rozdziale 4. Blok ten steruje bramkami tranzystorów przetwornika, aby uzyskać w uzwojeniach takie przebiegi prądów, które będą generowały odpowiednią amplitudę i prędkość wirowania pola magnetycznego stojana względem pola wirnika. Rysunek 9 przedstawia kolejność dokonywanych transformacji.

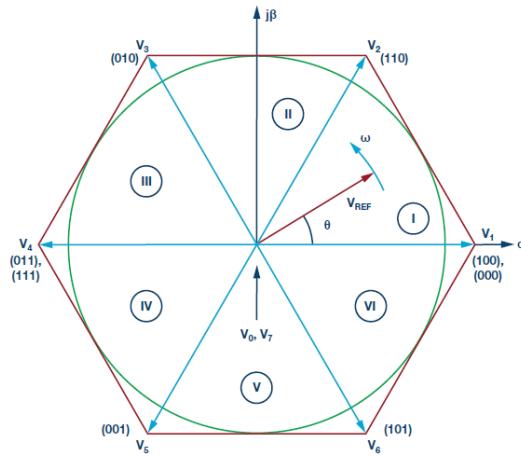


Rysunek 9: Transformacje w algorytmie FOC [5]

4 Modulacja wektorowa - SVPWM

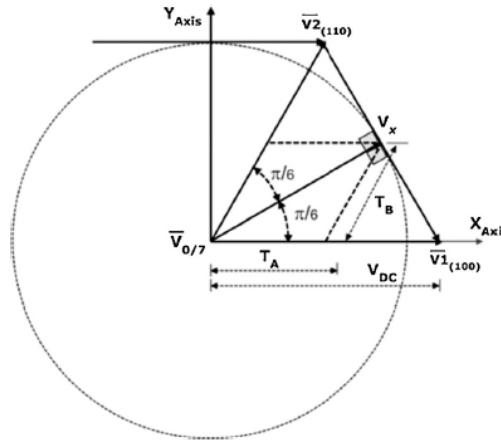
Wydajność prądowa mikrokontrolera jest niewystarczająca aby zasilić silnik, dlatego uzwojenia stojana są bezpośrednio podłączone do przekształtnika energoelektronicznego zasilanego napięciem stałym. Schemat przekształtnika przedstawia rysunek 4. SVPWM jest metodą pozwalającą określić, sekwencje i czas trwania załączeń tranzystorów. W porównaniu z konwencjonalną metodą SPWM (ang. Sine Pulse Width Modulation) do sterowania silników synchronicznych, pozwala na wykorzystanie szerszego zakresu napięcia zasilania i zmniejszenia zniekształceń spowodowanych przez wyższe harmoniczne.[9]

Elementami wykonawczymi są tranzystory mocy, które można podzielić na grupę trzech zaworów dodatnich i ujemnych. W każdej chwili aktywne są trzy zawory. W ten sposób uzyskujemy osiem możliwych kombinacji załączeń tranzystorów, które generują osiem wektorów (sześć wektorów aktywnych, dwa wektory zerowe lub nieaktywne) i dzielą przestrzeń na sześć sektorów - schematycznie przedstawiono na rysunku 10.



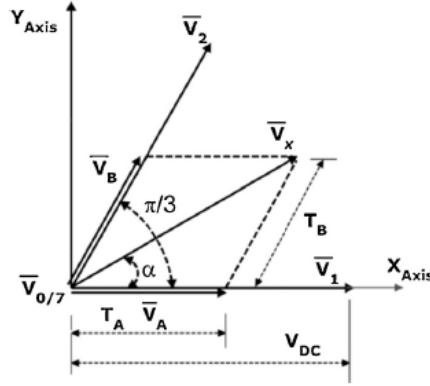
Rysunek 10: Wektory w przestrzeni stanu i podział na sektory [5]

Maksymalne napięcie fazowe jest równe $2/3$ VDC. Aby uzyskać nieodkształcony przebieg sinusoidalny maksymalne napięcie szczytowe jest ograniczone do wartości promienia okręgu wpisanego w sześciokąt i jest równe $V_{max} = V_{dc}\sqrt{3}/2$. Powyższe założenie przedstawiono na rysunku 11.[9]



Rysunek 11: Maksymalna długość wektora [5]

Na przykładzie rysunku 12 został objaśniony sposób generowania dowolnego wektora według metody SVPWM. Wektor V_x znajduje się w sektorze pierwszym i jest wynikiem dodania do siebie trzech wektorów V_A , V_B , V_0 , gdzie V_0 jest wektorem zerowym.[9]



Rysunek 12: Generowanie wektora V_X [9]

Dla sektora pierwszego wektor V_A jest zgodny z kierunkiem wektora V_1 a wektor V_B z kierunkiem wektora V_2 . Długość obu wektorów jest determinowana przez czas, w którym dana sekwencja zaworów (reprezentowana przez określony wektor $V1, \dots, V6$) jest aktywna.[9]

Z równania [9]:

$$\begin{cases} |V_x| \cos(\alpha) \times T_s = V_{DC} \times T_A + V_{DC} \times \cos(\pi/3) \times T_B \\ |V_x| \sin(\alpha) \times T_s = V_{DC} \times \sin(\pi/3) \times T_B \end{cases} \quad (19)$$

oraz z wyrażenia 20, które pozwala zredukować kąt α do przedziału od 0 do $\pi/3$ [2] :

$$\theta = \phi - \frac{k-1}{3}\pi \quad (20)$$

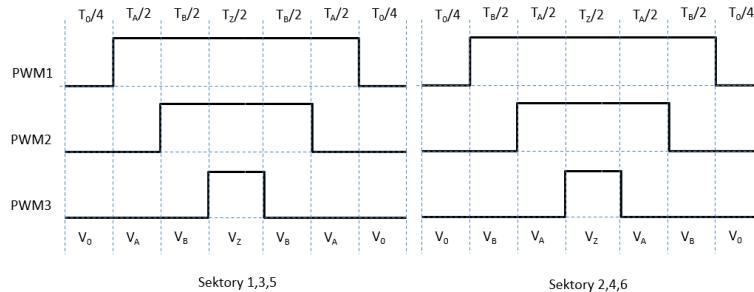
gdzie: k - sektor 1..6

można wyznaczyć zależność pozwalającą określić czasy T_A , T_B generowania wektorów V_A , V_B dla dowolnego sektora [2] :

$$\begin{cases} T_A = \frac{\sqrt{3}T_s V_x}{V_{DC}} (\sin(k\pi/3 - \theta)) \\ T_B = \frac{\sqrt{3}T_s V_x}{V_{DC}} (\sin((k-1)\pi/3 - \theta)) \end{cases} \quad (21)$$

Suma czasów T_A , T_B i T_0 jest równa okresowi modulacji szerokości impulsów T_S .

Znając sektor i czasy T_A , T_B , T_0 można wygenerować dowolny wektor w przestrzeni stanu. Aby tego dokonać należy zastosować odpowiednią sekwencję załączeń wektorów V_A , V_B , V_0 . Wektory te, w zależności od sektora, w którym się znajdują, mają swoją reprezentację w odpowiadającej im kombinacji załączeń kluczowych tranzystorów przekształtnika. Rysunek 13 przedstawia sekwencje wektorów V_A , V_B , V_0 i czasy ich aktywności w zależności od umiejscowienia wektora referencyjnego w przestrzeni stanu. W wyniku powyższych operacji otrzymujemy trzy sygnały PWM dla każdej gałęzi trójfazowego przekształtnika [8].

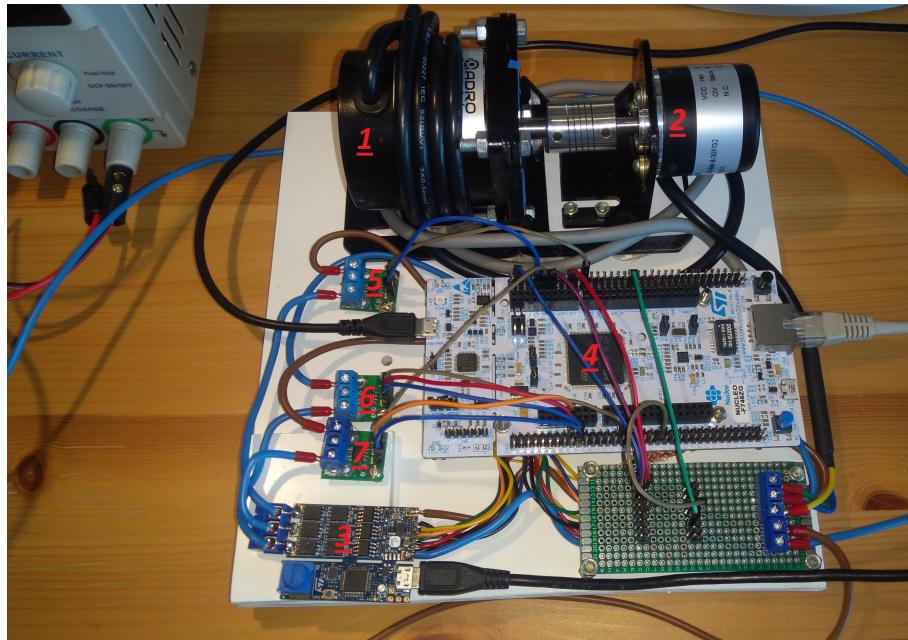


Rysunek 13: Sekwencja generowania wektora referencyjnego

5 Realizacja układu napędowego z silnikiem BLDC

5.1 Realizacja sprzętowa

Realizacja praktyczna układu napędowego jest przedstawiona na rysunku 14.

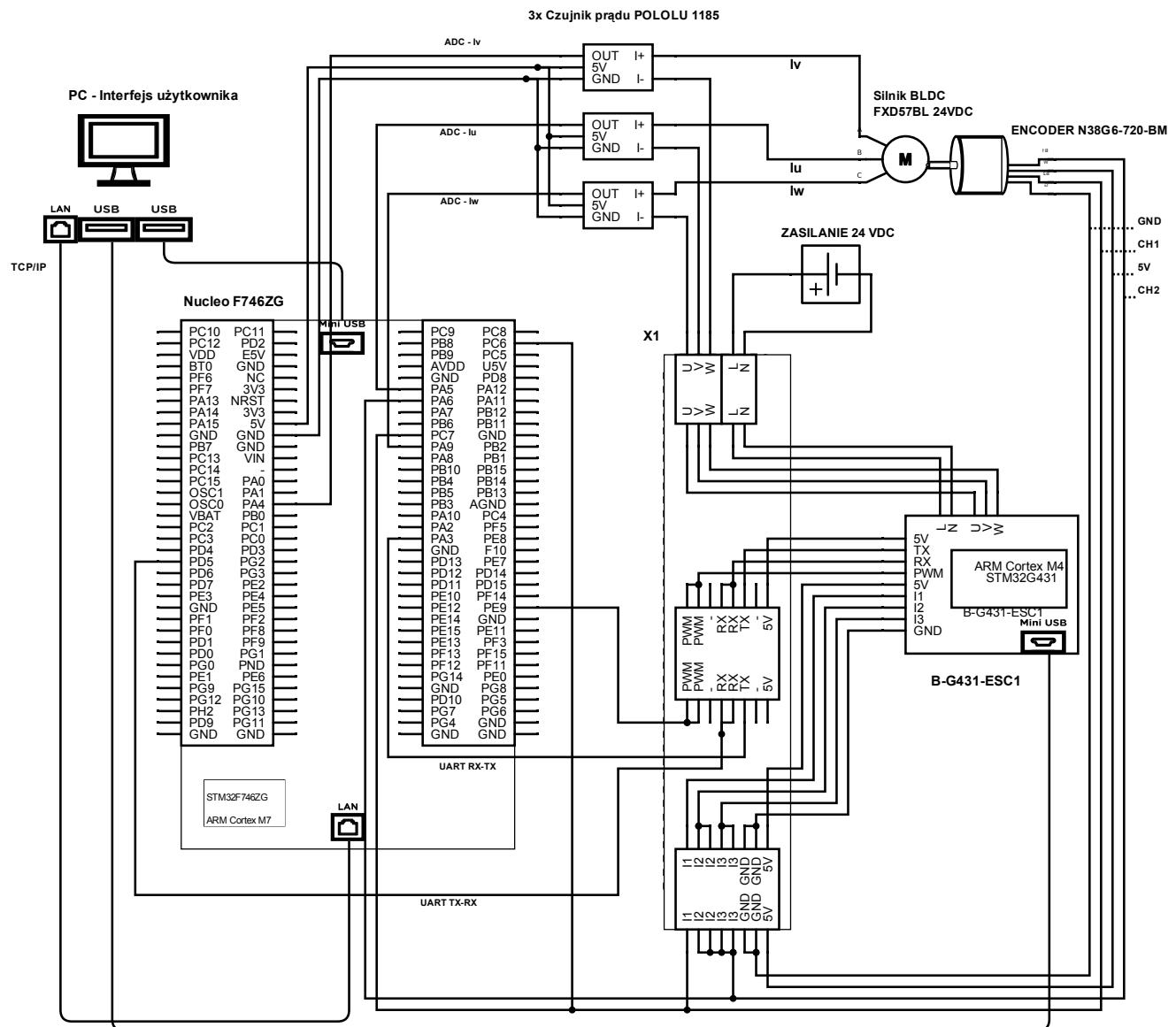


Rysunek 14: Napęd z silnikiem BLDC

5.2 Spis urządzeń

1. Silnik bezszczotkowy FXD57BL 24 VDC.
2. Przetwornik obrotowo-impulsowy N38G6-720-BM-8-30FG2.
3. Zestaw rozwojowy STM32 B-G431B-ESC1.
4. Zestaw rozwojowy STM32 Nucleo F746ZG.
5. Czujnik prądu Pololu 1185 ACS714 -5A : +5A.
6. Czujnik prądu Pololu 1185 ACS714 -5A : +5A.
7. Czujnik prądu Pololu 1185 ACS714 -5A : +5A.
8. Zasilacz laboratoryjny KORAD 0-30V, 0-5A.

5.3 Schemat elektryczny



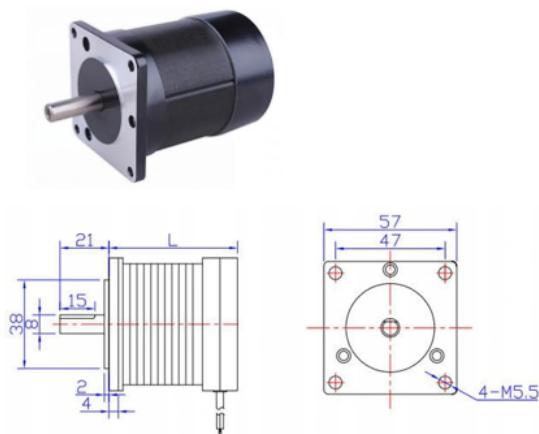
5.4 Opis i dane techniczne urządzeń

1. Silnik bezszczotkowy FXD57BL 24 VDC

Silnik wykorzystany do budowy napędu pokazany na rysunku 15

Parametry techniczne:

- moc: 60W,
- napięcie zasilania: 24 V,
- prąd znamionowy: 3.3 A,
- maksymalna prędkość obrotowa: 3000 obr/min,
- moment znamionowy: 0,18 Nm,
- wirnik dwupolowy,
- długość silnika: 55 mm.



Rysunek 15: Silnik BLDC

2. Inkrementalny przetwornik obrotowo-impulsowy N38G6-720-BM-8-30FG2

Przetwornik obrotowo-impulsowy inaczej enkoder jest czujnikiem mierzącym ruch. Przekształca ruch obrotowy w sygnał elektryczny o przebiegu prostokątnym. Enkoder inkrementalny nie generuje informacji o położeniu bezwzględnym.

Parametry techniczne:

- rozdzielcość: 720 impulsów/obrót,
- napięcie zasilania: 5 - 30 VDC,
- rodzaj wyjścia: PUSH - PULL,
- maksymalny pobór prądu: 150 mA,
- maksymalne obciążenie prądowe: 30 mA,
- maksymalna częstotliwość: 150 kHz,
- znamionowa prędkość pracy: 5000 obr/min.

3. Zestaw rozwojowy STM32 B-G431B-ESC1

Zestaw rozwojowy B-G431B-ESC1 firmy STMicroelectronics oparty na mikrokontrolerze STM32G431CB jest dedykowanym układem elektronicznym dla napędów z silnikami BLDC/PMSM. Posiada przekształtnik energoelektroniczny zbudowany w oparciu o sterowniki L6387 i tranzystory mocy MOSFET STL180N6F7. Dla sterowania i monitorowania pracą silnika oraz do współpracy z innymi układami zapewnia różne sposoby komunikacji - UART, CAN i PWM. STM32G431CB wyposażony w procesor ARM Cortex M4 32-bity, pełen zestaw instrukcji DSP (ang. Digital Signal Processing) oraz blok do przetwarzania liczb zmiennoprzecinkowych FPU (ang. Floating-Point Unit) zapewnia możliwość implementacji różnych zaawansowanych algorytmów sterowania. Posiada wbudowane układy mierzące prąd w silniku. Rysunek 16 przedstawia najistotniejsze elementy wbudowane w układ B-G431B-ESC1.[14]

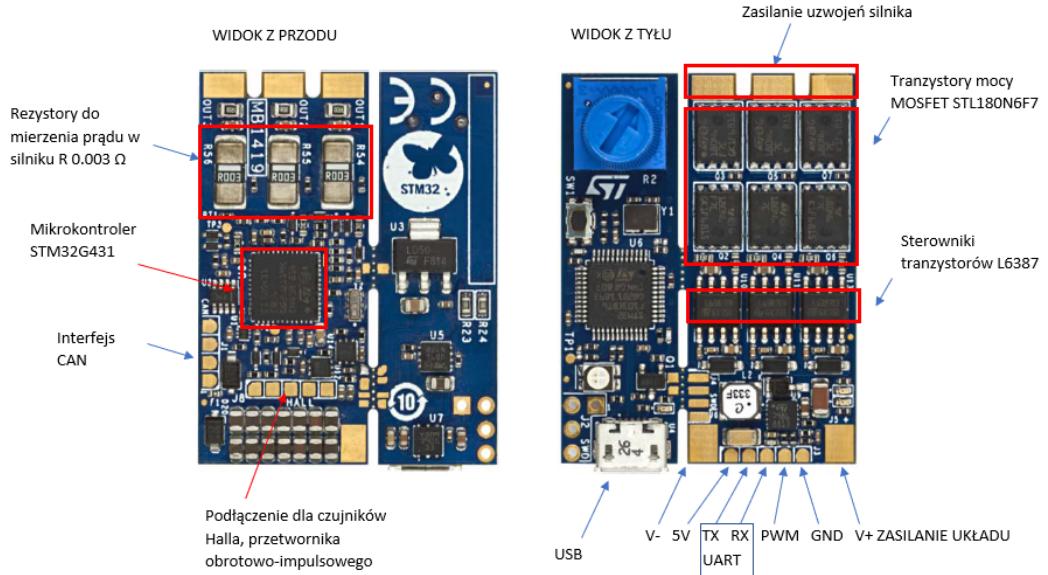
Parametry techniczne B-G431B-ESC1 [14]:

- napięcie znamionowe 60 V,
- maksymalny prąd szczytowy (z układem chłodzenia): 40 A.

