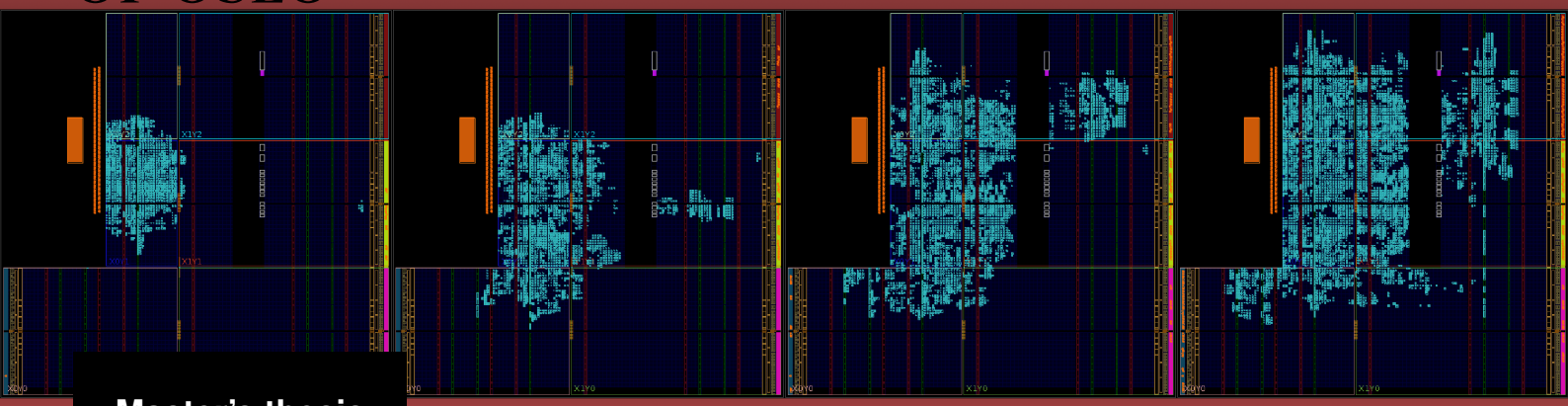
UNIVERSITY
OF OSLO



Master's thesis

Scalable Multi-Axis FOC on FPGA for Permanent Magnet Synchronous Motors

A Pipelined Approach

Oscar Hæstad Bjørnstad

Robotics and Intelligent Systems

60 ECTS study points

Department of Informatics

Faculty of Mathematics and Natural sciences

Spring 2024

Oscar Hæstad Bjørnstad

Scalable Multi-Axis FOC on FPGA for Permanent Magnet Synchronous Motors

A Pipelined Approach

Supervisor:

Yngve Hafting

Abstract

Brushless Direct Current (BLDC) motors are widely utilized in robotics due to their lower maintenance needs and higher efficiency compared to brushed motors [7]. However, BLDC motors need a controller to operate. Field-oriented control is one way to control such a motor. This thesis explores the simultaneous control of multiple BLDC motors on a single FPGA through pipelining the FOC logic. The proposed FOC-core provides a platform for further motor control development with easy-to-replace modules used within. The FOC core contains pipelined logic for Clarke, Park, inverse Park - transformations, and the use of a pipelined CORDIC algorithm to enable possibilities for more advanced PWM techniques. The PWM generation and PI control are done through non-pipelined logic. Surrounding this Core, an implementation of multi-axis FOC has been created with 12 sets of motor configurations, where the corresponding resource usage has been reported. The proposed core and the outcome of the utilization reports have shown that sharing logic through pipelining in FOC is possible and can be beneficial when controlling multi-axis robots.

Contents

List of Figures

List of Tables

Preface

Acknowledgements

I am deeply grateful for the enriching journey that this thesis represents. My studies and research work during my time at the University have been pivotal in fostering my personal and academic growth.

This accomplishment, together with my postgraduate achievements, is indebted to the unwavering guidance and support of my supervisor, Yngve Hafting.

I also extend my heartfelt gratitude to my fellow students for their kind gestures and expertise, which have been integral to my development and journey at the University of Oslo. Their companionship and support are treasures I will forever cherish.

The use of AI services when rephrasing sentences

To enhance the language quality of select sentences in this thesis, the GPT-3-based[9] UIO language model[27] has been utilized. The language model, also known as GPT UiO, was developed by the UiO IT department. The AI language model rephrased certain sentences while ensuring that no new information was introduced during this process. The rephrased sentences were used for inspiration or "as is" during the rephrasing of the previously mentioned sentences.

Chapter 1

Introduction

This chapter is an introductory chapter for this thesis. Section 1.1 will include the motivation for researching this topic. The following section 1.2 will include our 3 research questions 1.2.1, 1.2.2 and 1.2.3. Thereafter, section 1.3 is preserved for presenting some of the main contributions. Towards the conclusion of this section, in 1.4, an outline of this thesis will be provided.

1.1 Motivation

Robotics and robotic manipulators have seen wide adoption in tasks that were previously performed by humans. One such example is the quadrupedal walked Spot [19] by Boston Dynamics that can be seen performing autonomous inspections in industrial facilities [18]. To control such a system, control on multiple levels is required. The lowest level of control of such a manipulator is the control of currents passing through the electrical motors. In the Open Dynamics robot (ODR)[28] a set of brushless direct current (BLDC) motors is used for the movement of the robot. The motors are powered by drivers which again are controlled by a system optimizing the flow of current. Such a system can include Trapezoidal Control, Sinusoidal Drive control, Simplified Vector control, and Field Oriented Control (FOC)[20]. When researching robotics and electric motor fundamentals one might want to expand those underlying controllers. This could be an expansion in the complexity of controllers or an expansion in the number of motors controlled by a single chip. Controlling multiple motors through a single chip might reduce the hardware complexity as fewer controllers are needed. In both cases, it is important to know the insides of the selected motor-control system.

Utilizing Field Programmable Gate Arrays (FPGAs) a researcher can update and test multiple motor controllers on the same hardware. Having an open core for multi-axis control will also expand the possibilities for said researcher. By utilizing an open core, future research can expand upon said core with experimental code and newer algorithms. Said systems can enable research and implementations of "state of the art" algorithms for use in motor control. One of the methods enabling an FPGA to control multiple motors simultaneously is multiplying a single core until it matches the number of motors. Such a method can increase the resource consumption within the chip. One way computers have been able to utilize resources more effectively has been through the pipelining of instructions [34]. One approach to reduce resource consumption in an FOC system is the introduction of pipelining within multiple components in the FOC core. Studying and analyzing the effect of pipelining FOC on FPGA is therefore a goal of this thesis. The motivation is fueled by the desire to utilize fewer resources while providing a platform for future implementations of more complex functionalities.

Another justification for this thesis is found when looking at the computational power of FPGAs. FPGAs have often been used for hardware acceleration and solving computable problems [24]. Therefore a natural expansion of a robotics system is to utilize the remaining FPGA fabric for non-motor control applications. To allow such expansions an optimization in terms of resource consumption is desirable.

1.2 Research questions

This thesis aims to provide a platform for future investigation into FOC on FPGA while optimizing the resource consumption within the FOC-Core. Therefore, a few questions come to mind.

1.2.1 Research question 1

First and foremost, it is important to note how such a system scales when applied to various motor configurations. Hence, measuring resource consumption within the FOC-core is critical. Our first research question is at this moment formulated as follows:

> When scaling an FOC-core through pipelining, how is the resource consumption affected by the addition of multiple motors?

1.2.2 Research question 2

Furthermore, such a pipelined system might introduce a higher initial cost. Therefore, it is important to find out when it is beneficial to share the logic between motors and synchronize the input and outputs of the core. Research question number two is therefore formulated as follows:

> When is the proposed architecture capable of outperforming an n-number duplication of a single core in terms of resource usage on FPGA?

1.2.3 Research question 3

Moving over to our third and last research question in this thesis. For every system, there will always be something that limits the design. In this thesis, one of the goals is to push the constraints as far as possible. However, to move the limiting factors, one must know what limits the system and how this constrains the whole system. The aforementioned third research question is hence formulated as follows:

> What are the limiting factors for FOC on multiple motors with FPGA?

1.3 Main contributions

While researching the aforementioned research questions in 1.2, this thesis aims to provide a set of new contributions in the field of motor control. Said contributions boil down to:

- The creation of a modular and customizable FOC-core for simultaneous control of multiple BLDC motors.
- Gathering and presenting data regarding resource consumption of LUTs, Registers, and DSPs in a pipelined FOC-core.
- Optimization through pipelining of FOC-core so the FPGA resources can be utilized on precision models or the addition of functionalities beyond FOC.

1.4 Thesis outline

This section will present an overview of the contents and structure of the thesis. The chapters will have a short description following the name and numbering of each chapter. There is an appendix A trailing the conclusion 6. This thesis is structured as follows:

- **Chapter 1 - Introduction:** This chapter presents the motivation behind the thesis and provides the research questions and contributions of this thesis.
- **Chapter 2 - Background & Related Works:** This chapter briefly explains the technology and theory used during this thesis. Related works and the landscape of the motor control research field are also visited.
- **Chapter 3 - Implementation:** In this chapter, the physical hardware that the core was designed for is described. A deep dive into the implementation of a FOC-core with its outer layers and the software supporting it is also given.
- **Chapter 4 - Experiments & results:** An overview of the experiments and the following results are presented in this chapter.
- **Chapter 5 - Discussion:** This chapter exists to discuss and analyze the results obtained through the experiments.
- **Chapter 6 - Conclusion:** This chapter contains a quick summary of the thesis. The answering of the research questions and the listing of main contributions, as well as future work, are conducted in this chapter.
- **Appendix A - PWM state figures:** This chapter contains extra figures and illustrations not shown during the other chapters.

The code developed for the FOC-Core has been uploaded to the designated GitHub repository [4].

Chapter 2

Background & Related Works

The purpose of this chapter and the sections contained within is to give the reader an overview of the theory behind the implemented system. Section 2.1, 2.2, and 2.3 contains general information about tools and the importance of open-source solutions. The following sections 2.4 - 2.11 contains background information about Field Oriented Control (FOC) and the modules used within FOC. The section 2.12 contains information on motor control on FPGA and related works.

2.1 FPGA

A field-programmable gate array (FPGA) is a multi-purpose general integrated circuit. The design contained within the FPGA may be configured by the user multiple times as the technology is re-programmable. The functionality of the design is achieved through the connections and configurations of logic blocks within the FPGA. To program/configure the FPGA a block design or a hardware description language (HDL) such as VHDL or Verilog is synthesized into a bit-stream that may be uploaded to the FPGA. FPGAs are used in experimentation, low latency applications, and research as they run on a hardware level with a deterministic timing and are re-configurable.

2.2 Vivado

Vivado [55] is a program created by Xilinx for use with their Xilinx devices lineup. Xilinx is used to compile HLS designs, create block designs, and constrain-, Synthesize-, Implement-, and program said designs onto an FPGA or System on Chip (SoC)-FPGA. It supports HDL languages like VHDL, SystemVerilog, and Verilog. The design can also be created as a block design where it is possible to use IPs from a variety of libraries. To use the processor on a Xilinx product in conjunction with the FPGA fabric the block design has to include a block for the HDL code and a block for the Zynq Processing System.

For FPGA designs some sort of constrainment file is necessary. This constrainment file contains information about where the physical IO pins are connected, what frequency the design should be optimized to run at, and other project and FPGA chip-specific limitations that will need enforcement on the design routing [2]. Some other examples are constraints on path lengths, if a path crosses between clock domains, the timing between routing paths, the use of internal pull-up transistors and more advanced types. The constraints force Vivado to create appropriate connections within the interconnect to enable the device to work as intended.

When Vivado synthesizes a design, the RTL code and the block designs are combined into a netlist. A netlist describes the connectivity between and inside the modules of the design. During the implementation phase, the device resources are allocated and the netlist is placed and routed within the allocated device resources [1]. When routing and placing the design, the constraints for the design have to be taken into consideration. Vivado starts by optimizing the design to make it easier to place the nets and reduce timing[1]. Secondly it places the nets onto the device. Thirdly the design might get post-place optimized to reduce timing and fanout. The next step for the implementation is to route the design onto the targeted device. The last step during the implementation is to optimize the design even more if needed.

Post implementation it is possible to generate a bitstream. This process produces a file that can be loaded onto the targeted device so it can be used in the real world. For projects that include a processing system, the bitfile is needed for use in programs like Vitis [54].

[1]Timing refers to the time an operation takes in the physical design.

2.3 Black/White - Box

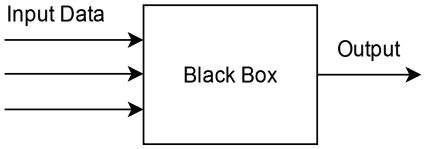
When a digital or software design is created, the system designer often has to rely on some premade packages or intellectual properties (IPs). The package and IP relied upon in a project might vary from full transparency to minimized transparency, where only the function's input and output are known. The Black-box and White-box phrases can be borrowed from the field of software testing. In this field, white-box testing considers the system's internal structure, and it is known to the tester [35].

A White-Box system consists of code that is fully transparent about its inner workings. The users of such packages and designs can see what happens inside the box. In a white box system, the system user can modify the module's inner workings due to its open nature. During the research, a white/open -box solution might be beneficial as it is possible to exchange the inner workings with other desired modules and modify Software/HDL for testing purposes.

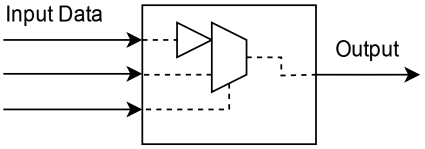
Companies that create IPs often have black box solutions, as it is easier to control the access to the IP and who gets to use it. Having black-box solutions makes it harder for individuals to directly copy code on the internet and use it in non-authorised settings.