Parametry techniczne STM32G431CB [14]:

- procesor: ARM Cortex M4 32-bity DSP,
- częstotliwość taktowania procesora: 170 MHz,
- pamięć Flash: 128 KB,
- pamięć SRAM: 32 KB,

- przetwornik analogowo-cyfrowy 12-bitowy: 2,
- komparatory: 4,
- wbudowane wzmacniacze operacyjne: 3,
- kontroler DMA,
- interfejsy: UART, PWM, CAN.



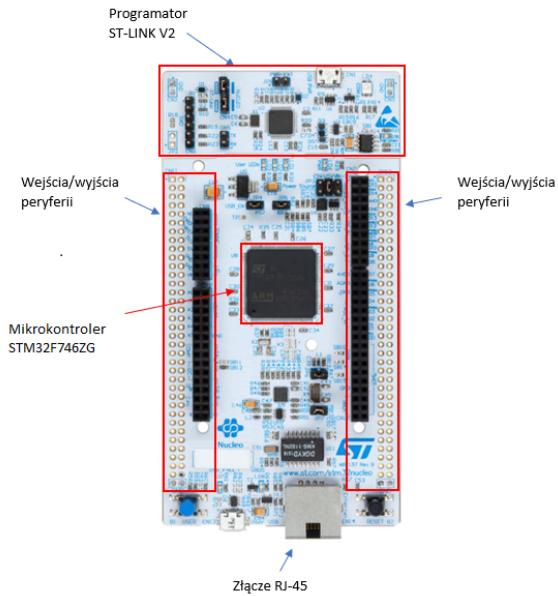
Rysunek 16: Zestaw rozwojowy STM32 B-G431-ESC1 [14]

4. Zestaw rozwojowy STM32 Nucleo F746ZG

Nucleo F746ZG jest zestawem rozwojowym opartym na mikrokontrolerze STM32F746Z. Duża ilość peryferiów oraz wyprowadzonych pinów do ich podłączenia zapewnia wiele możliwości prototypowania. Posiada wbudowany programator ST-LINK/V2-1 oraz złącze RJ-45 do połączenia sieciowego. Rysunek 17 przedstawia układ Nucleo F746ZG.

Parametry techniczne mikrokontrolera STM32F746Z [11]:

- procesor: ARM Cortex M7 32-bity FPU, DSP,
- częstotliwość taktowania procesora: 216 MHz,
- pamięć Flash: 1 MB,
- pamięć SRAM: 320 KB,
- przetwornik analogowo-cyfrowy 12-bitowy: 3,
- liczniki: 18,
- przetwornik cyfrowo-analogowe 12-bitowy: 2,
- interfejsy: UART, USART, I²C, SPI, CAN, SAIs, HDMI, USB,
- kontroler DMA.



Rysunek 17: Zestaw rozwojowy STM32 Nucleo F746ZG [12]

5.5 Konfiguracja ustawień mikrokontrolerów

Do konfiguracji ustawień mikrokontrolerów w układach rozwojowych STM32 B-G431-ESC1 i Nucleo F746ZG wykorzystano aplikacje STM32CubeMX. Umożliwia ona w sposób wizualny dokonać dowolnych zmian w mikroprocesorze i w peryferiach. Po zachowaniu ustawień zostaje wygenerowany plik w języku C, który implementuje wszystkie dokonane modyfikacje.

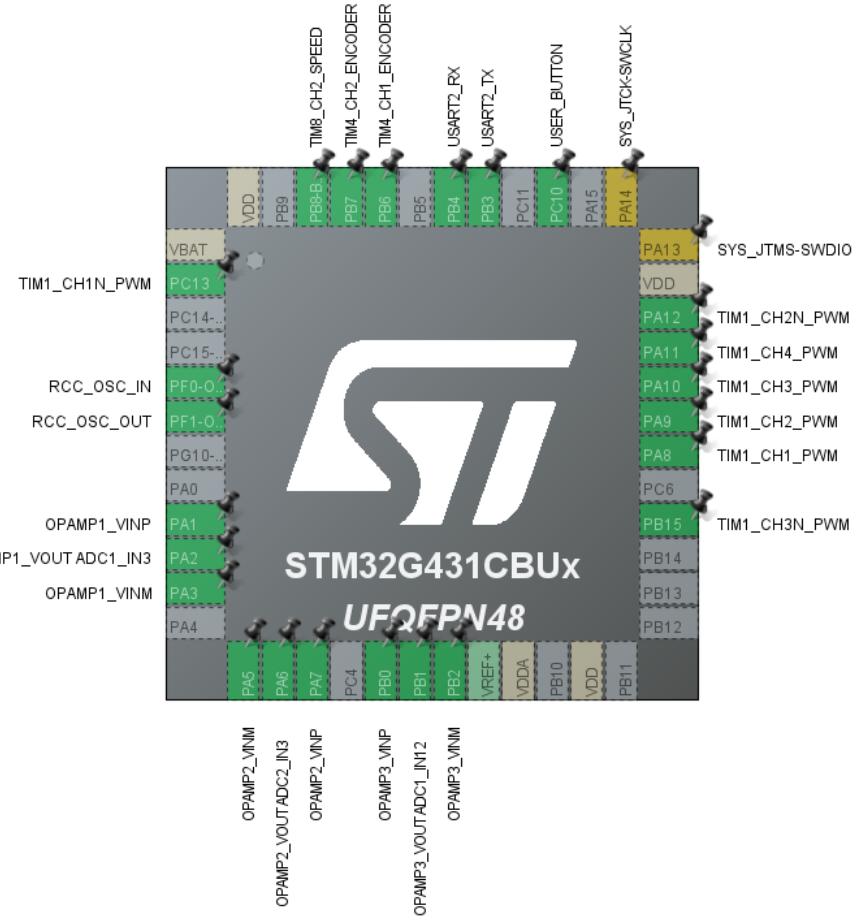
5.5.1 STM32G431CB

Zestaw prototypowy B-G431-ESC1 wyposażony jest w mikrokontroler STM32G431CB. Rysunek 18, 19 przedstawia konfigurację, opis wykorzystanych pinów i ustawienia częstotliwości taktowania poszczególnych peryferii za pomocą aplikacji STM32CubeMX.

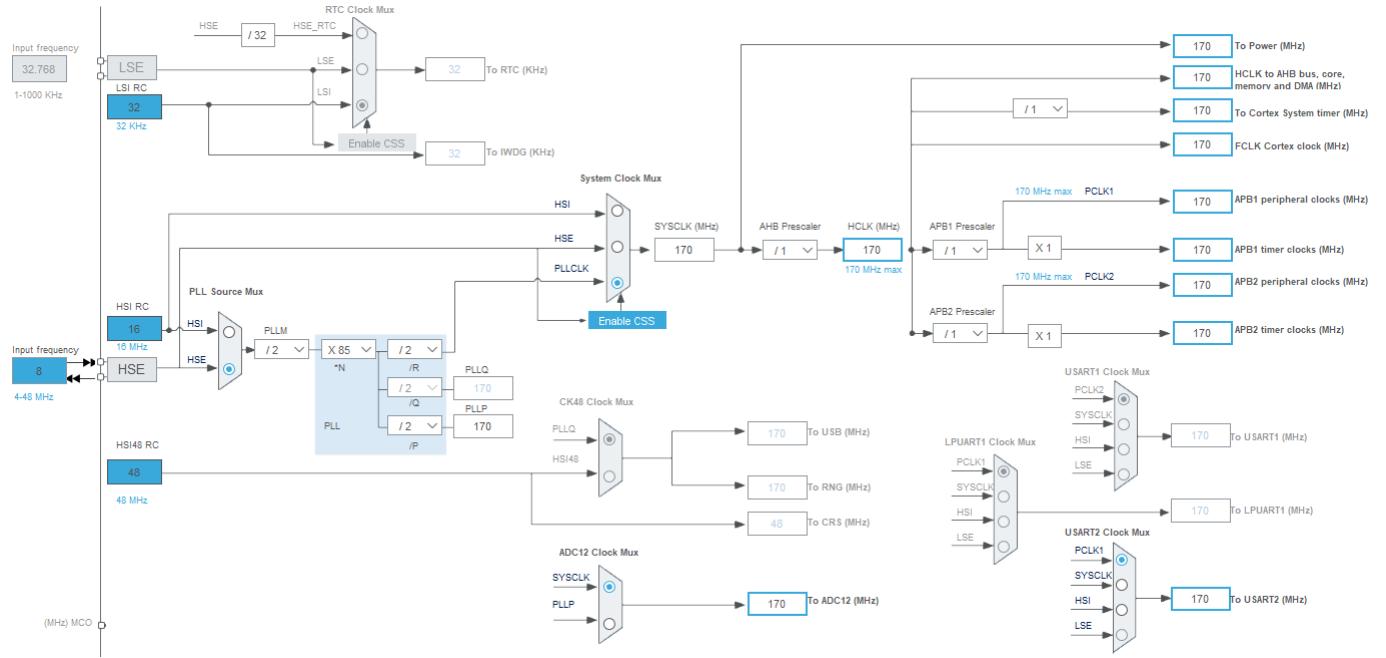
Dokonane ustawienia:

- PA8, PA9, PA10, PA12, PC13, PB15 są wyjściami kanałów licznika 1, które generują trzy sygnały PWM dla bramek tranzystorów MOSFET przetwornika energoelektronicznego,
- PA11 jest wyjściem kanału licznika 1 do wyzwalania konwersji przetworników ADC oraz synchronizacji pomiarów czujników z układem Nucleo F746ZG,
- PB4, PB3 - port szeregowy UART o przepustowości 115200 Bits/s,
- PB8 jest wejściem licznika 8 dla jednego z kanałów przetwornika obrotowo-impulsowego służącego do pomiaru prędkości. Licznik jest ustawiony w trybie Reset Mode co oznacza że każde zbocze narastające resetuje jego wartość, dzięki czemu można wyznaczyć okres między impulsami.
- PB7, PB6 są wejściami licznika 4 dla kanałów przetwornika obrotowo-impulsowego. Licznik jest ustawiony w trybie Encoder Mode, który pozwala na dodawanie i odejmowanie zliczonych impulsów z przetwornika w zależności od kierunku obrotu.
- Piny oznaczone jako OPAMPx_VOUT ADCx_INx są wyjściami wzmacniacza operacyjnego, który został wewnętrznie podłączony do przetwornika ADC w celu pomiaru prądów fazowych - patrz rozdział 5.6,
- piny oznaczone jako OPAMPx_VINP oraz OPAMPx_VINM są wejściami wzmacniacza operacyjnego,
- przetwornik ADC inicjuje wywołanie funkcji obsługi zdarzenia po zakończeniu wszystkich pomiarów,

- PC10 jest pinem przypisanym do przycisku, który realizuje funkcje uruchomienia i zatrzymania napędu.



Rysunek 18: Konfiguracja mikrokontrolera STM32G431



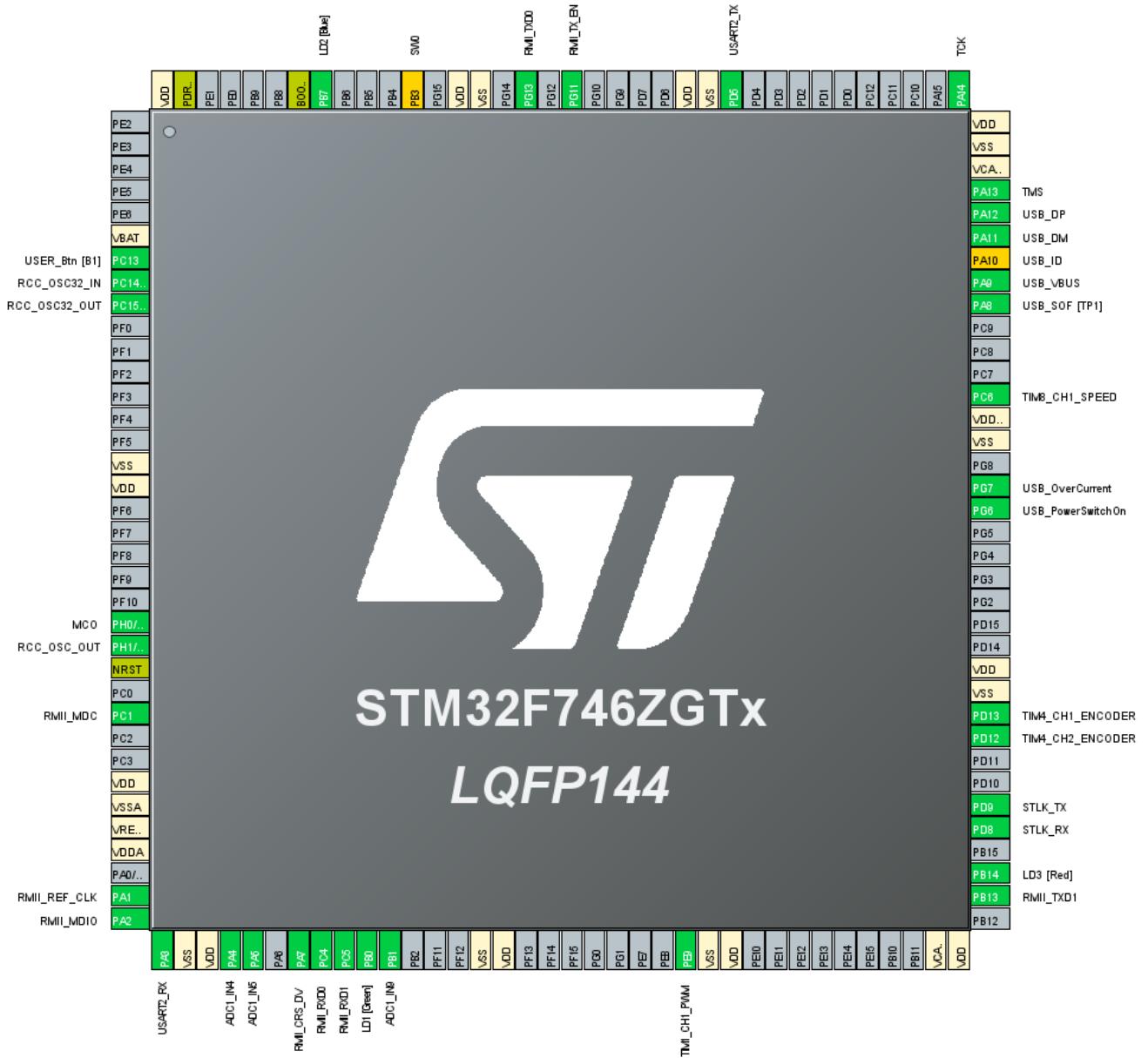
Rysunek 19: Konfiguracja zegara i częstotliwości taktowania peryferii STM32G431

5.5.2 STM32F746ZG

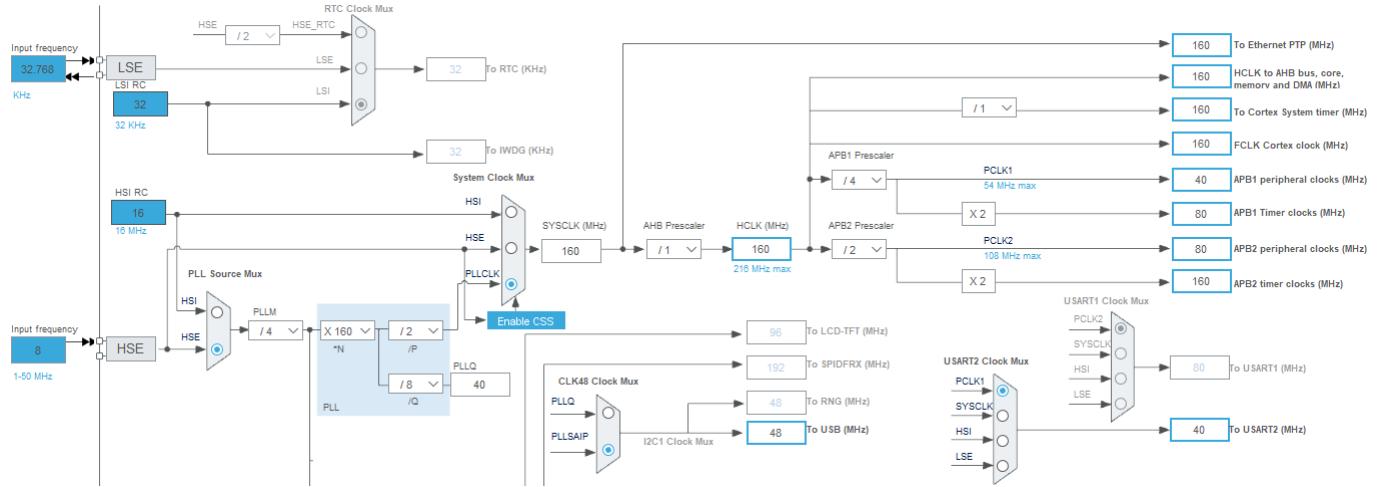
Zestaw prototypowy STM32 Nucleo F746ZG wyposażony jest w mikrokontroler STM32F746ZG. Rysunek 20, 21 przedstawia konfigurację, opis wykorzystanych pinów i ustawienia częstotliwości taktowania poszczególnych peryferii za pomocą aplikacji STM32CubeMX.

Dokonane ustawienia:

- PC1, PA1, PA2, PA7, PC4, PC5, PB13, PG11, PG13 tworzą interfejs Ethernet w trybie RMII,
- PA4, PA6, PB1 są kanałami przetworników ADC do pomiarów prądów fazowych,
- PE9 jest wejściem kanału licznika 1 do wyzwolenia konwersji przetworników ADC oraz synchronizacji pomiarów czujników ze sterownikiem B-G431-ESC1,
- PD5, PA3 - port szeregowy UART o przepustowości 115200 Bits/s,
- PC6 jest wejściem licznika 8 dla jednego z kanałów przetwornika obrotowo-impulsowego służącego do pomiaru prędkości. Licznik jest ustawiony w trybie Reset Mode co oznacza że każde zbocze narastające resetuje jego wartość, dzięki czemu można wyznaczyć okres między impulsami.
- PD12, PD13 są wejściami licznika 4 dla kanałów przetwornika obrotowo-impulsowego. Licznik jest ustawiony w trybie Encoder Mode, który pozwala na dodawanie i odejmowanie zliczonych impulsów z przetwornika w zależności od kierunku obrotu.



Rysunek 20: Konfiguracja mikrokontrolera STM32F746ZG



Rysunek 21: Konfiguracja zegara i częstotliwości taktowania periferii STM32F746ZG

5.5.3 Przedstawienie głównych funkcji w kodzie źródłowym

Kod programu został napisany za pomocą aplikacji STM32CubeIDE w języku C. Listing 1 przedstawia wybrane funkcje i ich opis w postaci komentarzy.

Listing 1:

```

1 //funkcja inicjalizująca - uruchamiana przy starcie programu
2 //przypisana do przycisku WYSŁIJ USTAWIENIA przy pierwszym uruchomieniu (interfejs użytkownika)
3 //wyzerowanie położenia wirnika, uruchamia liczniki, przetworniki ADC
4 //inicjalizuje struktury regulatorów PID
5 void start_up(void)
6 {
7     if(HAL_OK== ((HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED)) &&
8         HAL_ADCEx_Calibration_Start(&hadc2, ADC_SINGLE_ENDED)))
9     {
10         if(HAL_OK== (HAL_OPAMPEx_SelfCalibrateAll(&hopamp1, &hopamp2, &hopamp3)))
11         {
12             //linia 10 – 45 algorytm wyzerowania położenia wirnika
13             //ustawienie licznika TIM1 odpowiedzialnego za generowanie sygnałów PWM
14             TIM1->ARR= TIM1_ARR;
15             TIM1->PSC= TIM1_PSC;
16
17             TIM1->CCR1=(TIM1->ARR/10);
18             TIM1->CCR2=0;
19             TIM1->CCR3=0;
20             TIM1->CCR4=TIM1_CCR4;
21
22             HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
23             HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
24             HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
25             HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
26             HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
27             HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_3);
28             //HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
29
30             HAL_Delay(800);
31
32             //uruchomienie licznika do wyznaczenia kąta położenia wirnika
33             TIM4->ARR= TIM4_ARR;
34             TIM4->PSC= TIM4_PSC;
35             HAL_TIM_Base_Start_IT(&htim4);
36             HAL_TIM_Encoder_Start_IT(&htim4, TIM_CHANNEL_1);
37             HAL_TIM_Encoder_Start_IT(&htim4, TIM_CHANNEL_2);
38
39             HAL_Delay(400);
40
41             TIM1->CCR1=0;
42             TIM1->CCR2=0;
43             TIM1->CCR3=0;
44
45             HAL_Delay(200);
46
47             //uruchomienie licznika do wyznaczenia prędkości obrotowej wirnika
48             TIM8->ARR= TIM8_ARR;
49             TIM8->PSC= TIM8_PSC;
50             HAL_TIM_IC_Start(&htim8, TIM_CHANNEL_2);
51

```

```

52
53 //uruchomienie wzmacniaczy operacyjnych dla przetowrników ADC
54 HAL_OPAMP_Start(&hopamp1);
55 HAL_OPAMP_Start(&hopamp2);
56 HAL_OPAMP_Start(&hopamp3);
57
58 //uruchomienie przetworników ADC z obsługą zdarzeń
59 HAL_ADCEx_InjectedStart_IT(&hadc1);
60 HAL_ADCEx_InjectedStart_IT(&hadc2);
61
62 //inicjalizacja struktur regulatorów PID
63 set_d=0;
64 pid_d.Kp=1;
65 pid_d.Ki=1;
66 pid_d.Kd=0;
67 arm_pid_init_f32(&pid_d, 1);
68
69 //regulator prądu Id
70 set_q=0.7;
71 pid_q.Kp=4;
72 pid_q.Ki=1;
73 pid_q.Kd=0;
74 arm_pid_init_f32(&pid_q, 1);
75
76 //regulator prądu Iq
77 set_speed=2200;
78 pid_iq_speed.Kp=5;
79 pid_iq_speed.Ki=5;
80 pid_iq_speed.Kd=0;
81 arm_pid_init_f32(&pid_iq_speed, 1);
82
83 //regulator prędkości
84 set_angle=20000;
85 pid_angle.Kp=5;
86 pid_angle.Ki=5;
87 pid_angle.Kd=0;
88 arm_pid_init_f32(&pid_angle, 1);
89 }
90 }
91
92 }
93
94 // funkcja odpowiedzialna za uruchomienie generowania sygnałów PWM
95 // przypisana do przycisku START(interfejs użytkownika), uruchamia napęd
96 void start(void)
97 {
98     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
99     HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
100    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
101    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
102    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
103    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_3);
104    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
105 }
106
107 // funkcja odpowiedzialna za zatrzymanie generowania sygnałów PWM
108 // przypisana do przycisku STOP(interfejs użytkownika), zatrzymuję napęd
109 void stop(void)
110 {
111     TIM1->CCR1=0;
112     TIM1->CCR2=0;
113     TIM1->CCR3=0;
114
115     HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
116     HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_1);
117     HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
118     HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_2);
119     HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
120     HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_3);
121     HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
122     HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_4);
123
124     arm_pid_reset_f32(&pid_d);
125     arm_pid_reset_f32(&pid_q);
126     arm_pid_reset_f32(&pid_iq_speed);
127     arm_pid_reset_f32(&pid_angle);
128
129 }
130
131
132 // funkcja pobiera współrzędne wektora [alpha, beta] i oblicza moduł oraz kąt
133 void AlphaBeta_To_Angle_Vref(float32_t alpha, float32_t beta, float32_t *angle_current_rad, float32_t *
134 Vref)
135 {
136     *angle_current_rad = atan2f(beta, alpha);
137     arm_sqrt_f32(((alpha*alpha)+(beta*beta)), Vref);

```

```

138     if(*Vref>=sv_Vdc_limit) // saturacja Vref
139         *Vref=sv_Vdc_limit;
140     }
141
142 // funkcja pobiera kat wektora [rad] i oblicza w ktorym sektorze znajduje sie ten wektor
143 void Angle_To_Sector(float32_t angle_current_rad ,uint8_t *sector)
144 {
145
146     if((angle_current_rad>0) && (angle_current_rad<=1.047197)) // pi/3
147         *sector=1;
148     else if((angle_current_rad >1.047197) && (angle_current_rad<=2.094395)) //2/3*pi
149         *sector=2;
150     else if((angle_current_rad >2.094395) && (angle_current_rad<=3.141593))
151         *sector=3;
152     else if((angle_current_rad >-3.141593) && (angle_current_rad<=-2.094395))
153         *sector=4;
154     else if((angle_current_rad >-2.094395) && (angle_current_rad<=-1.047197))
155         *sector=5;
156     else if ((angle_current_rad >-1.047197) && (angle_current_rad<=0))
157         *sector=6;
158     else{}
159 }
160
161
162 // funkcja pobiera kat, modul wektora i numer sektora w ktorym wektor sie znajduje
163 // na podstawie tych informacji oblicza szerokosc
164 // wypeplnienia impulsow dla trzech sygnalow PWM
165 void SVPWM(uint8_t sector ,float32_t angle_current_rad ,float32_t Vref, float32_t T[], float32_t T_gate
166 [], float32_t *S1, float32_t *S2, float32_t *S3)
167 {
168
169     T[1]= (1.7320508 * (( Vref * sv_Tz)/Vdc)) * arm_sin_f32((sector * 1.047197) - (
170         angle_current_rad)); // pi/3 = 1,0472 sqrt(3)=1.7320508
171     T[2]= (1.7320508 * (( Vref * sv_Tz)/Vdc)) * arm_sin_f32((-sector-1) * 1.047197) +
172         angle_current_rad) ;
173     T[0]=sv_Tz-T[1]-T[2];
174
175     t1=T[1];
176     t2=T[2];
177     t3=T[0];
178
179     T_gate[0]=(T[0]/2);
180     T_gate[1]= T[1]+(T_gate[0]);
181     T_gate[2]= T[2]+(T_gate[0]);
182     T_gate[3]= T[1]+T[2]+(T_gate[0]);
183
184     if(sector == 1)
185     {
186         *S1=T_gate[3];
187         *S2=T_gate[2];
188         *S3=T_gate[0];
189     }
190     else if(sector == 2)
191     {
192         *S1=T_gate[1];
193         *S2=T_gate[3];
194         *S3=T_gate[0];
195     }
196     else if(sector == 3)
197     {
198         *S1=T_gate[0];
199         *S2=T_gate[3];
200         *S3=T_gate[2];
201     }
202     else if(sector == 4)
203     {
204         *S1=T_gate[0];
205         *S2=T_gate[1];
206         *S3=T_gate[3];
207     }
208     else if(sector == 5)
209     {
210         *S1=T_gate[2];
211         *S2=T_gate[0];
212         *S3=T_gate[3];
213     }
214     else if(sector == 6)
215     {
216         *S1=T_gate[3];
217         *S2=T_gate[0];
218         *S3=T_gate[1];
219     }
220 }
221

```

```

222 // funkcja obsługi zdarzenia wywoływana po zakończeniu konwersji ADC
223 // główna funkcja odpowiedzialna za dokonywanie obliczeń układów regulacji
224 void HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef *hadc)
225 {
226     // obliczanie prędkości obrotowej wirnika
227     capture_tim8_ccr2= TIM8->CCR2;
228     speed=revolution_per_min/capture_tim8_ccr2;
229
230     // sprawdzenie kierunku obrotu
231     // do zmiennej direction jest przypisany bit z rejestru CR1 licznika
232     // który zawiera informacje o tym czy licznik jest inkrementowany czy dekrementowany
233     if(direction!=0)
234         speed=-speed;
235
236     // odczytanie wartości prądów z przetowrników ADC
237     adc_la= HAL_ADCEx_InjectedGetValue(&hadc1 , ADC_INJECTED_RANK_1);
238     while((hadc1.Instance->ISR &= (0x1<<5))!=0){}
239     adc_lc =HAL_ADCEx_InjectedGetValue(&hadc1 , ADC_INJECTED_RANK_2);
240     while((hadc1.Instance->ISR &= (0x1<<5))!=0){}
241     adc_lb =HAL_ADCEx_InjectedGetValue(&hadc2 , ADC_INJECTED_RANK_1);
242     while((hadc2.Instance->ISR &= (0x1<<5))!=0){}
243
244     // wyznaczenie offsetu dla przetwornika ADC
245     if(index_event_adc<300)
246     {
247         la=0;
248         lb=0;
249         lc=0;
250         index_event_adc++;
251
252     }
253     else if(index_event_adc == 300)
254     {
255         offset1=adc_la;
256         offset2=adc_lb;
257         offset3=adc_lc;
258         index_event_adc++;
259     }
260     else
261     {
262
263         adc_la=(adc_la-offset1);
264         adc_lb=(adc_lb-offset2);
265         adc_lc=(adc_lc-offset3);
266
267         // konwersja wartości ADC na wartość wyrażoną w amperach [A]
268         la-=adc_la/33.0;
269         lb-=adc_lb/33.0;
270         lc-=adc_lc/33.0;
271
272         // transformacja clark'a
273         arm_clarke_f32(la , lb , &lalpha , &lbeta );
274
275         angle_rotor_deg=TIM4->CCR1;
276         arm_sin_cos_f32(angle_rotor_deg , &pSinVal , &pCosVal );
277
278         // transformacja park'a
279         arm_park_f32(lalpha , lbeta , &Id , &Iq , pSinVal , pCosVal );
280
281         // obliczanie położenia wirnika [stopnie]
282         if(counter_angle>=0)
283             angle=angle_rotor_deg + (counter_angle * 360);
284         else
285             angle=-angle_rotor_deg + (counter_angle * 360);
286
287         // włączenie/wyłącznie regulatora położenia (interfejs użytkownika)
288         if(allow_angle[0]=='y')
289         {
290             // możliwość zmniejszenia częstotliwości obliczania pętli regulacji położenia
291             // domyślnie obliczenia są wykonywane za każdym uruchomieniem
292             // funkcji obsługi przerwania ADC
293             index_angle_loop++;
294             if(index_angle_loop==1)
295             {
296                 //uchyb położenia
297                 e_angle=set_angle-angle;
298                 //zachowanie zmiennej z poprzedniego cyklu na potrzeby anti-windup
299                 angle_prev=pid_angle.state[2];
300                 // obliczenie wartości wyjścia regulatora PID
301                 angle_speed=arm_pid_f32(&pid_angle , e_angle );
302                 //saturacja i anti-windup
303                 if(angle_speed>=set_speed)
304                 {
305                     pid_angle.state[2]=angle_prev;
306                     angle_speed=set_speed;
307                 }
308             }

```

```

309
310         if( angle_speed<=(-set_speed))
311     {
312         pid_angle.state[2]=angle_prev;
313         angle_speed=(-set_speed);
314     }
315     set_sp= angle_speed;
316 }
317 if(index_angle_loop==1)
318 index_angle_loop=0;
319
320 }
321 else
322 {
323     index_angle_loop=0;
324     set_sp=set_speed;
325 }
326
327 //możliwość zmniejszenia częstotliwości obliczania pętli regulacji prędkości
328 // domyślnie obliczenia są wykonywane za każdym uruchomieniem
329 //funkcji obsługi przerwania ADC
330 index_speed_loop++;
331 if(index_speed_loop==1)
332 {
333     //uchyb prędkości obrotowej
334     e_speed=set_sp-speed;
335     //zachowanie zmiennej z poprzedniego cyklu na potrzeby anti-windup
336     iq_speed_prev=pid_iq_speed.state[2];
337     // obliczenie wartości wyjścia regulatora PID
338     iq_speed=arm_pid_f32(&pid_iq_speed, e_speed);
339     // saturacja i anti-wind-up
340     if(iq_speed>current_limit_max_iq)
341     {
342         pid_iq_speed.state[2]=iq_speed_prev;
343         iq_speed=current_limit_max_iq;
344     }
345     if(iq_speed<=(-current_limit_max_iq))
346     {
347         pid_iq_speed.state[2]=iq_speed_prev;
348         iq_speed=(-current_limit_max_iq);
349     }
350 }
351 if(index_speed_loop==1)
352 index_speed_loop=0;
353
354 //obliczenie uchybu prądu Id
355 ed=set_d_Id;
356 //zachowanie zmiennej z poprzedniego cyklu na potrzeby anti-windup
357 Vd_prev=pid_d.state[2];
358 // obliczenie wartości wyjścia regulatora PID
359 Vd=arm_pid_f32(&pid_d, ed);
360 // saturacja i anti-wind-up
361 if(Vd>sv_Vdc_limit)
362 {
363     pid_d.state[2]=Vd_prev;
364     Vd=sv_Vdc_limit;
365 }
366
367 if(Vd<=(-sv_Vdc_limit))
368 {
369     pid_d.state[2]=Vd_prev;
370     Vd=(-sv_Vdc_limit);
371 }
372
373 //obliczenie uchybu prądu Iq
374 eq=iq_speed-iq;
375 //zachowanie zmiennej z poprzedniego cyklu na potrzeby anti-windup
376 Vq_prev=pid_q.state[2];
377 // obliczenie wartości wyjścia regulatora PID
378 Vq=arm_pid_f32(&pid_q, eq);
379 // saturacja i anti-wind-up
380 if(Vq>sv_Vdc_limit)
381 {
382     pid_q.state[2]=Vq_prev;
383     Vq=sv_Vdc_limit;
384 }
385
386 if(Vq<=(-sv_Vdc_limit))
387 {
388     pid_q.state[2]=Vq_prev;
389     Vq=(-sv_Vdc_limit);
390 }
391
392 //odwrotna transformacja park'a
393 arm_inv_park_f32(Vd, Vq, &Valpha, &Vbeta, pSinVal, pCosVal);
394 // obliczenie modułu wektora i kąta
395 AlphaBeta_To_Angle_Vref(Valpha, Vbeta, &angle_current_rad, &Vref);