"An intellectual property core (IP core) is a functional block of logic or data used to make a field-programmable gate array (FPGA) or application-specific integrated circuit for a product" [59]. An IP in FPGA technology might be a core for motor control. The transparency and configurability might vary from core to core.

For some applications, the ease of configuring and setting up a black box might be more efficient than finding and using an open-source offering. For other settings like research, open-source might be preferred as it is possible to change the inner workings of the core.



(a) A Black Box system is a closed system where the developer and user are not able to see the inner functions that happen behind the scenes. The user/developer only knows the input and output of the function/Box



(b) A White Box system is an open system where the system's inner workings are open and can be changed by the developer. The inner working of this system is known to the user and developer

Figure 2.1: The black box is closed since it is impossible to see what is inside. The white box is open since the inner workings and the inside is possible to view and modify

2.4 BLDC

A BLDC (Brushless DC) -motor is a type of synchronous electric motor used in various applications [7]. Some applications are Drones, fans, robotic manipulators, and radio-controlled cars. A BLDC motor consists of two major components: the stator and the rotor. The stator is the motor's housing. It contains multiple sets of electrical winding used as electromagnets to induce a magnetic field that works on the rotor.

The rotor is a free-spinning component that houses permanent magnets. The magnets spin due to changes in the electromagnetic field caused by the electromagnets in the stator. The direction and amplitude of the field are controlled by the current passing through the coils in the electromagnets. The induced field either attracts or repels the permanent magnets.

To create continuous movement for BLDC motors, one would have to run the electromagnets in different phases. A common type of BLDC motor is the 3-phase BLDC motor. In this type of motor, each electromagnet is connected so that every third electromagnet is in the same phase. The groups are, therefore, commonly referred to as phases.

To achieve a continuous and finite torque or speed control of the motor, the current induced in the coils would need to follow a graph like 2.2. To achieve maximum torque and efficiency, a voltage is applied to the electromagnets, so the magnetic field from the currents is 90 electrical degrees on the rotor. In short, we apply voltages to the phases so the magnetic fields either push or pull on the rotor. The current flow created by the voltages follows a sinusoidal wave with a $\frac{360}{3}^{\circ}$= 120-degree offset between each phase.

Two main configurations exist for a three-phase BLDC motor: Wye and Delta. The most common connection is the wye (Y) connection (fig 2.3a). In this configuration, the phases meet in the middle and form a common neutral point. When a voltage is applied to one coil, and a current is created, the two other coils will have a current flow that sums up to the negative of the first coil according to Kirchhoff's current law. A simple visualization of this would be to add up Phase 1 + Phase 2 + Phase 3 and see that the sum will always be 0.

The other main configuration is the Delta connection. Here, the coils are connected in series and form the structure of a triangle (fig 2.3b). The circuit that switches the

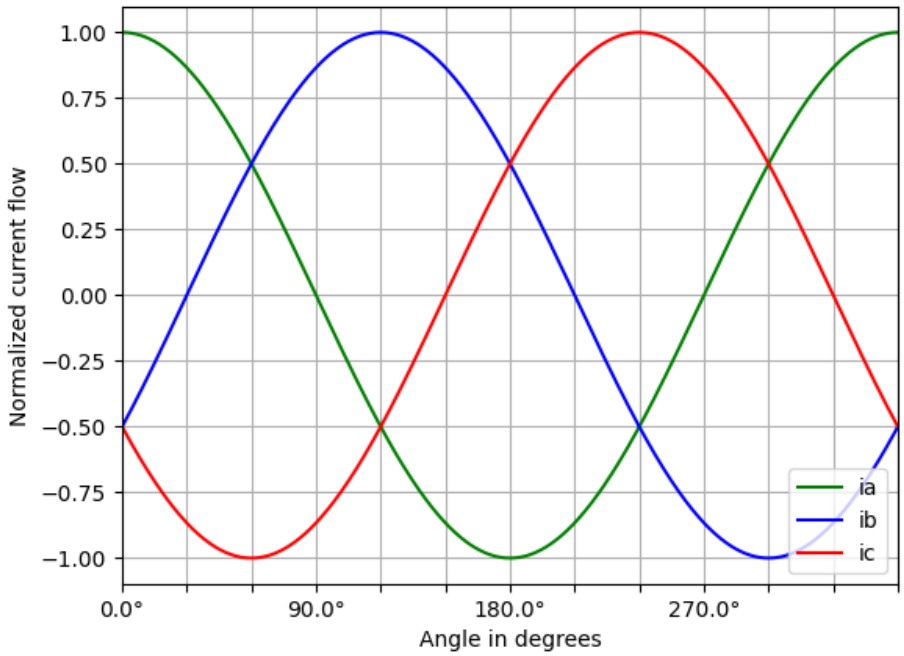


Figure 2.2: 3 phases following a sinusoidal waveform offset by 120 degrees from each other.

voltages hence controlling the current flow that creates the magnetic field, is called an inverter. For the inverter to achieve commutation, the position of the rotor needs to be known.

When working with the position of the rotor, it is important to differentiate between electrical and mechanical degrees. Mechanical degrees are the physical angle of the rotor, while electrical angles depend on the number of physical magnets placed within the motor. The formula for the relationship between electrical angle and mechanical (Physical) angle is (eq 2.1) [29]. Where each physical magnet in the motor is one pole.

(a) Wye configuration (b) Delta configuration

Figure 2.3: Different coil configuration for 3-phase BLDC motor

Every pole comes in a pair where one magnet has the north pole facing the rotor, while the other magnet has its south pole facing the rotor. A motor with eight magnets would have four pole pairs. **If not mentioned otherwise, all angles mentioned during the background will work with electrical angles.**

$$\psi_{electrical} = \theta_{mechanical} * PolePairs \tag{2.1}$$

For direct-driven systems, we often try to sense the motor's position as this allows us to apply the proper voltages. With the knowledge of the position, it is possible to calculate what voltages are required and where to apply them to achieve the desired position or velocity. As for the sensing of this position, it is possible to sense the movement by using data collected by internal sensors, external sensors, a combination of both, or with no sensors at all.

The so-called sensorless method often relies upon measuring currents and voltages induced by the rotor's magnets passing by the stator's electric coils (also called Back-EMF[11]). The data is then combined into a model to stipulate the voltages we want to apply. A big caveat to the sensorless approach is the measurement of the currents in the coils. The current flow must be measured when no voltages are applied to the circuit.