```

```

396 //wyznaczenie w którym sektorze jest wektor prądu
397 Angle_To_Sector(angle_current_rad, &sector);
398 //wyznaczenie szerokości impulsów i przypisanie ich do rejestrów
399 //kanałów licznika, które sterują bramkami przetwornika
400 SVPWM(sector, angle_current_rad, Vref, sv_T, sv_T_gate, &sv_S1, &sv_S2, &sv_S3);
401 TIM1->CCR1 = sv_S1;
402 TIM1->CCR2 = sv_S2;
403 TIM1->CCR3 = sv_S3;
404 }
405 //ponowne uruchomienie przetwornika ADC w tryb nasłuchu
406 HAL_ADCEx_InjectedStart_IT(&hadc1);
407 HAL_ADCEx_InjectedStart_IT(&hadc2);
408 }
409
410 // funkcja obsługi przerwania dla portu szeregowego
411 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
412 {
413     //sprawdzenie czy właściwy port szeregowy wywołał zdarzenie obsługi przerwania
414     if(huart->Instance==USART2)
415     {
416         //funkcja konfiguracyjna start_up()
417         //wywoływana tylko przy pierwszym uruchomieniu
418         if(config==0)
419         {
420             config=1;
421             start_up();
422         }
423         //pobranie łańcucha znaków z portu szeregowego i konwersja do obiektu JSON
424         cJSON * root = cJSON_Parse((char *)jstring);
425         cJSON * set = cJSON_GetObjectItemCaseSensitive(root, "set");
426         settings = atoi(cJSON_GetStringValue(set));
427         //sprawdzenie czy są to dane konfiguracyjne
428         //czy komenda o starcie/zatrzymaniu pracy napędu
429         if(settings==0)
430         {
431             cJSON * start_stop = cJSON_GetObjectItemCaseSensitive(root, "start_stop");
432             startstop = atoi(cJSON_GetStringValue(start_stop));
433
434             if(startstop==1)
435                 start1();
436             else
437                 stop();
438         }
439     }
440     else
441     {
442         cJSON * speed = cJSON_GetObjectItemCaseSensitive(root, "speed");
443         set_speed = atoi(cJSON_GetStringValue(speed));
444
445         cJSON * current = cJSON_GetObjectItemCaseSensitive(root, "current");
446         sscanf(cJSON_GetStringValue(current), "%f", &current_limit_max_iq);
447
448         cJSON * iq_Kp = cJSON_GetObjectItemCaseSensitive(root, "iq_Kp");
449         pid_q.Kp = atoi(cJSON_GetStringValue(iq_Kp));
450
451         cJSON * iq_Ki = cJSON_GetObjectItemCaseSensitive(root, "iq_Ki");
452         pid_q.Ki = atoi(cJSON_GetStringValue(iq_Ki));
453
454         cJSON * iq_Kd = cJSON_GetObjectItemCaseSensitive(root, "iq_Kd");
455         pid_q.Kd = atoi(cJSON_GetStringValue(iq_Kd));
456
457         cJSON * id_Kp = cJSON_GetObjectItemCaseSensitive(root, "id_Kp");
458         pid_d.Kp = atoi(cJSON_GetStringValue(id_Kp));
459
460         cJSON * id_Ki = cJSON_GetObjectItemCaseSensitive(root, "id_Ki");
461         pid_d.Ki = atoi(cJSON_GetStringValue(id_Ki));
462
463         cJSON * id_Kd = cJSON_GetObjectItemCaseSensitive(root, "id_Kd");
464         pid_d.Kd = atoi(cJSON_GetStringValue(id_Kd));
465
466         cJSON * speed_Kp = cJSON_GetObjectItemCaseSensitive(root, "speed_Kp");
467         pid_iq_speed.Kp = atoi(cJSON_GetStringValue(speed_Kp));
468
469         cJSON * speed_Ki = cJSON_GetObjectItemCaseSensitive(root, "speed_Ki");
470         pid_iq_speed.Ki = atoi(cJSON_GetStringValue(speed_Ki));
471
472         cJSON * speed_Kd = cJSON_GetObjectItemCaseSensitive(root, "speed_Kd");
473         pid_iq_speed.Kd = atoi(cJSON_GetStringValue(speed_Kd));
474
475         cJSON * a_set = cJSON_GetObjectItemCaseSensitive(root, "a_set");
476         sprintf(allow_angle, "%os", (char*)(cJSON_GetStringValue(a_set)));
477
478         cJSON * angle = cJSON_GetObjectItemCaseSensitive(root, "angle");
479         set_angle = atoi(cJSON_GetStringValue(angle));
480
481         cJSON * a_Kp = cJSON_GetObjectItemCaseSensitive(root, "a_Kp");
482         pid_angle.Kp = atoi(cJSON_GetStringValue(a_Kp));

```

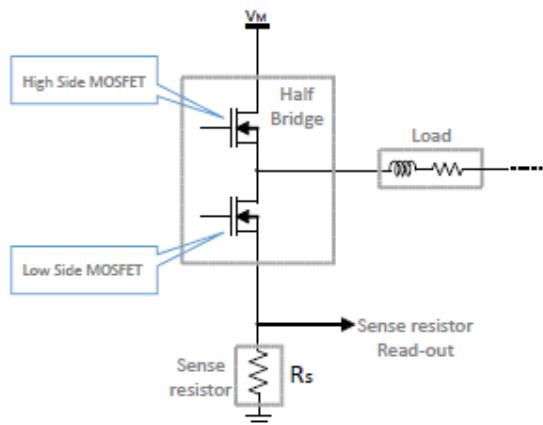
```

483
484     cJSON * a_Ki = cJSON_GetObjectItemCaseSensitive(root , "a_Ki");
485     pid_angle.Ki = atoi(cJSON_GetStringValue(a_Ki));
486
487     cJSON * a_Kd = cJSON_GetObjectItemCaseSensitive(root , "a_Kd");
488     pid_angle.Kd = atoi(cJSON_GetStringValue(a_Kd));
489     //usuńcie obiektu JSON
490     cJSON_Delete(root);
491     //inicjalizacja struktur regulatorów PID nowymi parametrami
492     arm_pid_init_f32(&pid_d, 1);
493     arm_pid_init_f32(&pid_q, 1);
494     arm_pid_init_f32(&pid_iq_speed, 1);
495     arm_pid_init_f32(&pid_angle, 1);
496 }
497 //ponowne uruchomienie nasłuchiwanie portu szeregowego
498 HAL_UART_Receive_IT(&huart2 , jstring ,size_uart_tab);
499
500 }
```

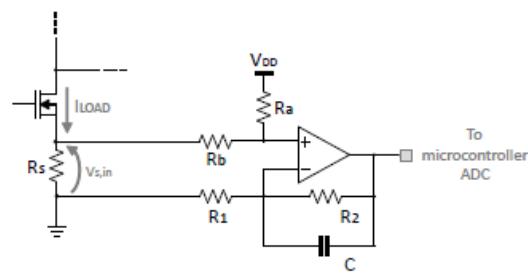
5.6 Implementacja układu regulacji

W zestawie rozwojowym B-G431-ESC został zaimplementowany układ regulacji prądu, prędkości i położenia dla napędu z silnikiem BLDC przy wykorzystaniu metod FOC i SVPWM.

Do mierzenia prądów fazowych zostały wykorzystane wbudowane, dedykowane układy. Mierzony jest spadek napięcia na rezystorach umieszczonych między dolną grupą tranzystorów mocy a masą. Na rysunku 22 przedstawiono sposób pomiaru. W wyniku bardzo małej rezystancji wynoszącej 0.03Ω sygnał jest wzmacniany przez wzmacniacz operacyjny i za pomocą przetwornika analogowo-cyfrowego uzyskiwane aktualne wartości prądów - rysunek 23.



Rysunek 22: Pomiar prądu [13]



Rysunek 23: Wzmacnianie mierzonego sygnału [13]

Pomiar prędkości odbywa się za pomocą enkodera. Licznik skonfigurowany w trybie Mode Reset resetuje się w momencie podania stanu wysokiego na jeden z kanałów, do którego podłączony jest enkoder. Znając częstotliwość zliczania i okres między wystąpieniem dwóch stanów wysokich można w łatwy sposób obliczyć aktualną prędkość.

Pomiar położenia wirnika również odbywa się przy pomocy enkodera. Licznik w konfiguracji Encoder Mode zlicza wykryte zbocza z dwóch kanałów, do których podłączone są dwa sygnały z czujnika ruchu przesunięte w fazie. W zależności od kierunku obrotu licznik jest inkrementowany lub dekrementowany. Po zliczeniu ilości impulsów równej rozdzielcości enkodera licznik zlicza od zera.

Biblioteka CMSIS-DSP udostępnia szereg funkcji dla obliczeń transformat, regulatorów PID oraz funkcji trygonometrycznych. Funkcja obliczająca wartość sygnału wyjściowego regulatora PID ma postać [1]:

```
float32_t arm_pid_f32 (arm_pid_instance_f32 *S, float32_t in)
```

gdzie:

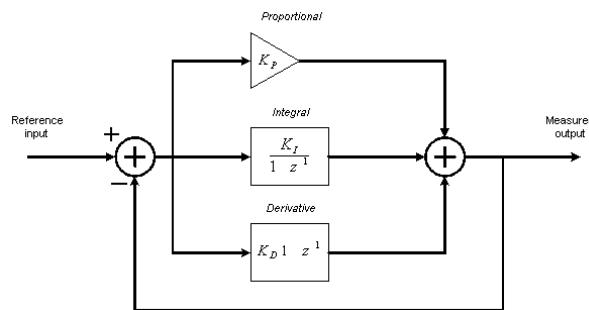
float32_t in – uchyb

arm_pid_instance_f32 *S - struktura [A0, A1, A2, Kp, Ki, Kd, state[3]] opisana wyrażeniem 22:

$$\begin{cases} y[n] = y[n-1] + A0 * x[n] + A1 * x[n-1] + A2 * x[n-2] \\ A0 = Kp + Ki + Kd \\ A1 = (-Kp) - (2 * Kd) \\ A2 = Kd \end{cases} \quad (22)$$

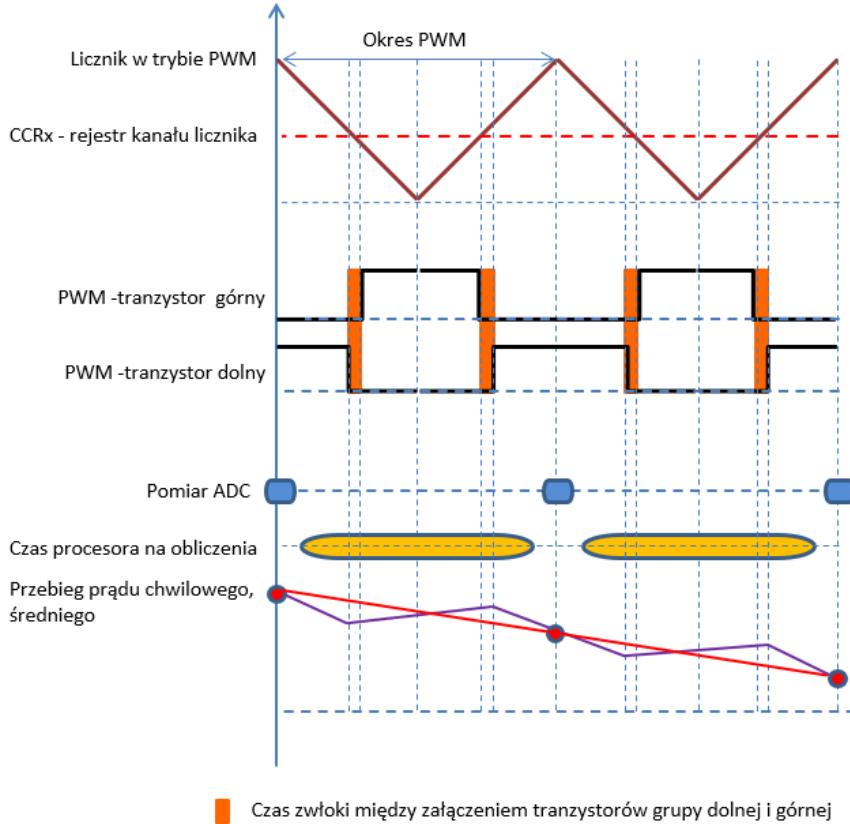
Kp- wzmacnienie, Ki- stała całkowania, Kd- stała różniczkowania

Rysunek poniżej reprezentuje schemat blokowy regulatora PID biblioteki CMSIS-DSP.



Rysunek 24: Schemat regulatora PID biblioteki CMSIS-DSP [1]

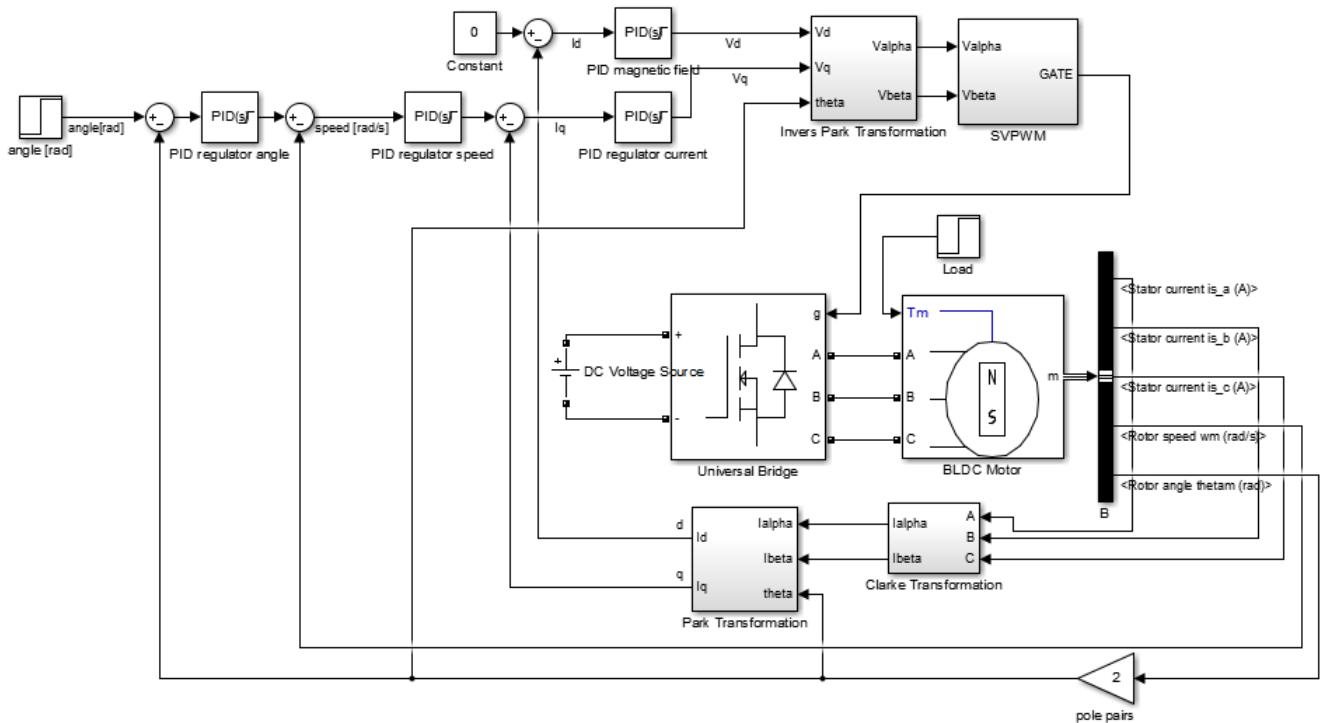
Modulacja wektorowa SVPWM składa się z funkcji, które na podstawie zadanego wektora referencyjnego wyznacza trzy sygnały PWM oraz licznika skonfigurowanego w trybie PWM Center Mode z trzema kanałami do wygenerowania sygnałów PWM dla trzech par tranzystorów przekształtnika energoelektronicznego. Dodatkowo czwarty kanał przeznaczony jest do określenia momentu uruchomienia przetwornika ADC i przypada on w połowie okresu PWM, gdzie wszystkie tranzystory grupy dolnej są włączone a grupy górnej wyłączone. Na rysunku 25 przedstawiono synchronizację wyzwalania przetwornika ADC z sygnałami PWM.



Rysunek 25: Synchronizacja sygnału PWM z wyzwalaniem przerwania od przetwornika ADC

W funkcji obslugi przerwania od zdarzenia przetwornika ADC obliczane są transformaty, funkcje regulatorów PID, transformaty odwrotne i funkcje modulacji wektorowej. W wyniku tych operacji, za pomocą licznika z kanałami sterującymi bramkami tranzystorów przekształtnika, zostają wygenerowane trzy sygnały PWM.

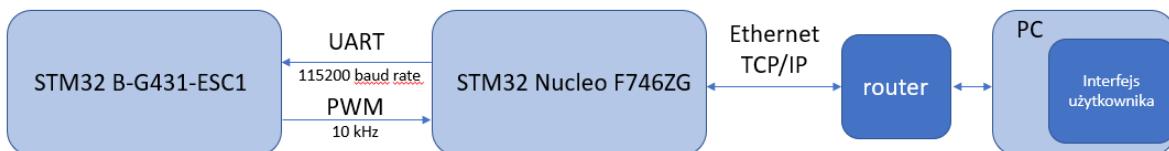
Do regulacji prądu, prędkości i położenia zastosowano cztery regulatorы PID połączone ze sobą w sposób kaskadowy. Wynik regulatora zewnętrznej pętli wprowadzany jest na wejście regulatora pętli wewnętrznej. Schemat przedstawiono na rysunku 26. Regulacja odbywa się z częstotliwością wynoszącą 10kHz.