2.5 Clarke Transformation

The Clarke transformation (also called Alpha-Beta transformation)[14] is a transformation to simplify the calculation and use of three-phase circuits. A Clarke transformation is applied to a system where a, b and c are stationary components of a three-phase reference frame and it simplifies them into an orthogonal reference frame with α, β and 0 components. In a BLDC motor consisting of 3 phases 120 electrical degrees apart, the current in each electrical magnet is denoted as i_a, i_b and i_c (referred to as a,b and c in

2.4). According to [14] "by definition instantaneous phase currents i_a, i_b, i_c of normal phase order abc at any point of their instantaneous α, β, 0 components of current, are"

$$i_a = i_\alpha + i_0 \tag{2.2}$$

$$i_b = -\frac{1}{2}i_\alpha + i_0 + \frac{\sqrt{3}}{2}i_\beta \tag{2.3}$$

$$i_b = -\frac{1}{2}i_\alpha + i_0 - \frac{\sqrt{3}}{2}i_\beta \tag{2.4}$$

solving for i_α, i_β and i_0 via simultaneous solution gives the following equations

$$i_\alpha = \frac{2}{3}(i_a - \frac{(i_b + i_c)}{2}) \tag{2.5}$$

$$i_\beta = \frac{(i_b - i_c)}{\sqrt{3}} \tag{2.6}$$

$$i_0 = \frac{(i_a + i_b + i_c)}{3} \tag{2.7}$$

Given that i_a, i_b and i_c are in a balanced system, the assumption below holds.

$$i_a + i_b + i_c = 0 \tag{2.8}$$

This leads to the simplified expression for i_0, i_α and i_β.

$$i_0 = \frac{0}{3} = 0, \quad i_\alpha = i_a, \quad i_\beta = \frac{i_a + 2 * i_b}{\sqrt{3}} \tag{2.9}$$

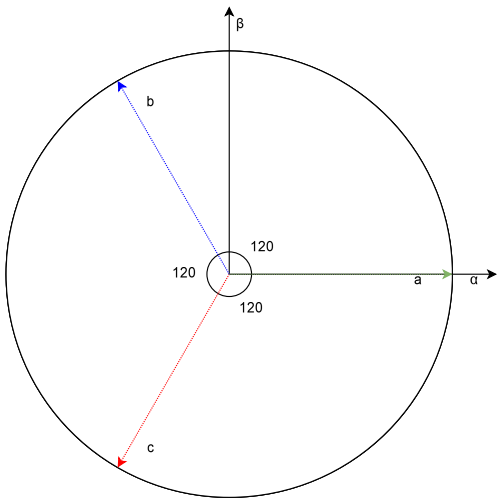


Figure 2.4: Transformation from a, b, c to α, β via Clarke transformation

2.6 Park and inverse-Park Transformation

To maximize the efficiency and torque of a BLDC motor, the magnetic field induced inside the motor needs to be 90^o electrical degrees to the rotor. To describe this optimal rotation of the field given a rotor at direction θ, a transformation from the stationary α, β-frame to the rotary frame following the rotor was proposed by Robert H. Park et al. [41]. The Park transformation transforms a current in the α, β - domain into a coordinate frame, with the two components d and q following the rotor. The Direct - axis (here referred to as d -axis) follows the rotor in the direction of the rotor's magnetic field, while the Quadrature - axis (referred to as q -axis) has a 90^o electrical degrees offset counterclockwise from the d-axis. A magnetic field induced in the direction of the q-axis will give a positive torque, while a magnetic field in the opposite direction will give a negative torque resulting in forces pulling the rotor in the other direction. Applying a magnetic field on the d-axis will result in a strengthened or weakened field of the rotor. The formula for transformation to i_d and i_q given i_a,i_b and i_c is:

$$i_d = \frac{2}{3}\{i_a cos(\theta) + i_b cos(\theta - 120^o) + i_c cos(\theta + 120^o)\} \tag{2.10}$$

$$i_q = \frac{2}{3}\{i_a sin(\theta) + i_b sin(\theta - 120^o) + i_c sin(\theta + 120^o)\} \tag{2.11}$$

Transforming i_a,i_b, and i_c into the Clarke reference frame with the Clarke transformation and applying it to equation 2.10 and 2.11 gives the following equations.

$$i_d = i_\alpha cos(\theta) + i_\beta sin(\theta) \tag{2.12}$$

$$i_q = -i_\alpha sin(\theta) + i_\beta cos(\theta) \tag{2.13}$$

Figure 2.5 shows the Park transformation as an overlay over figure 2.4. The transformation back from the rotating d, q -domain to the α, β domain is called the inverse-park transformation. This transformation is done after a control algorithm is applied. The transformation can be described as:

$$V_\alpha = -V_q sin(\theta) + V_d cos(\theta) \tag{2.14}$$

$$V_\beta = V_q cos(\theta) + V_d sin(\theta) \tag{2.15}$$

Note the use of V_d and V_q in 2.14 and 2.15 as the domain has been changed over to a voltage domain. The change is due to the control algorithm controlling the voltages instead of the current directly.

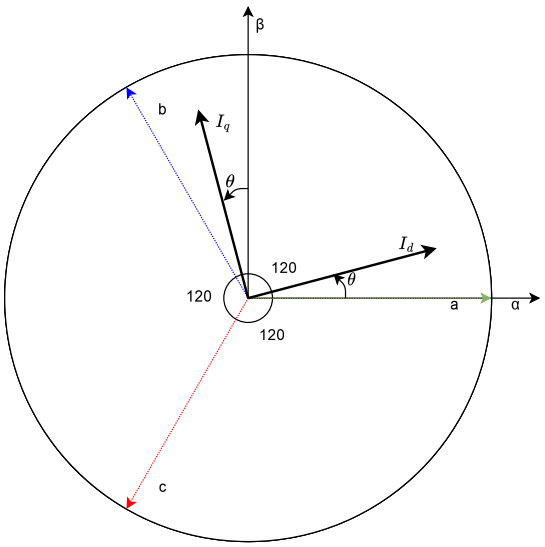


Figure 2.5: Image of Clarke and Park transformation on top of i_a,i_b i_b where θ is the angle of the rotor. Figure based on 2.4

2.7 Control-Theory

Control theory is the application of mathematics, modeling, and real-time data to a dynamic system to deal with the control of said system [6] (p73-130 by S. Simrock). "A dynamical system is a particle or ensemble of particles whose state varies over time and thus obeys differential equations involving time derivatives. Analytical resolution of such equations or their integration over time through computer simulation facilitates the prediction of the future behavior of the system."[17] The representation of these states can be depicted as vectors, numerical values, or through alternative means. An example of this might be the positioning/velocity of an object or the heat of a particle. In the domain of motor control, there might be multiple levels of dynamical systems [16]. Some examples of dynamic systems are the current flowing through a coil in a BLDC motor and the positioning/velocity of the BLDC motor. To reach a desired state in a dynamic system a controller is needed.

There are two main types of controllers, open-loop and closed-loop (feedback). An open-loop controller (fig 2.6) has no feedback from the output to the input. One example of this is a controller based on a timer.

Closed-loop controllers (fig 2.7) have some feedback from the output. The previously

mentioned BLDC positioning is one example of an application where feedback is needed. The feedback here could be the current draw, position, or velocity.

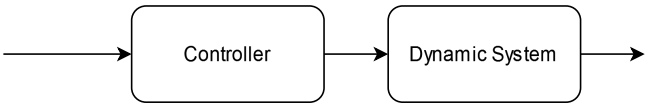


Figure 2.6: Open loop controller with no feedback from the system output.

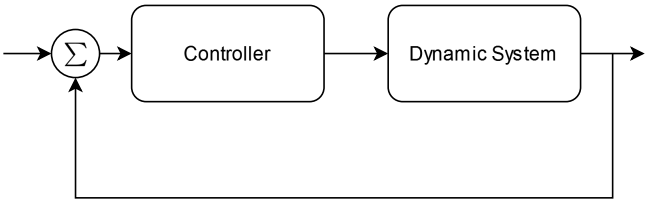


Figure 2.7: Closed loop controller with feedback from the system output.

A widespread simple implementation for a control algorithm is the PID (Proportional-Integral-Derivative) controller. The PID controller works by summing up three terms multiplied by a set of constants K_p,K_d, and K_i. The mathematical function for a PID controller is 2.16, where $e(t)$ = error(time).

$$u(t) = K_p * e(t) + K_i * \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{2.16}$$

The proportional portion of the controller consists of K_p and $e(t)$. $e(t)$ is the error function, this scales based on distance from set-point. The integral part sums the error over time to remove any steady-state error. This is then multiplied up with the constant K_i. The final part of the function is the derivative. This part looks at the pace of the change in the dynamic system state and multiplies it with the last constant K_d. In many applications, this part of the equation is omitted due to noise in the sensing. In the use-case of control of currents inside a BLDC motor, the standard has been to use a PID controller with no derivative part (in short, a PI controller). In some use cases, a PID controller might be too simple/inaccurate for the application or if there is a need to decrease the need for sensing, a model of the dynamic system can be used. The more sophisticated models might also lead to faster response and more accurate responses with the need for less tuning or smaller overshoots/settling times.

2.8 Cordic Algorithm

CORDIC (Coordinate rotation digital computer)[56] is an algorithm created by Jack E Volder. The CORDIC algorithm is able to calculate trigonometric functions, hyperbolic functions, square roots, multiplications, divisions, exponents, and logarithms with an arbitrary base with efficient use of resources. This is done by iteratively adding, subtracting, shifting, and scaling the inputs until the desired accuracy is achieved. The CORDIC algorithm has two modes, rotation and vectoring mode. In this thesis, the CORDIC algorithm was used in vectoring mode to calculate polar coordinates (θ and amplitude) from Cartesian coordinates (x,y). In this mode, the x input needs to be a positive number while the y coordinate is arbitrary. This is due to a limit in the algorithm where the rotation of any angle between -99.88° and 99.88°, shown in equation 2.24. To meet this requirement the x component of the position is taken in as an absolute value. This effectively mirrors the point, and the requirement for the algorithm is fulfilled. The mirroring of the position is adjusted after the algorithm has calculated an answer by adding 180° and subtracting θ. To calculate the angle and length of our vector a series of rotations is performed. The direction of rotation is chosen so that $||y||$ is minimized. Firstly a rotation of 45^o is performed. Then the subsequent rotations will be in a decreasing order where we rotate by less and less until the vector converges towards $y = 0$ and $x = K * r$. Every rotation performed is recorded and will sum up to the initial angle of the vector. The value of our final x will contain the amplitude of our vector scaled by a factor of K. The factor K is pre-calculated and the output is scaled. The X and Y components of a vector can be described as

$$(x, y) = (r * cos(\theta), r * sin(\theta)) \tag{2.17}$$

Rotating equation 2.17 with an angle of $\pm\alpha$ gives the following equation

$$(x', y') = (r * cos(\theta \pm \alpha), r * sin(\theta \pm \alpha)) \tag{2.18}$$

Equation 2.18 can be rewritten as

$$x' = r * cos(\theta) * cos(\alpha) - r * sin(\theta) * sin(\alpha) \tag{2.19}$$

$$y' = r * sin(\theta) * cos(\alpha) + r * cos(\theta) * sin(\alpha) \tag{2.20}$$

inserting equation 2.18 into 2.19 and 2.20 gives the formulas

$$x' = x * cos(\alpha) - y * sin(\alpha) \tag{2.21}$$

$$y' = y * cos(\alpha) + x * sin(\alpha) \tag{2.22}$$

Rewriting the formulas 2.21 and 2.22 into a matrix multiplication gives:

$$K \begin{bmatrix} 1 & -tan(\alpha) \\ tan(\alpha) & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

where $K = cos(\alpha)$

The angle α is chosen from the formula below.

$$\alpha_n = tan^{-1}(2^{-n}) \tag{2.23}$$

The angle from equation 2.23 can be used as a base for the CORDIC algorithm since equation $2.23 > 90^o$.

$$\theta_{max} = \sum_{n=0}^{\infty} tan^{-1}(2^{-n}) \approx 99.88^o \tag{2.24}$$

Using equation 2.23 as the angle for iteration n, the computation of equation 2.21 and 2.22 simplifies into:

$$x' = x - y * 2^{-n} \tag{2.25}$$

$$y' = y + x * 2^{-n} \tag{2.26}$$

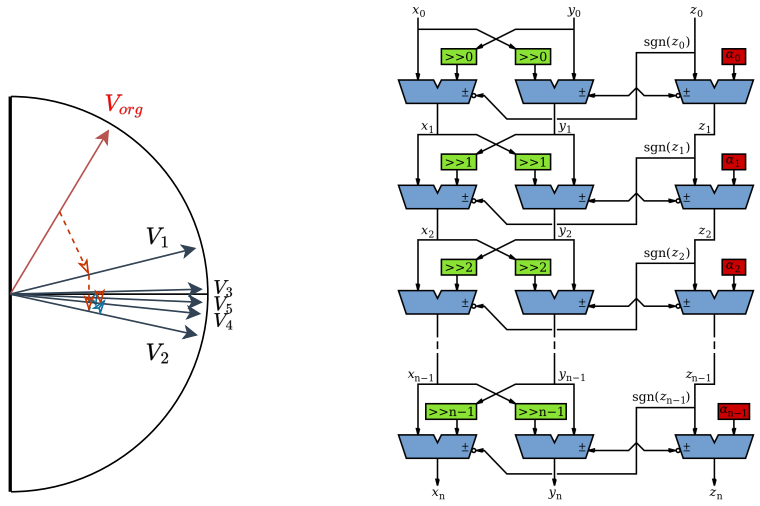
The computation in equation 2.25 and 2.26 has now been reduced to a simple addition and bit-shift operation. The formulas described in 2.25 and 2.26 are then repeated n times with the previous x', y' as inputs until the desired accuracy is achieved. After n iterations, the result is normalized by multiplying with our pre-calculated constant K. The constant K is found via the equation 2.27.

$$K_{iterations} = \prod_{n=0}^{iterations} cos(\alpha_n) \tag{2.27}$$

The constant K and the angle α for the 10 first iterations are listed in table 2.1.

n	$K_n approximate$	α_n
0	0.707107	45.00000°
1	0.632456	26.56505°
2	0.613572	14.03624°
3	0.608834	7.125016°
4	0.607648	3.676334°
5	0.607352	1.789911°
6	0.607278	0.8951737°
7	0.607259	0.4476142°
8	0.607254	0.2238105°
9	0.607253	0.1119057°

Table 2.1: n is the number of iterations done, K_n is the constant K after n iterations (formula 2.27), alpha is the rotation applied during the current iteration n (formula 2.23).