Rysunek 26: Model napędu silnika BLDC

5.7 Komunikacja

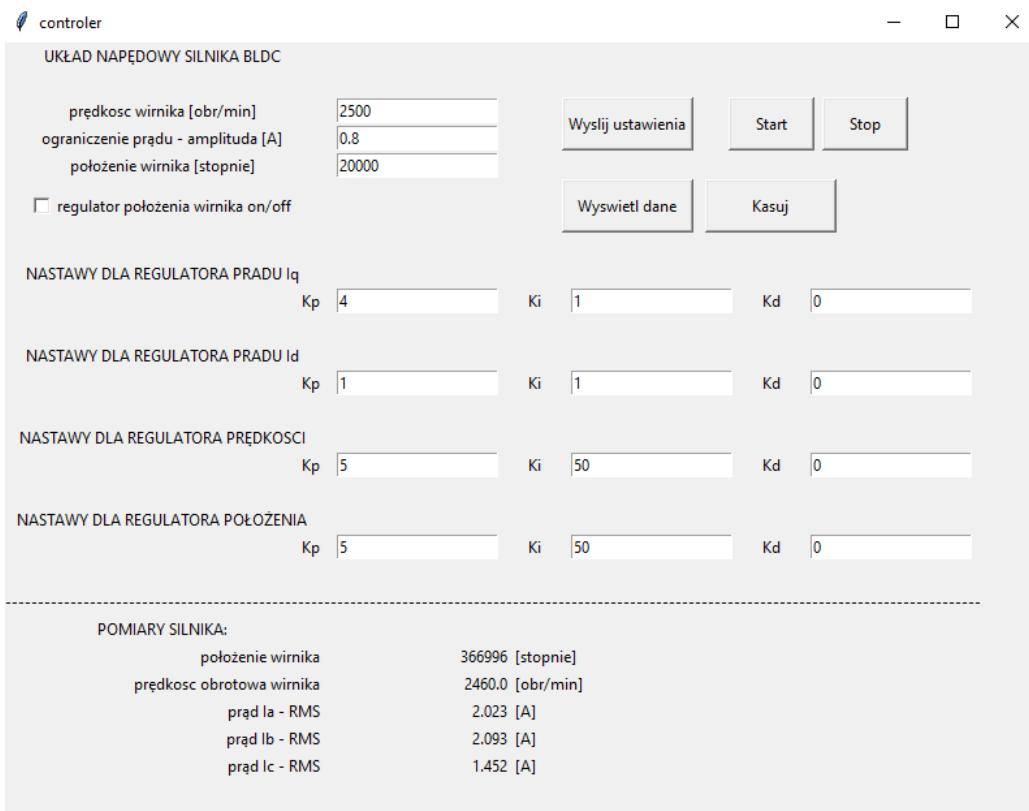
Zestaw ewaluacyjny Nucleo F746ZG jest głównym węzłem komunikacyjnym. Łączy się ze sterownikiem B-G431-ESC1 za pomocą portu szeregowego UART o prędkości transmisji 115200 bitów na sekundę. Za pomocą tego portu wysyłane są wartości zadane prądu, prędkości, położenia wirnika, nastawy regulatorów oraz ustawienia konfiguracyjne. B-G431-ESC1 generuje sygnał PWM do synchronizacji pomiarów i obliczeń z płytą Nucleo F746ZG. Rysunek 27 przedstawia graficzny schemat układu komunikacji. Nucleo F746ZG, wyposażony w Ethernet i złącze RJ45, łączy się z interfejsem użytkownika zaimplementowanym na zdalnym komputerze za pomocą przewodu sieciowego i routera. Informacje przekazywane są za pomocą protokołu TCP/IP w postaci łańcuchów znaków we formacie JSON (ang. JavaScript Object Notation). JSON jest formatem zapisu struktur danych przeznaczonym do wymiany informacji między aplikacjami. Dane zapisywane w notacji atrybut - wartość charakteryzuje się prostotą i czytelnością. Za pomocą interfejsu użytkownika wysyłane są wartości zadane, nastawy regulatorów i ustawienia a odbierane bieżące wartości prądów, prędkości i położenia wirnika.



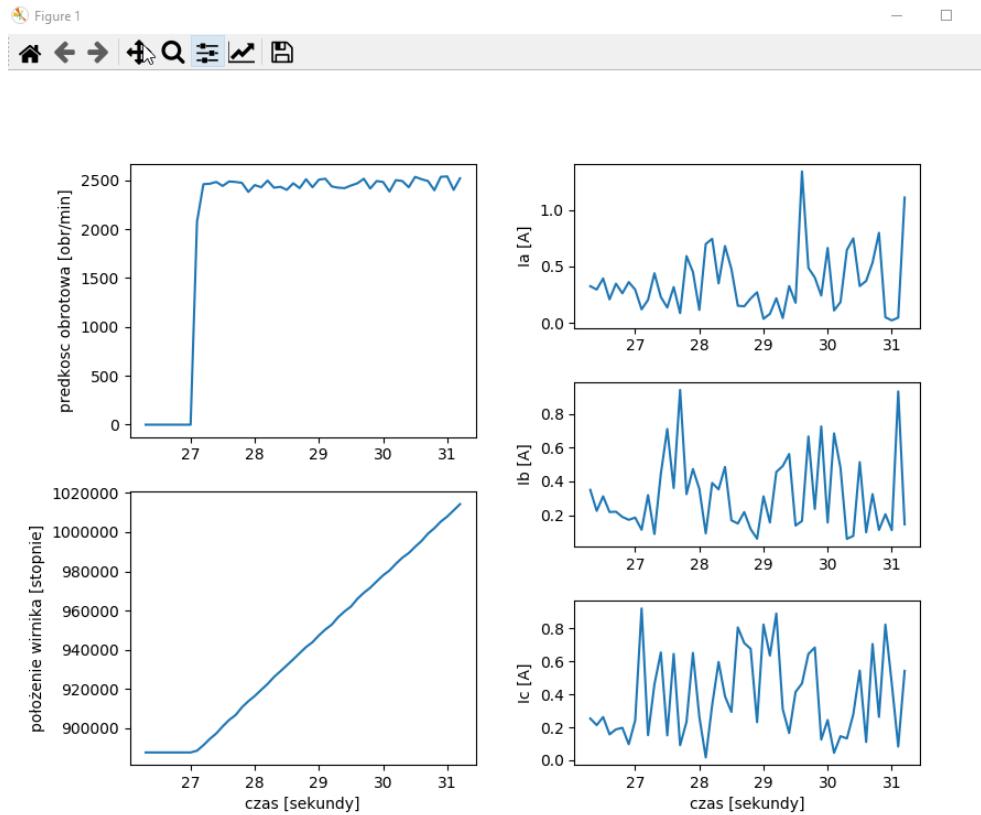
Rysunek 27: Schemat układu komunikacji

5.8 Interfejs użytkownika

Interfejs użytkownika został napisany w języku Python przy pomocy oprogramowania Spyder w środowisku Anaconda. Aplikacja za pomocą Ethernetu i protokołu TCP/IP łączy się z płytą Nucleo F746ZG na której zaimplementowany jest serwer TCP. Informacje wysyłane są w postaciłańcuchów znaków we formacie JSON. Aplikacja wysyła do serwera zapytania cyklicznie co 20 ms. W odpowiedzi serwer zwraca informacje o bieżących wartościach prądów fazowych, prędkości i położeniu wirnika. Na rysunku 28 i 29 przedstawiony jest interfejs aplikacji. Użytkownik ma możliwość określania wartości zadanych prądu, prędkości, położenia wirnika oraz nastaw dla regulatorów PID. Dodatkowo istnieje możliwość wyłączenia regulatora położenia i pozostawienia tylko aktywnego regulatora prądu i prędkości. Przycisk *Wyślij ustawienia* powoduje dodanie do zapytania danych konfiguracyjnych i aktualizowanie ustawień napędu. *Start, stop* uruchamia i zatrzymuje prace silnika. Przycisk *Wyświetl dane* przedstawia aktualne informacje z czujników w części aplikacji oznaczonej jako *POMIARY SILNIKA* oraz dodatkowo w osobnym oknie dialogowym przedstawia dane w postaci wykresów. Przycisk *Kasuj* zatrzymuje i czyści obszar wyświetlania wykresów.



Rysunek 28: Interfejs użytkownika

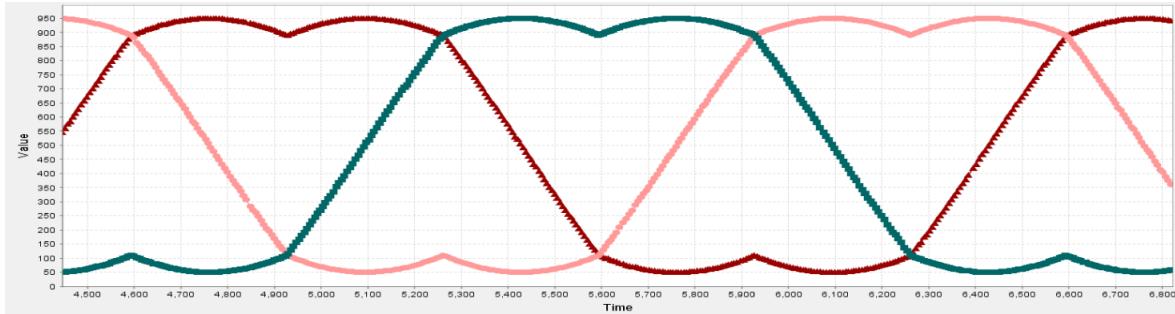


Rysunek 29: Interfejs użytkownika

6 Testowanie układu napędowego

6.1 Sprawdzenie poprawności działania funkcji SVPWM

Jednym z elementów układu jest modulacja SVPWM. Składa się one ze zbioru funkcji, które zostały zaimplementowane i przetestowane w celu sprawdzenia poprawności działania przed przystąpieniem do testowania wraz z regulatorami PID. Do funkcji zostały wprowadzone trzy sygnały sinusoidalne o amplitudzie 1000, częstotliwości 0.5 Hz, przesunięte w fazie o 120 stopni. W wyniku działania SVPWM generowany jest referencyjny wektor napięciowy. Aby uniknąć zjawiska nadmodulacji i wynikającego z niej zwiększenia sygnału sinusoidalnego amplituda jest ograniczona do wartości $\sqrt{3}/2 * Vdc$. Wektor referencyjny za pomocą transformat odwrotnych jest przekształcany w trzy przebiegi pokazane na rysunku 30. W kolejnym etapie są one porównywane ze sygnałem trójkątnym tworząc przebiegi prostokątne i wprowadzone na przetwornik energoelektroniczny. Do podglądu przebiegów zostało wykorzystane oprogramowanie STMSStudio, które umożliwia graficzne wyświetlenie stanu zmiennych programu w czasie rzeczywistym.



Rysunek 30: Przebiegi generowanych sygnałów za pomocą SVPWM na podstawie trzech wejściowych sygnałów sinusoidalnych

6.2 Testowanie układu regulacji

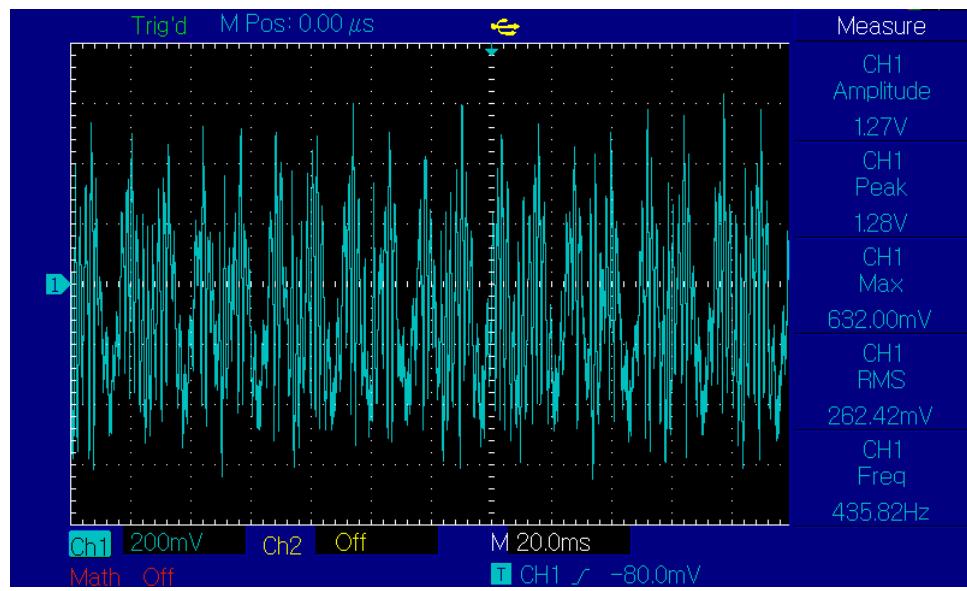
Układ został przetestowany dla nastaw regulatorów dobranych doświadczalnie i opisanych na rysunku 31. Pomiary zostały wykonane za pomocą oscyloskopu UNI-T UTD2102CEX i oprogramowania STM-Studio do podglądu zmiennych programu w czasie rzeczywistym.

NASTAWY DLA REGULATORA PRĄDU I_q	K_p <input type="text" value="4"/>	K_i <input type="text" value="1"/>	K_d <input type="text" value="0"/>
NASTAWY DLA REGULATORA PRĄDU I_d	K_p <input type="text" value="1"/>	K_i <input type="text" value="1"/>	K_d <input type="text" value="0"/>
NASTAWY DLA REGULATORA PRĘDKOŚCI	K_p <input type="text" value="5"/>	K_i <input type="text" value="50"/>	K_d <input type="text" value="0"/>
NASTAWY DLA REGULATORA POŁOŻENIA	K_p <input type="text" value="5"/>	K_i <input type="text" value="50"/>	K_d <input type="text" value="0"/>

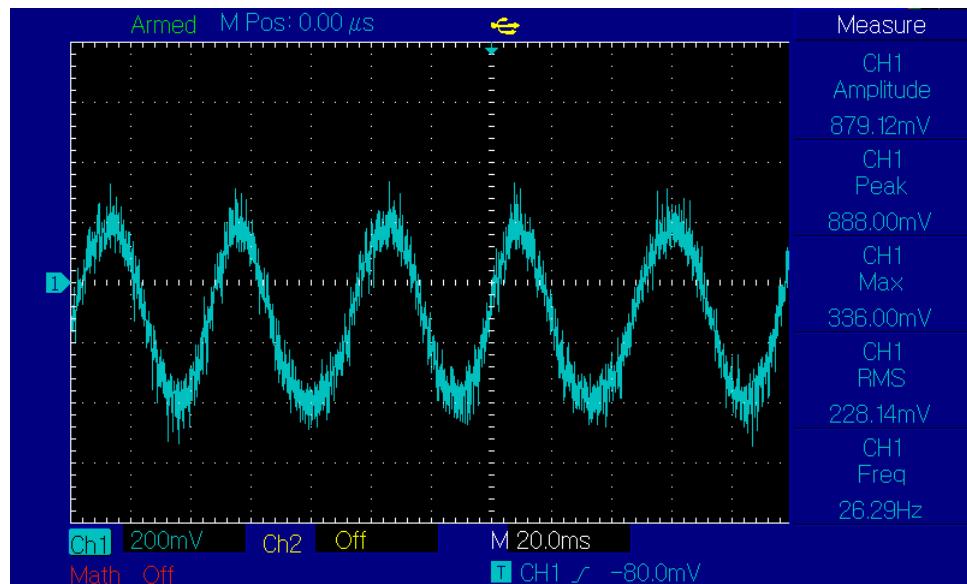
Rysunek 31: Nastawy regulatorów PID

6.2.1 Regulacja prądu

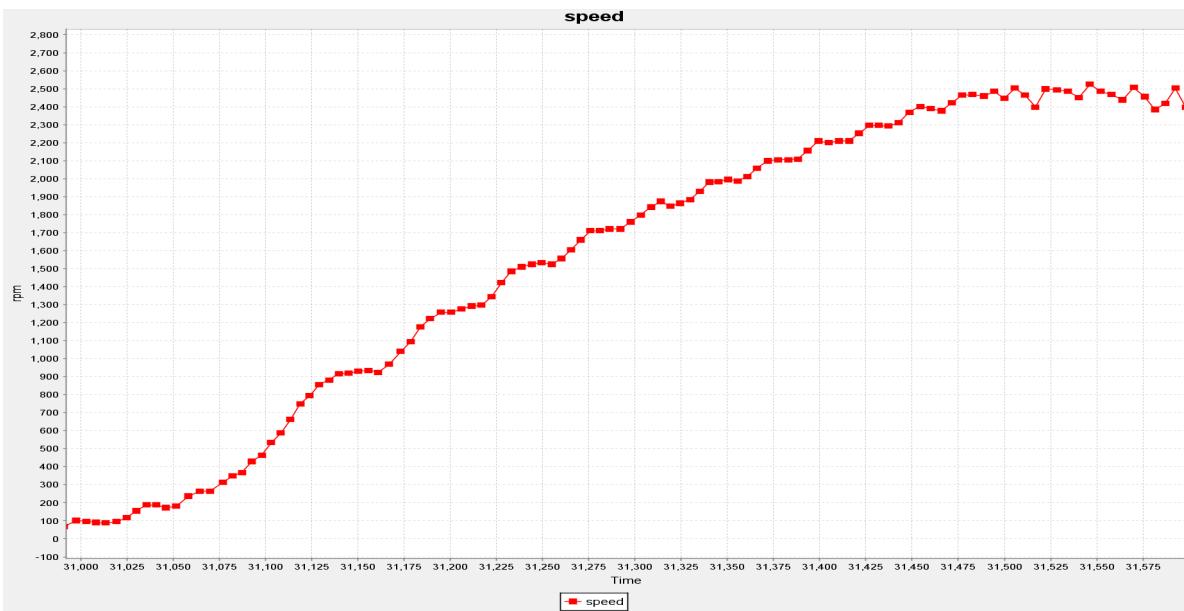
W celu sprawdzenia poprawności działania regulatorów wykonano pomiary dla trzech wartości amplitud składowej prądu I_q : 0.3A, 0.5A, 1A dla przypadku gdzie wirnik nie został obciążony i dla przypadku działania zewnętrznego momentu oporowego. Wartość zadana prędkości obrotowej wynosi 2500 obr/min. Przebiegi przedstawione są rysunkach od 32 do 45. Aby dokonać pomiaru prądu przy pomocy oscyloskopu do wyprowadzeń uzwojeń silnika zostały podłączone rezystory o wartości 1Ω dzięki czemu spadek napięcia o wartości 1V odpowiada prądowi o wartości 1A.



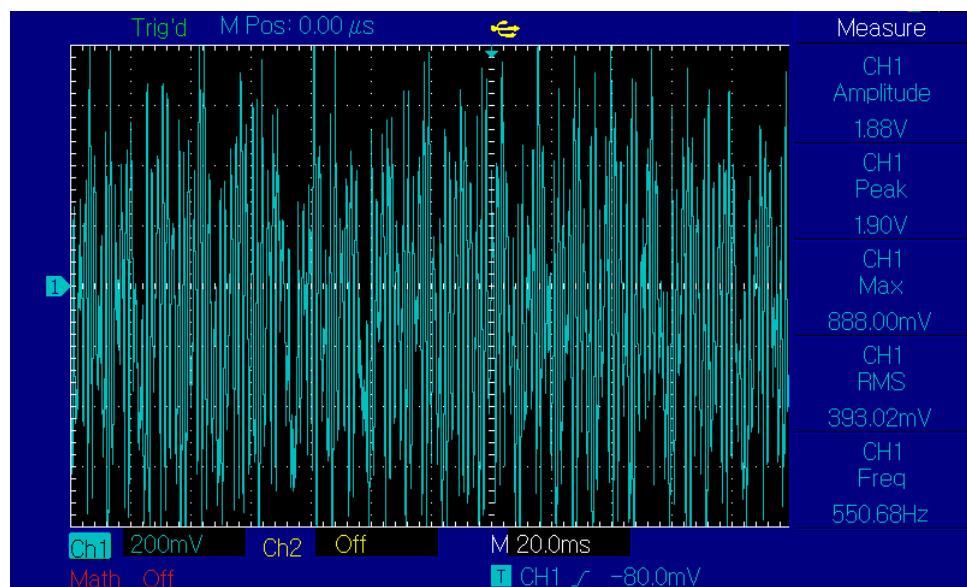
Rysunek 32: Przebieg prądu dla wartości zadanej amplitudy 0.3A bez obciążenia zewnętrzny momentem oporowym



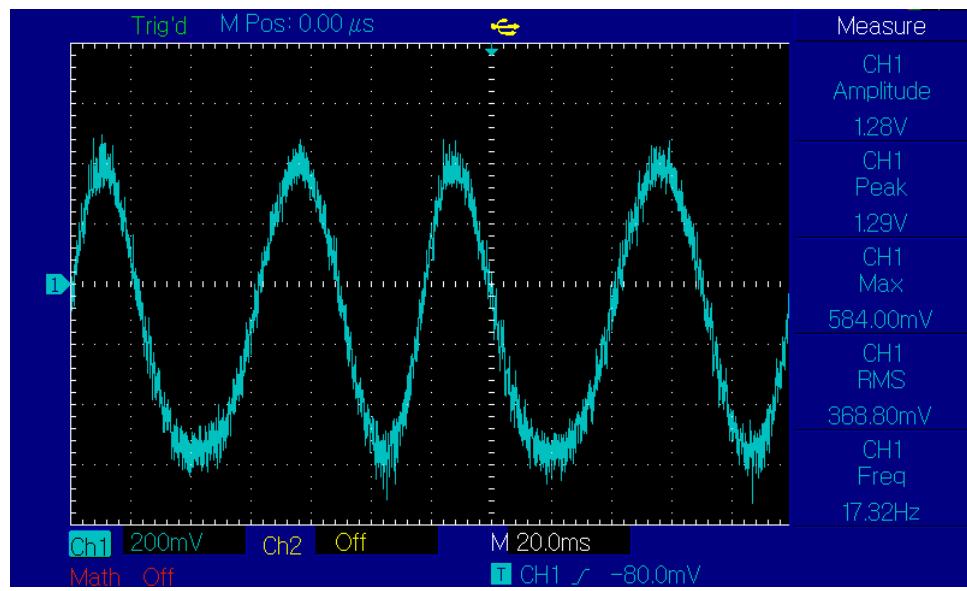
Rysunek 33: Przebieg prądu dla wartości zadanej amplitudy 0.3A z obciążeniem zewnętrzny



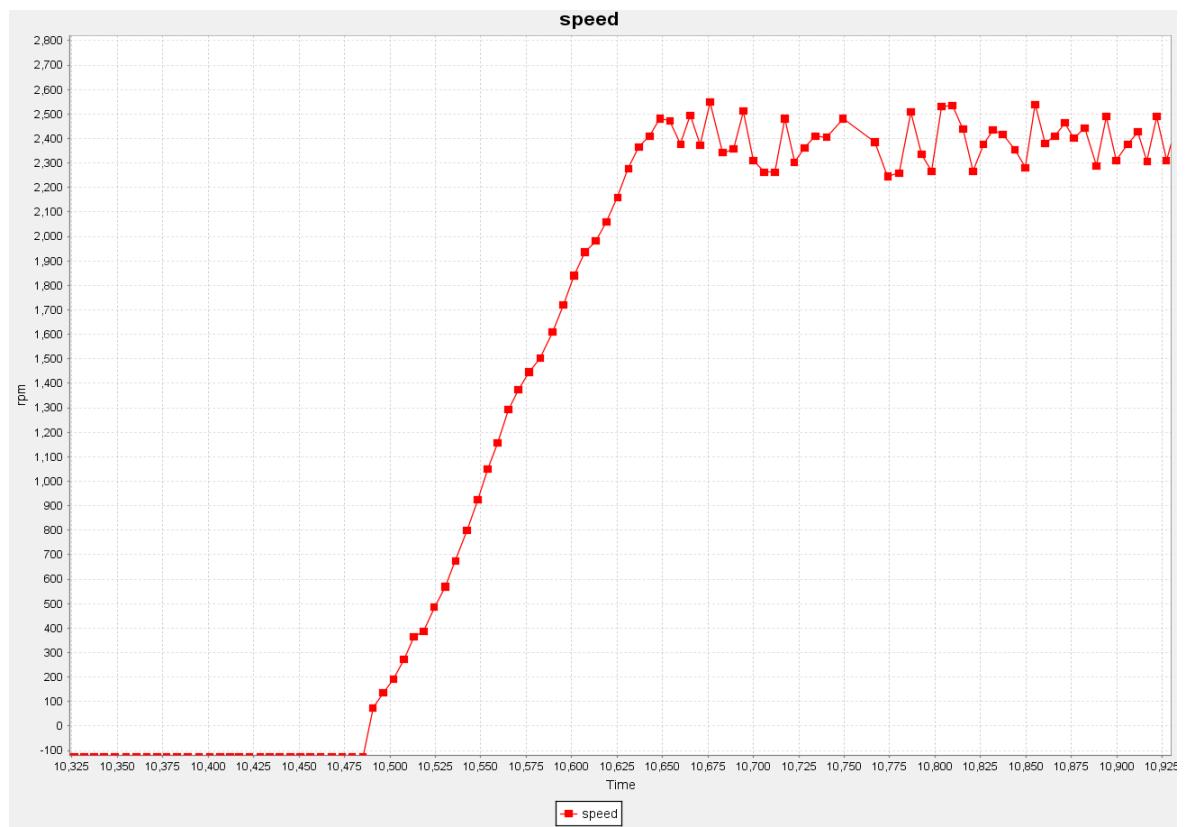
Rysunek 34: Przebieg prędkości obrotowej dla wartości zadanej 2500 obr/min i prądu 0.3A



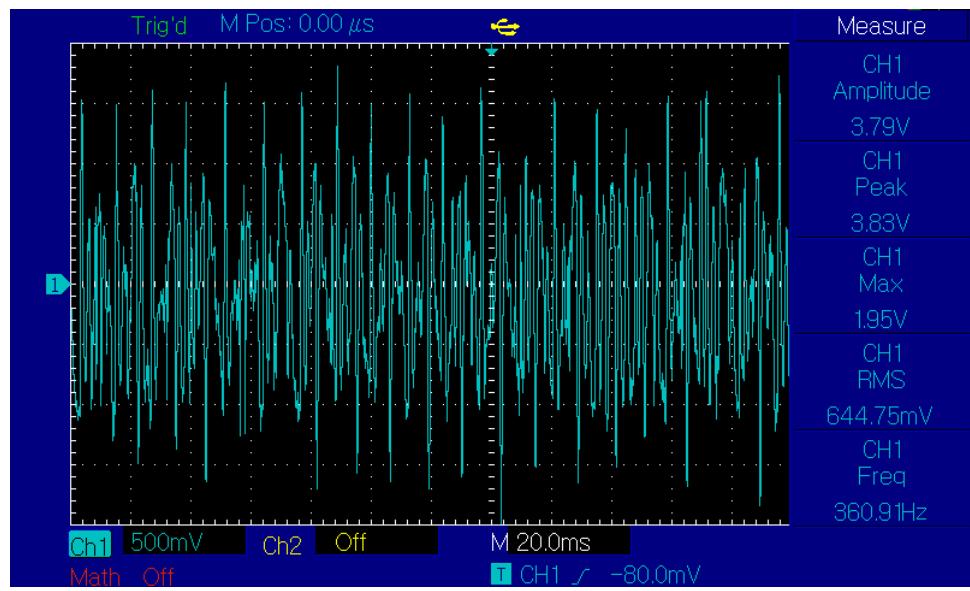
Rysunek 35: Przebieg prądu dla wartości zadanej amplitudy 0.5A bez obciążenia zewnętrzny momentem oporowym



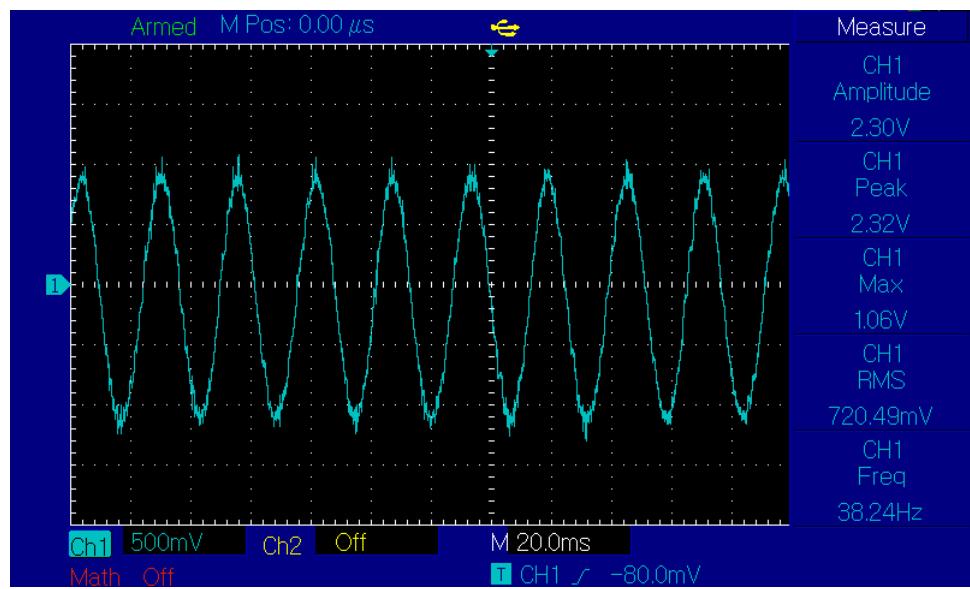
Rysunek 36: Przebieg prądu dla wartości zadanej amplitudы 0.5A z obciążeniem zewnętrznym



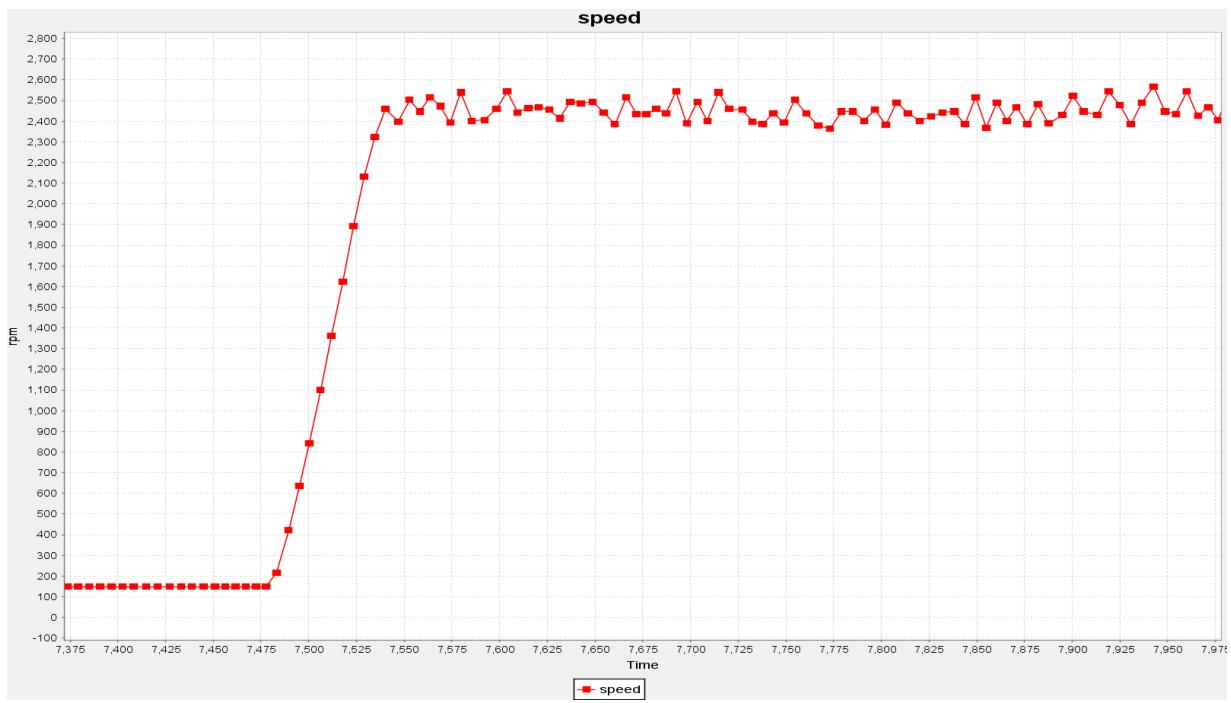
Rysunek 37: Przebieg prędkości obrotowej dla wartości zadanej 2500 obr/min i prądu 0.5A



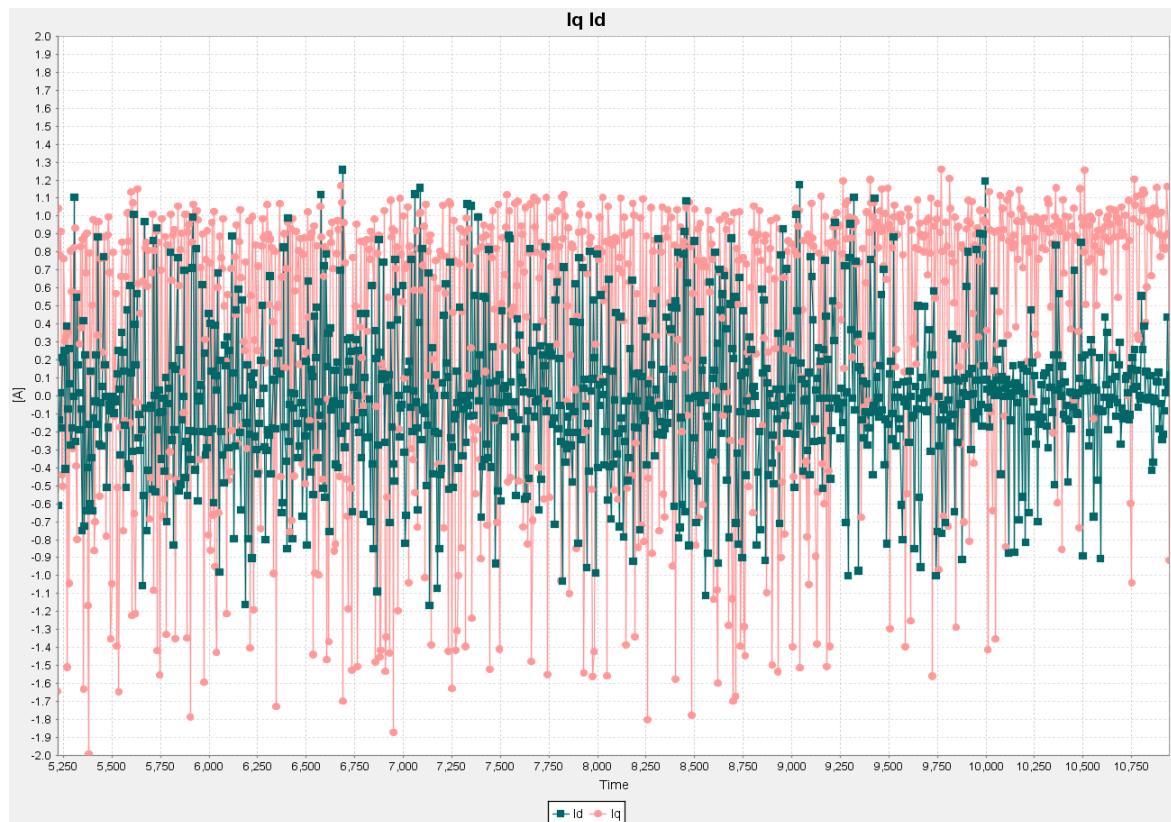
Rysunek 38: Przebieg prądu dla wartości zadanej amplitudy 1A bez obciążenia zewnętrzny momentem oporowym



Rysunek 39: Przebieg prądu dla wartości zadanej amplitudy 1A z obciążeniem zewnętrzny



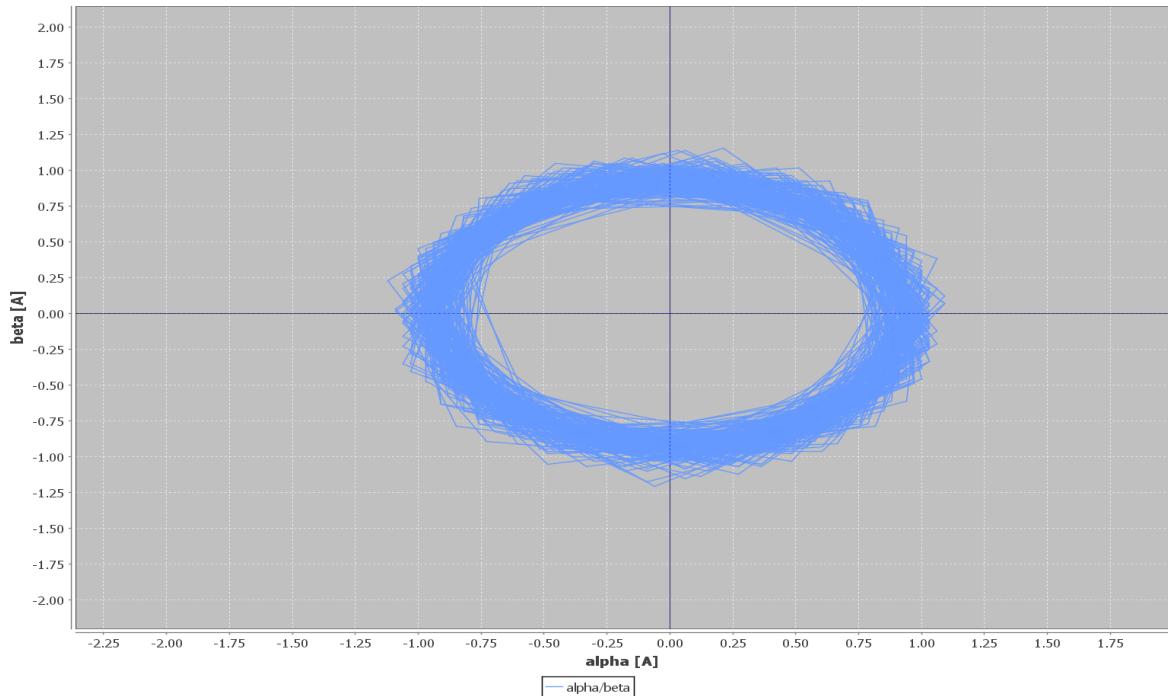
Rysunek 40: Przebieg prędkości obrotowej dla wartości zadanej 2500 obr/min i prądu 1A