(a) (b)

Figure 2.8: a) CORDIC algorithm used in rotation-mode moving towards 0° so the resulting amplitude is along the horizontal axis and the initial angle is the result of the angular movement summarized. V_{org} is the initial vector while $V_1, V_2, V_3 ... V_n$ is the vectors created by iteration n. The red-dotted lines show rotations done while Y>1, and blue-dotted lines show rotations when y<1.

b) A Schematic of the unrolled CORDIC algorithm. Source [32]

2.9 PWM/PDM

A metal-oxide-semiconductor field-effect transistor (MOSFET) has the ability to change its conductivity based on the voltage applied to an insulated gate [39]. This means that a high-voltage circuit can be controlled by a low-voltage signal like 3.3V. In some applications, pure switching between 0V and max Voltage is desired. However, a voltage between maximum and 0 might be desired in other applications like motor control. In BLDC motor control, the voltages in each electric coil are controlled by an inverter. This inverter is again controlled through a PWM/PDM signal. The PWM/PDM signal tries to mimic an analogue signal by switching the power supply or inverter between 0% and 100%V faster than the device can load the charge [42]. The output of this switching can then follow an analogue signal between 0 and 100% of the maximum voltage. This is useful in the application of motor control as some implementations need the voltages/currents to follow a three-phase curve like figure 2.2.

2.9.1 PWM

Pulse Width Modulation in short PWM, is one way of modulating the signal between 0 and 1 to approximate an analog signal 2.9. When modulating a PWM signal, we have a fixed frequency at which we run our PWM signal. This frequency may vary from application to application. The duty cycle is the time spent "on" during a PWM period. This means a duty cycle of 50% will spend half the time "on" and half the time off. Given a PWM frequency of 100Hz and a duty cycle of 25%, the PWM signal will stay on for 2.5ms and off for 7.5ms before we start a new period where another signal is sent. The positioning of where the signal is high versus low differs between three main configurations: leading, trailing, and centre. If it is configured for leading the signal starts as high and ends as a low signal. In the trailing configuration, the signal starts as low and has the "on" period towards the end of the cycle. Centring the signal so the on period is placed towards the middle of the signal is called centre configuration. The positioning and creation of a PWM signal can be done through various methods suited for various applications.

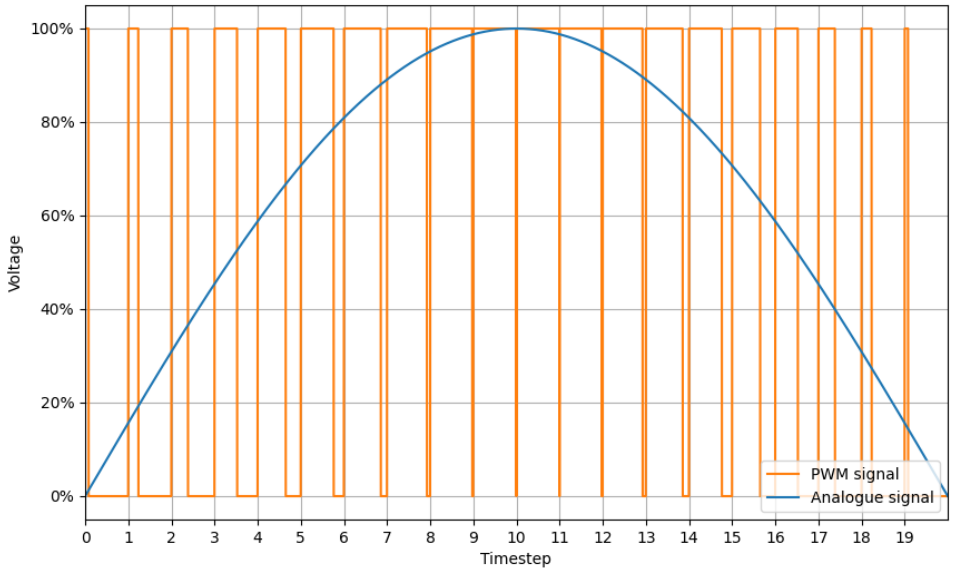


Figure 2.9: A PWM signal overlaid the sinusoidal wave it is trying to modulate

2.9.2 PDM

Pulse Density Modulation in short PDM is another method for modulating a signal to approximate an analogue signal 2.10. PDM works by changing the density of the signal. PDM has a fixed length for the pulses, but how often the pulses appear during our modulation varies. For the example of trying to approximate a voltage of 50%, the output signal might end up as the sequential vector "10101010101010", while a PWM signal would end up as "11111110000000". When designing a system with PDM modulation, one would have to ensure that the pulses are longer than the switching period of the inverter/power supply, as shorter pulses might be too short to change the signal.

2.10 SVPWM

For a system in robotics, the PWM (section 2.9) signals may control the motor and make it move as desired. While a simple commutation following a sinusoidal wave might result in a smooth movement, it is important to note that the torque achieved through this modulation can be improved and, therefore, increased. The torque of a motor is given from the currents induced by the voltage differences between the three phases.

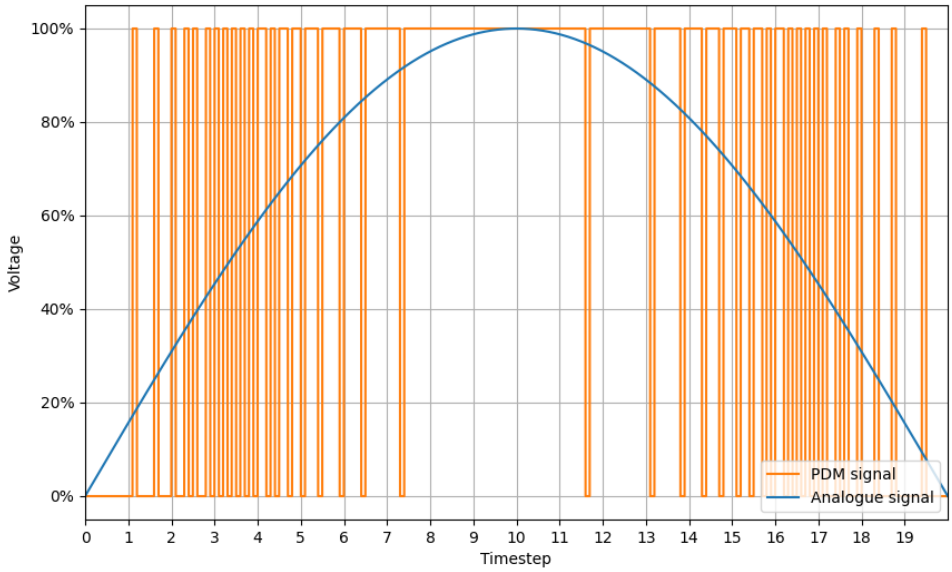


Figure 2.10: A PDM signal overlaid the sinusoidal wave it is trying to modulate

This difference differs from 76-87% (equation 2.29) of the maximum difference possible when one phase is maximum voltage, and another is minimum voltage.