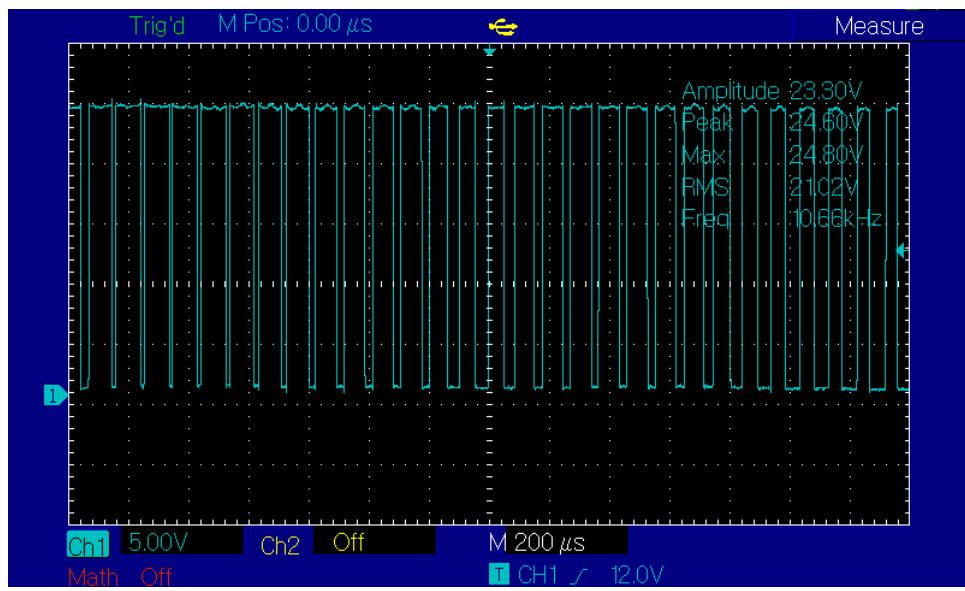
Rysunek 41: Przebieg składowej Id, Iq dla wartości zadanej amplitudy prądu 1A bez obciążenia zewnętrznym momentem oporowym



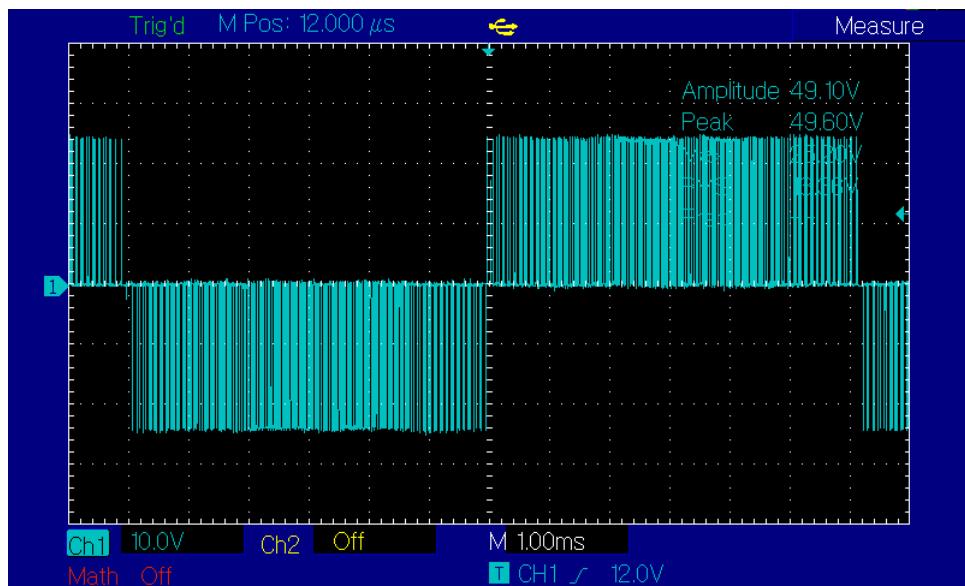
Rysunek 42: Przebieg składowej Id, Iq dla wartości zadanej amplitudy prądu 1A z obciążeniem zewnętrznym momentem oporowym



Rysunek 43: Hodograf prądu w układzie współrzędnych alfa, beta dla wartości zadanej 1A



Rysunek 44: Przebieg napięcia fazowego



Rysunek 45: Przebieg napięcia międzyfazowego

Z pomiarów wynika, że regulacja prądu przy ustalonej prędkości obrotowej i przy braku obciążenia zewnętrznym momentem oporowym, prąd ma przebieg zniekształcony i nieprzypominający sinusoidy. Z wykresu składowych prądu I_q i strumienia magnetycznego I_d , pokazanego na rysunku 41 widać, że przebiegi są przemienne, co oznacza, że nie jest utrzymywany stały kąt między polem magnetycznym wirnika i stojana. Moment elektromagnetyczny jest iloczynem strumienia magnetycznego, prądu i kosinusa kąta między polem magnetycznym stojana i wektorem siły przeciwelektromotorycznej. Przy ustalonej prędkości obrotowej i braku obciążenia generowany jest niski moment napędowy, który nie jest w stanie utrzymać kąta 90 stopni między polami magnetycznymi. Układ regulacji mimo wszystko próbuje uzyskać określona wartość zadawaną generując oscylacje.

Zwiększąc moment oporowy przebieg prądu przyjmuje kształt sinusoidy o amplitudzie wartości zadanej. Kąt między polem magnetycznym wirnika i stojana ustala się na poziomie bliskim 90 stopni. W takim przypadku moment elektrodynamiczny jest równy iloczynowi prądu I_q i strumienia magnetycznego. Regulator prędkości steruje prądem I_q aby utrzymać prędkość obrotowa niezależnie od momentu

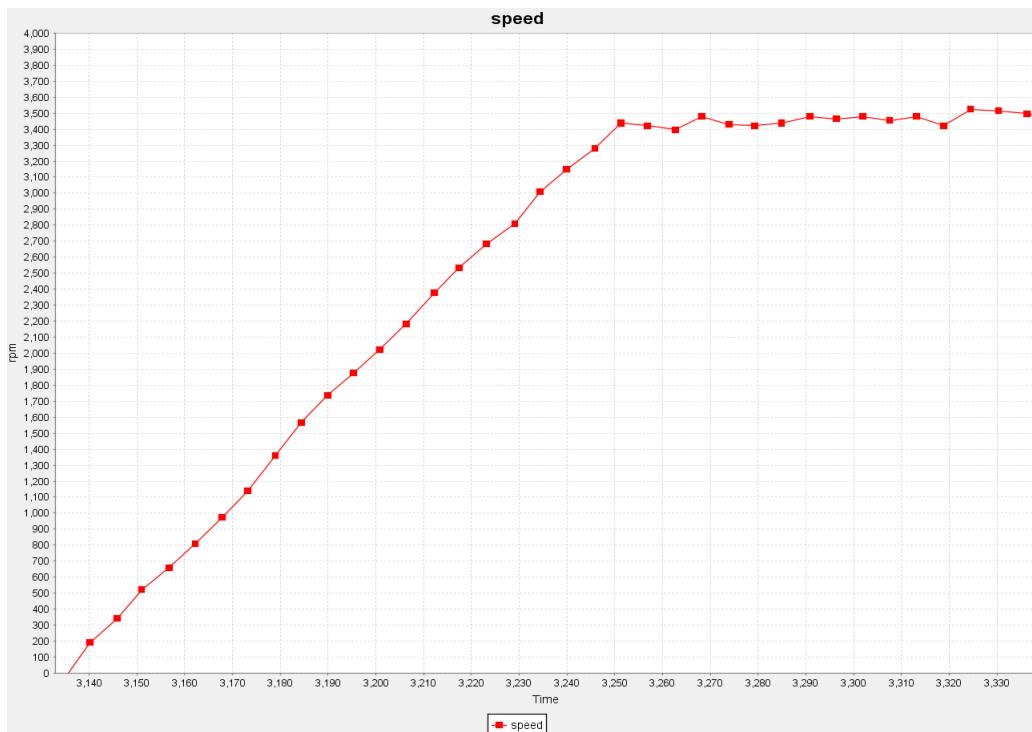
oporowego. Wartość zadana prądu I_q ustala maksymalny moment przy którym nie ma spadku prędkości obrotowej.

Z przebiegów prędkości obrotowej dla wartości zadanej 2500 obr/min dla nastaw wartości prądu 0.3A, 0.5A, 1A, które zostały przedstawione na rysunkach 34, 37 i 40 wynika, że wartość zadana prądu I_q bezpośrednio wpływa na moment obrotowy i dynamikę układu. Wraz ze wzrostem wartości prądu I_q wzrasta przyspieszenie silnika. Czas potrzebny do osiągnięcia ustalonej prędkości zadanej 2500 obr/min wynosi kolejno:

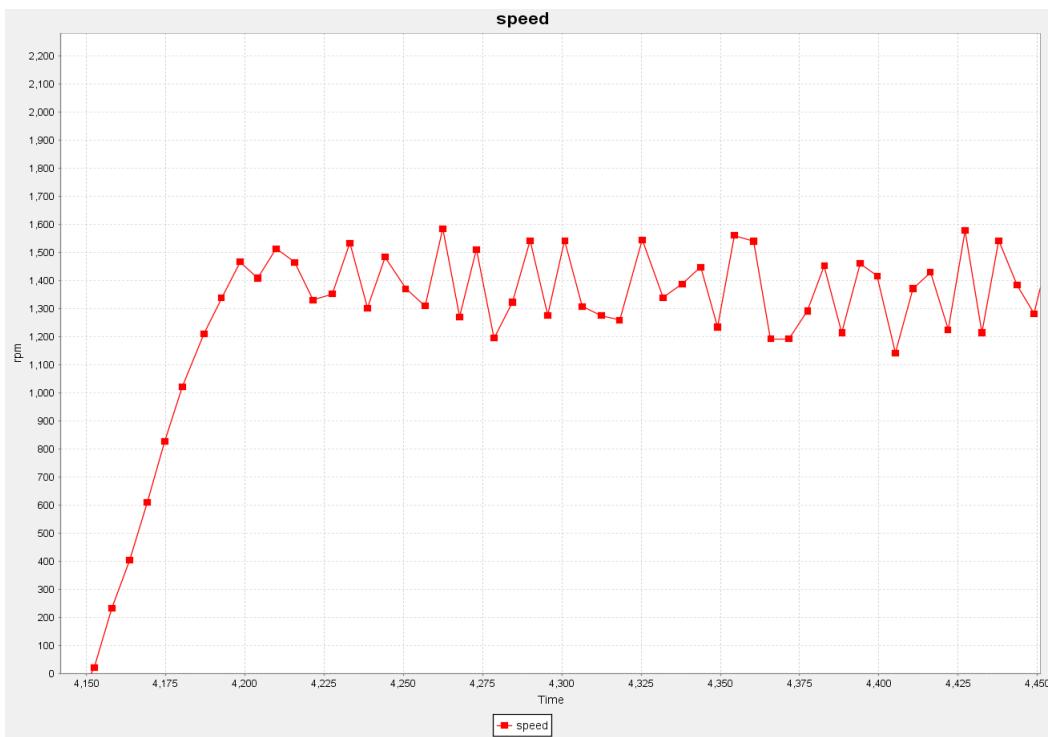
- 0.3 A - 0.5 sekundy,
- 0.5 A - 0.22 sekundy,
- 1 A - 0.09 sekundy.

6.2.2 Regulacja prędkości obrotowej

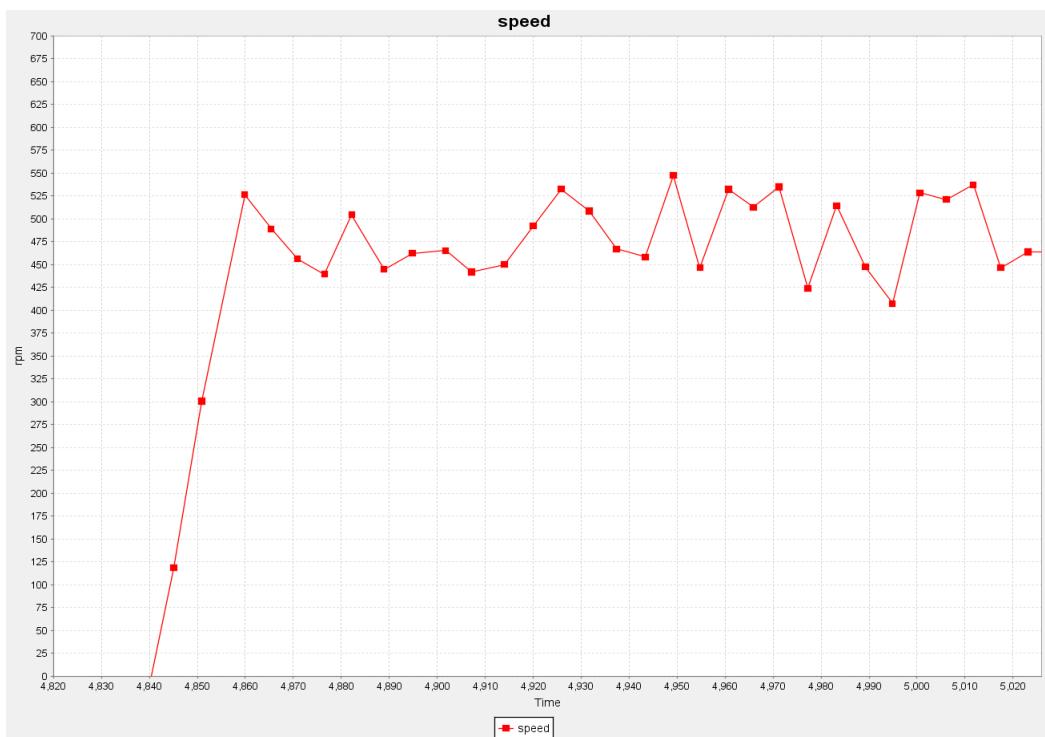
Pomiary przebiegów prędkości obrotowej zostały dokonane dla prędkości obrotowych 3500, 1500, 500, 200 obr/min przy amplitudzie prądu ograniczonej do 0.8A. Przebiegi przedstawione na rysunkach 46, 47, 48 i 49.



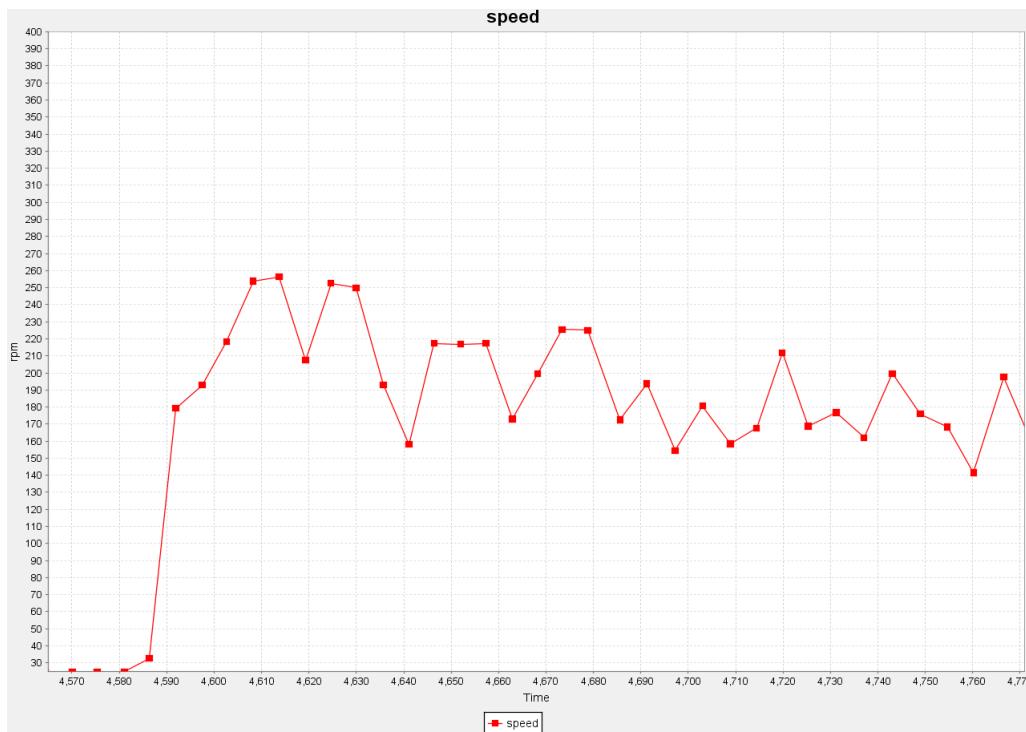
Rysunek 46: Przebieg prędkości obrotowej o wartości zadanej 3500 obr/min