Space vector modulation in conjunction with third harmonic injection could increase the current flow and apply more torque. Space vector PWM is a centred PWM scheme that applies a voltage vector to a three-phase motor [49]. Space vector PWM works by creating a vector containing the desired electrical angle and the magnitude of the voltage. Figure 2.11a shows the basic vectors and directions we modulate during space vector modulation. The vectors show the magnetic fields created by the active and inactive SVPWM states (shown in appendix A in figure A.2 and A.3) where every angle in between the vectors (v) can be made by combining two vectors.

In a setting where the PWM signal for an amplitude of 72.1% and an angle of 33.69 degrees is desired (figure 2.11b). The vector can be achieved by rapidly switching between the vectors v1, v3, and v0/v7. The angle is decided by the amount of time we spend on v1 versus v3, while the amplitude of 72.1% is reached by having the zero vector on for 27.9% of the total time.

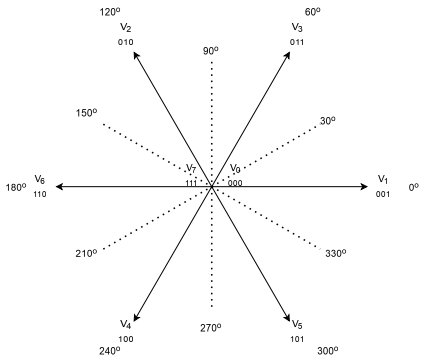
A third harmonic injection amplifies the sinusoidal signal, so we use more of the available power. The principle here is to inject a third sinus signal with a frequency three times higher than the three main phases and an amplitude equal to about 15%

of the maximum power of any phase. This gives us a voltage that follows figure 2.12. Injecting this sinusoidal component into each phase increases the torque and harmonics of the system. However, it also creates a non-zero voltage at the midpoint of the coils in a Y configuration (figure 2.3a).

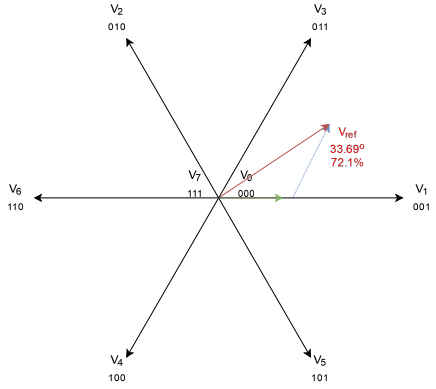
$$V_a = sin(\theta), V_b = sin(\theta + 120°), V_c = sin(\theta + 240°) \tag{2.28}$$

Voltage for V_a, V_b and V_c following a sine-wave (equation 2.28).

$$\frac{\max[V_a, V_b, V_c] - \min[V_a, V_b, V_c]}{2} * 100\% \tag{2.29}$$



(a) The eight base vectors for SVPWM. Each positive base-vector has a 120° separation. Each positive base vector has its respective negative variant offset by 180°. The negative variants exist due to the possibility of a negative or positive charge in the winding. The Zero-vectors v0 (000) and v7 (111) are placed in the origo of the frame. This totals to 6+2 vectors used for SVPWM control.



(b) The vector V_{ref} is created by rapidly switching between V_1, V_3 and the zero vectors. The relationship between V_1 and V_3 gives the direction of 33.69°. Meanwhile, the relationship between the on-states versus the zero-states gives the signal amplitude.

Figure 2.11: Vectors in the Space vector frame

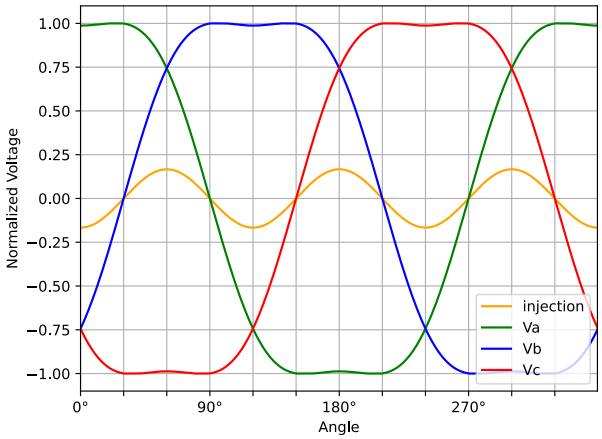


Figure 2.12: Red: phase1, Green: phase2, Blue: phase 3, yellow: injected sinus signal. The Y axis is the voltage, and the X axis is the angle θ. All phases have a 120° offset α from each other. The phases consists of $\frac{2\sqrt{3}}{3}\sin(\theta + \alpha) + \frac{1}{6}\sin(3\theta)$.

2.11 Field-Oriented-Control

Field Oriented Control, in short FOC, is a control methodology for BLDC motors [53]. FOC works by applying a magnetic field to the rotor that is 90^o degrees on the rotor's magnetic field. When this is done correctly we get the maximum efficiency out of the current we send through the motor, as the torque is applied perpendicular to the motor and no torque repels or contracts towards the center of rotation [26]. A FOC system measures the currents in the electrical coils and transforms them into the I_q, I_d domain, where the d axis follows the rotor and the q axis is normal to the d-vector. This is done through the Clarke 2.5 and park 2.6 transformations. The set-point of the currents I_q and I_d are controlled by the user or an outer control loop for positioning/velocity. An inner controller/ model uses the setpoint and measurements to calculate what voltages to apply in the d,q domain following the rotor. The set-point values for this controller are often set to $I_q = I_{desired}$ and $I_d = 0$ as we want the induced current to generate torque. An inverse park transformation is done to convert our vectors to stationary frame (α, β). The deduced components are then used in a PWM/PDM creation scheme. The method of PWM/PDM creation may vary from implementation to implementation, but some common ones are SPWM [47] and SVPWM 2.10. Both of the mentioned schemes create a vector with the magnetic coils for a field that rotates the rotor. Figure 2.13 shows a typical FOC system with SVPWM used for the generation of PWM signals.