Rysunek 47: Przebieg prędkości obrotowej o wartości zadanej 1500 obr/min



Rysunek 48: Przebieg prędkości obrotowej o wartości zadanej 500 obr/min

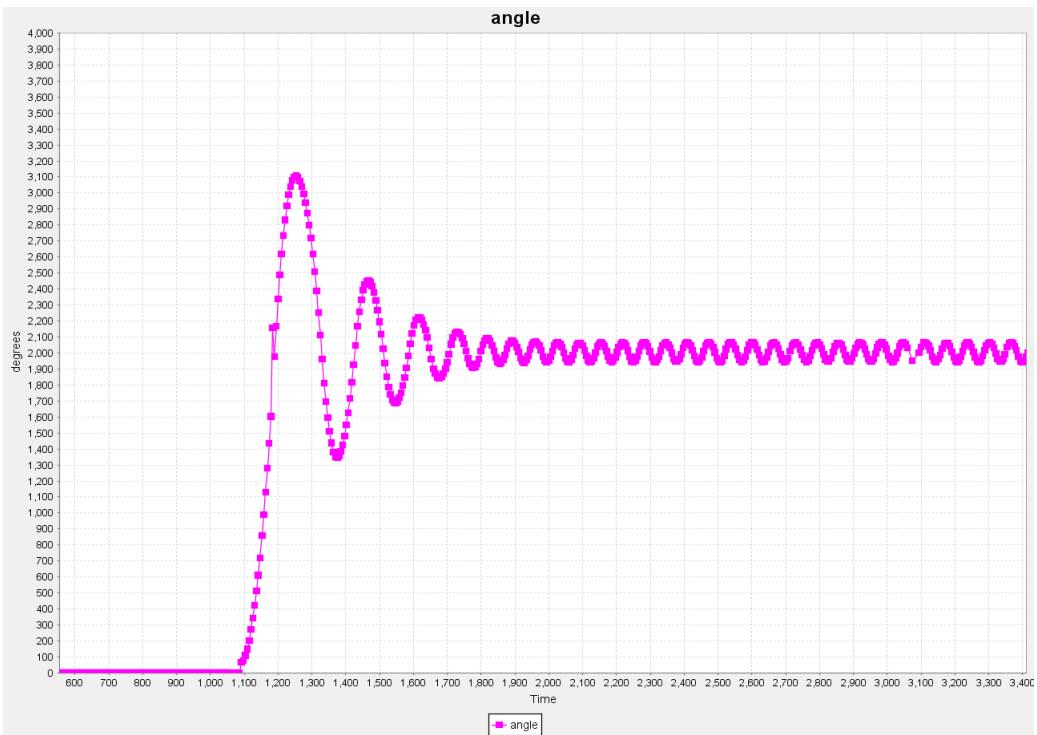


Rysunek 49: Przebieg prędkości obrotowej o wartości zadanej 200 obr/min

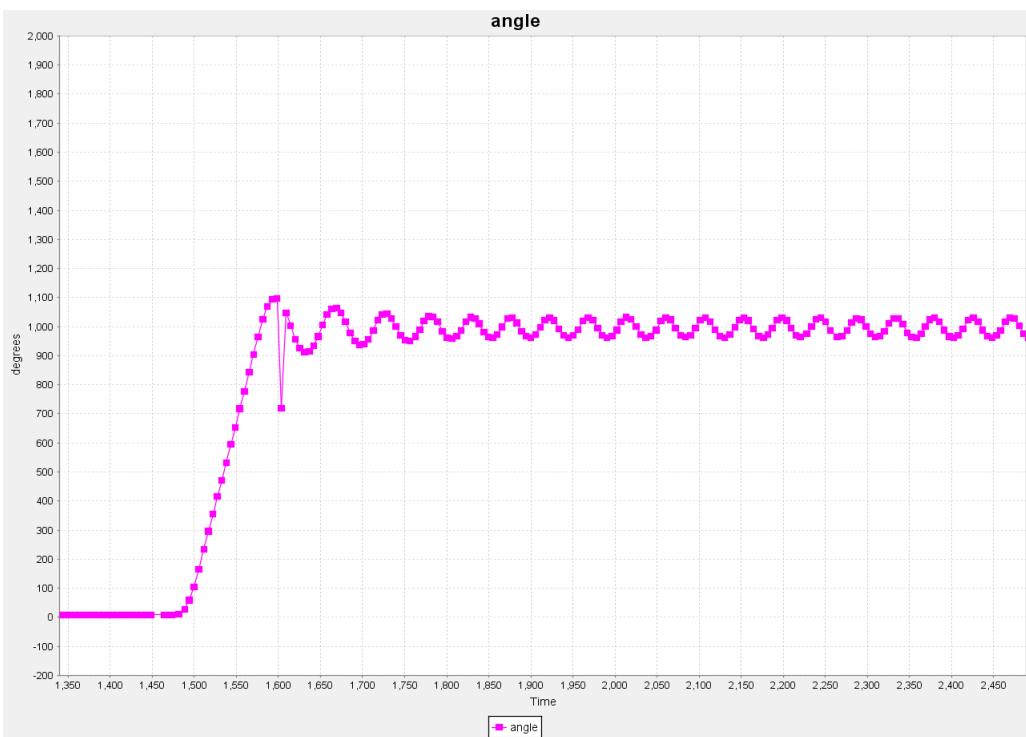
Z pomiarów wynika, że przy zastosowaniu układu napędowego ze sterowaniem wektorowym dla silnika trójfazowego uzyskujemy płynną regulację prędkości obrotowej w całym zakresie.

6.2.3 Regulacja położenia wirnika

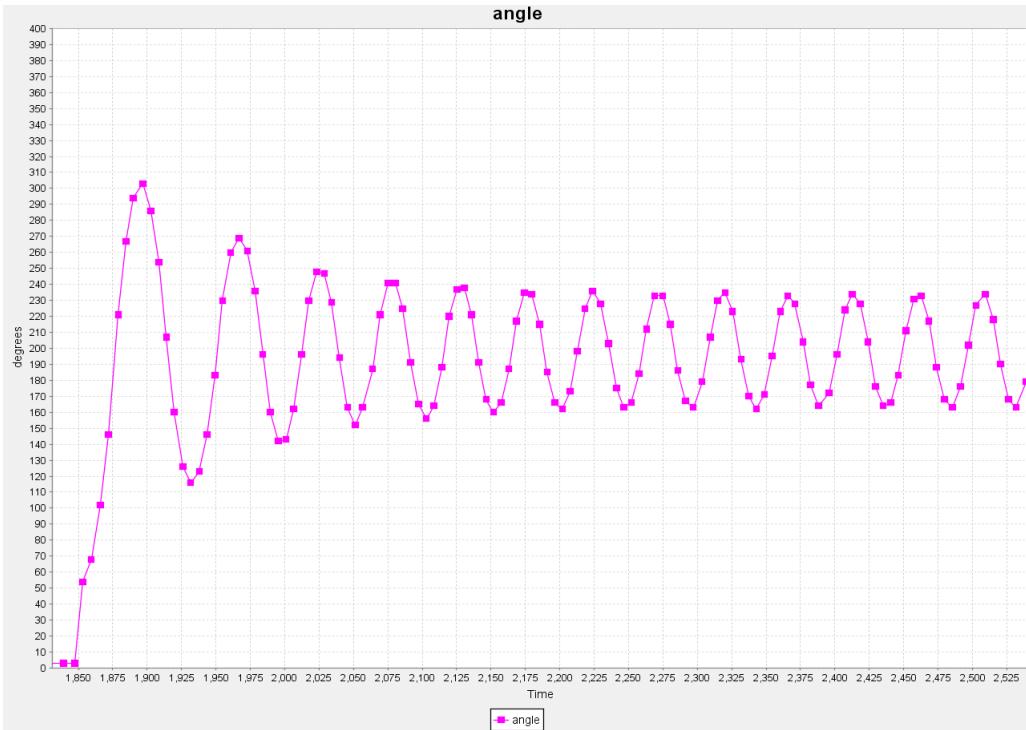
W celu sprawdzenia jakości działania regulatora położenia wału silnika pomiary wykonano dla wartości zadanych 2000,1000 i 200 stopni. Przebiegi przedstawione na rysunkach 50, 51 i 52. Rozdzielczość przetwornika obrotowo-impulsowego wynosi 720 impulsów/obr.



Rysunek 50: Przebieg położenia wału silnika dla wartości zadanej 2000 stopni



Rysunek 51: Przebieg położenia wału silnika dla wartości zadanej 1000 stopni



Rysunek 52: Przebieg położenia wału silnika dla wartości zadanej 200 stopni

Podczas montażu nie udało się osiągnąć dokładnej liniowości wału silnika i enkodera. Z tego powodu wykorzystano sprzęgło sprężynowe, które powoduje, że silnik jest obciążony dodatkowym momentem oporowym o niestałej wartości w zakresie jednego obrotu. Zjawisko to wpływa oscylacje położenia wirnika względem wartości zadanej.

7 Podsumowanie

Do sprzętowej konfiguracji napędu wykorzystano zestaw rozwojowy STM32 B-G431-ESC1 wyposażony w rdzeń ARM Cortex G4, przetwornik energoelektroniczny i w obwody pomiaru prądu. Jest układem dedykowanym do budowania napędów silnikowych. Za komunikację między sterownikiem a interfejsem użytkownika jest odpowiedzialny zestaw rozwojowy STM32 Nucleo F746ZG, który posiada wbudowany interfejs Ethernet za pomocą którego łączy się z aplikacją użytkownika. Ze sterownikiem jest połączony dzięki wykorzystaniu łącza szeregowego UART. Interfejs użytkownika został zaprogramowany w języku Python za pomocą oprogramowania Spyder w środowisku Anaconda. Umożliwia wprowadzanie wartości zadanych prądu, prędkości, położenia wirnika i nastaw dla regulatorów PID. Wyświetla aktualne parametry silnika w czasie rzeczywistym w postaci tekstowej i wykresów. Układ napędowy został zaimplementowany na płycie STM32 B-G431-ESC1 w postaci sterowania wektorowego FOC i modulacji wektorowej SVPWM. Do stworzenia oprogramowania napędu wykorzystano środowisko programistyczne STM32CubeIDE. Za pomocą oscyloskopu UNI-T UTD2102CEX i aplikacji STMStudio, która umożliwia podgląd do zmiennych w czasie rzeczywistym, dokonano pomiarów przebiegów prądu, prędkości obrotowej, położenia wirnika dla różnych wartości zadanych przy określonych nastawach regulatorów PID dobranych doświadczalnie.

Zgodnie z wynikami przebiegów przedstawionymi w podrozdziale 6.2.1, zaprojektowany napęd silnika BLDC reguluje wartością prądu, który jest odpowiedzialny za moment elektromagnetyczny i dynamikę układu. Regulacja odbywa się w sposób prawidłowy, ponieważ dla przyłożonego momentu oporowego, przebieg prądu przyjmuje kształt sinusoidy o amplitudzie wartości zadanej prądu I_q . Im większa wartość zadana prądu tym czas na rozpoczęcie się wirnika do ustalonej prędkości obrotowej jest krótszy. Jednym z założeń układu napędowego ze sterowaniem wektorowym stanowi możliwość płynnej regulacji prędkości w całym zakresie. Podrozdział 6.2.2, gdzie pokazano przebieg prędkości obrotowej wirnika dla różnych wartości zadanych, udowodnia że cel ten został zrealizowany. Oscylacje na wykresach prędkości i położenia wynikają z faktu, że nie udało się dokładnie wycentrować wirnika z wałem enkodera. Z tego powodu

wykorzystano sprzęgło sprężynowe, które powoduje, że silnik jest obciążony dodatkowym momentem oporowym o niestalej wartości w zakresie jednego obrotu.

Dużym wyzwaniem okazało się zaprojektowanie bezawaryjnego i efektywnego sposobu komunikacji między sterownikiem B-G431-ESC1, płytą Nucleo F746Zg i aplikacją użytkownika. Po wielu próbach i konfiguracjach najbardziej niezawodny okazał się sposób, w którym sterownik B-G431-ESC1 za pomocą portu szeregowego otrzymuje tylko informacje o ustawieniach i parametrach napędu, natomiast do zestawu Nucleo F746ZG podłączony zostaje przetwornik obrotowo-impulsowy i czujniki prądu. Dzięki temu rozwiązaniu układy pracują w sposób bardziej niezależny, mniej awaryjny i w maksymalny sposób ograniczone wykorzystanie portu szeregowego, który powodował najwięcej problemów z uzyskaniem synchronizacji między elementami układu i aplikacją użytkownika.

W przyszłości, do zaprojektowanego układu napędowego można dodać serwer WWW i napisać dodatkową aplikację dla urządzeń mobilnych. Interfejs użytkownika, który jest napisany w języku Python można rozszerzyć o procedurę automatycznego sterowania napędu silnika.

Spis rysunków

1	Przebiegi siły elektromotorycznej E, prądu I i momentu T dla jednej fazy dla: a) silnika PMSM, b) silnika BLDC [15]	4
2	Stojan silnika BLDC [6]	4
3	Wirnik silnika BLDC [6]	4
4	Uproszczony schemat układu sterowania i zasilania silnika [3]	5
5	Schemat silnika synchronicznego z magnesami trwałymi	6
6	Algorytm FOC [5]	8
7	Transformata Clark'a [4]	8
8	Transformata Park'a [4]	9
9	Transformacje w algorytmie FOC [5]	9
10	Wektory w przestrzeni stanu i podział na sektory [5]	10
11	Maksymalna długość wektora [5]	10
12	Generowanie wektora V_x [9]	11
13	Sekwencja generowania wektora referencyjnego	11
14	Napęd z silnikiem BLDC	12
15	Silnik BLDC	14
16	Zestaw rozwojowy STM32 B-G431-ESC1 [14]	15
17	Zestaw rozwojowy STM32 Nucleo F746ZG [12]	16
18	Konfiguracja mikrokontrolera STM32G431	17
19	Konfiguracja zegara i częstotliwości taktowania peryferii STM32G431	18
20	Konfiguracja mikrokontrolera STM32F746ZG	19
21	Konfiguracja zegara i częstotliwości taktowania peryferii STM32F746ZG	20
22	Pomiar prądu [13]	26
23	Wzmacnianie mierzonego sygnału [13]	26
24	Schemat regulatora PID biblioteki CMSIS-DSP [1]	27
25	Synchronizacja sygnału PWM z wyzwalaniem przerwania od przetwornika ADC	28
26	Model napędu silnika BLDC	29
27	Schemat układu komunikacji	29
28	Interfejs użytkownika	30
29	Interfejs użytkownika	31
30	Przebiegi generowanych sygnałów za pomocą SVPWM na podstawie trzech wejściowych sygnałów sinusoidalnych	32
31	Nastawy regulatorów PID	32
32	Przebieg prądu dla wartości zadanej amplitudy 0.3A bez obciążenia zewnętrznym momentem oporowym	33
33	Przebieg prądu dla wartości zadanej amplitudy 0.3A z obciążeniem zewnętrznym	33
34	Przebieg prędkości obrotowej dla wartości zadanej 2500 obr/min i prądu 0.3A	34
35	Przebieg prądu dla wartości zadanej amplitudy 0.5A bez obciążenia zewnętrznym momentem oporowym	34
36	Przebieg prądu dla wartości zadanej amplitudy 0.5A z obciążeniem zewnętrznym	35
37	Przebieg prędkości obrotowej dla wartości zadanej 2500 obr/min i prądu 0.5A	35

38	Przebieg prądu dla wartości zadanej amplitudy 1A bez obciążenia zewnętrznym momentem oporowym	36
39	Przebieg prądu dla wartości zadanej amplitudy 1A z obciążeniem zewnętrznym	36
40	Przebieg prędkości obrotowej dla wartości zadanej 2500 obr/min i prądu 1A	37
41	Przebieg składowej I_d , I_q dla wartości zadanej amplitudy prądu 1A bez obciążenia zewnętrznym momentem oporowym	37
42	Przebieg składowej I_d , I_q dla wartości zadanej amplitudy prądu 1A z obciążeniem zewnętrznym momentem oporowym	38
43	Hodograf prądu w układzie współrzędnych alfa, beta dla wartości zadanej 1A	38
44	Przebieg napięcia fazowego	39
45	Przebieg napięcia międzyfazowego	39
46	Przebieg prędkości obrotowej o wartości zadanej 3500 obr/min	40
47	Przebieg prędkości obrotowej o wartości zadanej 1500 obr/min	41
48	Przebieg prędkości obrotowej o wartości zadanej 500 obr/min	41
49	Przebieg prędkości obrotowej o wartości zadanej 200 obr/min	42
50	Przebieg położenia wału silnika dla wartości zadanej 2000 stopni	43
51	Przebieg położenia wału silnika dla wartości zadanej 1000 stopni	43
52	Przebieg położenia wału silnika dla wartości zadanej 200 stopni	44

Literatura

- [1] CMSIS Version 5.7.0 Software Interface Standard for Arm Cortex-based Microcontrollers. https://www.keil.com/pack/doc/CMSIS/DSP/html/group__PID.html, 2008. [Online; accessed 01-07-2020].
- [2] Mokhtar Bouanane, Mohamed Bourahla, and Madjid Guerouad. Design and implementation of three-phase ipm inverter based on svpwm for ac motor applications using dspic30f4011. *Przegląd Elektrotechniczny*, 95, 2019.
- [3] Arkadiusz Domoracki and Krzysztof Krykowski. Silniki bldc-klasyczne metody sterowania. *Zeszyty Problemowe-Maszyny Elektryczne*, 72, 2005.
- [4] Texas Instruments Europe. Field orientated control of 3-phase ac-motors. *Literature Number: BPRA073*, 1998.
- [5] Charles Frick. Brushless dc motors introduction for next-generation missile actuation systems outline. *Analog Devices, Inc.*, 6, 2018.
- [6] Zbigniew Goryca. Metody sterowania silników bldc. *Prace Naukowe Instytutu Maszyn, Napędów i Pomiarów Elektrycznych Politechniki Wrocławskiej*, (66):32–47, 2012.
- [7] Krzysztof Krykowski. *Silniki PM BLDC*. Wydawnictwo BTC, Legionowo, 2015.
- [8] Dominik Łuczak and Krzysztof Siembab. Comparison of fault tolerant control algorithm using space vector modulation of pmsm drive. In *Proceedings of the 16th international conference on mechatronics-mechatronika 2014*, pages 24–31. IEEE, 2014.
- [9] Ahmed A Mansour. Novel svpwm based on first order equation. *Journal of Electrical Systems and Information Technology*, 2(2):197–206, 2015.
- [10] Jacek Przepiókowski. Trojfazowy falownik wektorowy zestaw ewaluacyjny tms320f28035 podstawy teoretyczne. *SKLEP.AVT.PL ELEKTRONIKA PRAKTYCZNA*, 4, 2010.
- [11] STMicroelectronics. *STM32F745xx, STM32F746xx*, 2016.
- [12] STMicroelectronics. *UM1974, User manual, STM32 Nucleo-144 boards*, 2017.
- [13] STMicroelectronics. *AN5397, Amplification network schematic for bipolar current sensing*, 2019.
- [14] STMicroelectronics. *UM2516 User Manual, Electronic speed controller Discovery kit for drones with STM32G431CB*, 2020.
- [15] Konrad Zajkowski and Stanisław Duer. Sterowanie prędkością obrotową silnika bezszczotkowego bldc. *Autobusy: technika, eksploatacja, systemy transportowe*, 14:303–306, 2013.