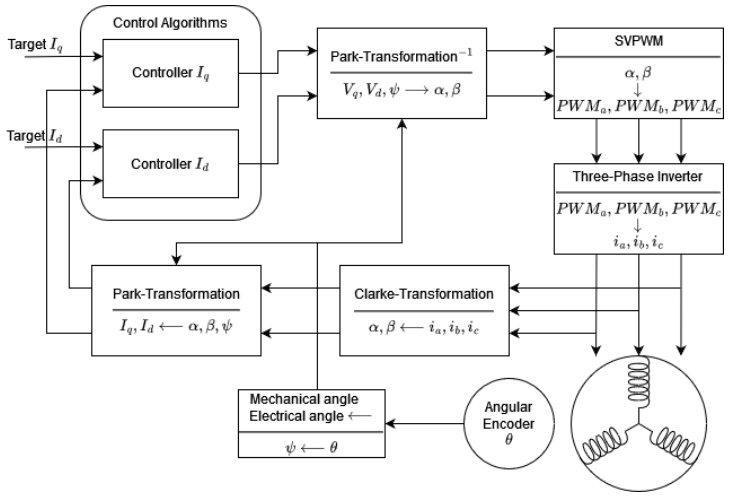


Figure 2.13: A diagram over a possible implementation of FOC using Clarke, Park, PI-controllers, Inverse Park, and SVPWM to control one motor. Simplified image of FOC core 3.11

2.12 Related Works

In this section, we will explore related works in robotics, FPGAs, and motor control. Since the invention of BLDC motors in the 19th century [7], extensive research has been conducted to drive their further development.

2.12.1 FPGA-Based Robotics and Automation

An article on "fpgainsights" starts with the statement: "Innovation is king in the rapidly changing industries of robotics and automation. Hardware and software distinctions are becoming increasingly hazy because of Field-Programmable Gate Arrays (FPGAs), which have become essential parts. Robots are now equipped with FPGAs, which enable real-time data processing, quick execution of sophisticated algorithms, and quick environment adaptation."[44]. In this article, it is concluded that FPGAs have started a new area within the field of robotics and automation. Throughout the article, multiple applications of FPGAs are mentioned. One such application mentioned is Real-Time control, which includes motor control and control loops.

2.12.2 A Survey of FPGA-Based Robotic Computing

The survey named "A Survey of FPGA-Based Robotic Computing" by Zishen Wan and Bo Yu et al.[57] provides a comprehensive overview of FPGA-based robotics and the

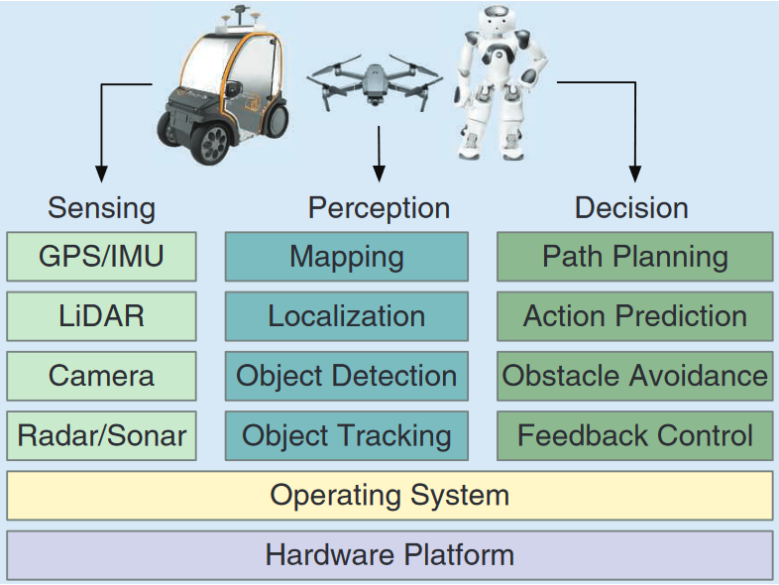


Figure 2.14: The stack of the robotic system. source: [57]

previous research conducted in this domain. The survey discusses the benefits of FPGA and FPGA + SoC implementations over traditional CPU and GPU implementations. During the survey, a focus has been on the acceleration of robotic application workloads (figure 2.14), perception, localization, and planning and control[2]. The survey indicates that FPGAs can be well suited for robotics and that "FPGA can achieve more than 10x better performance and energy efficiency compared to the CPU and GPU implementations." for some specific implementations. It is also mentioned that "the authors believe that FPGAs are the best compute substrate for robotic applications [...] robotic algorithms are still evolving rapidly, and thus any ASIC-based accelerator will be months or even years behind the state-of-the-art algorithms". However, it is also mentioned that FPGAs are not the mainstream platform for robotics as they are harder to program, and the supply of FPGA engineers is limited.

[2]computation of how the robot should maneuver

2.12.3 Efficient FPGA implementation of Field Oriented Control for 3-Phase Machine Drives

As mentioned in both subsection 2.12.1 and subsection 2.12.2, using FPGAs in robotics may provide multiple benefits. In a paper by Burak Tufekci et al.[51] field, oriented control has been implemented on FPGA and optimized using CORDIC 2.8. The proposed system in this paper aims to use the CORDIC algorithm to increase the performance in terms of area and speed with respect to other FPGA-based FOC designs. The paper's introduction mentions that FPGA implementations have latencies as low as one millisecond or less, while "the best CPUs introduce latency of approximately 50 milliseconds". The deterministic nature of FPGA designs is also mentioned, as there is no underlying cache or OS. The proposed system using Cordic can run the clock at 100MHz on a Zynq-7020 FPGA. However, it should be noted that the FOC core implemented in this paper supports only one motor. To accommodate multiple motors, the utilization of multiple cores would be necessary.

2.12.4 Rapid prototyping

A multi-axis FOC system was created in the thesis "Rapid prototyping -development, and evaluation of Field Oriented Control using LabView FPGA" by Joakim E. and Luciano H. [21]. Their report concluded that controlling multiple motors with FOC using the same FOC loop is possible without losing performance. However, it is stated that due to timing problems in the multi-axis implementation, they could not control more than two motors simultaneously. They also concluded that if optimizations were to be made, it would be possible to control up to four motors. The new limiting factor would then originate from a limitation in the amount of digital IO. A total of two designs were created using LabView [59]. One was designed for single-axis control, while the other supported multi-axis control.

As written in [37], LabView will give reduced project completion time. It is also said that a LabView implementation will include more overhead than a system written in HDL.

Chapter 3

Implementation

This chapter aims to explain and give insight into the hardware and software developed and used during the testing of the pipelined FOC core. The section 3.1 contains the physical hardware used and the physical configuration used during the testing and development of the core. Section 3.2 contains information about the implementation of the pipelined FOC-core and the modules contained within. The following section 3.3 contains information about how an outer compatibility layer for communication with sensors and an outer system was established. The last chapter 3.4 contains information about the code and protocols the SoC uses to communicate with an external computer.

3.1 Hardware/electronics

3.1.1 Trenz FPGA

To create a system able to control multiple motors simultaneously, the Trenz TE0720-04-61C33MAS Starter Kit[25] was chosen. This kit includes a TE0720-04-61C33MAS SoC module featuring a Xilinx Zynq™XC7Z020-CLG484C FPGA with an ARM dual-core Cortex-A9 MPCore on a Te0703-07 carrier board. The board module contains 1 GByte (32-bit) DDR3 SDRAM, 32 MByte QSPI Flash memory (for configuration and operation), and 8 GByte e.MCC (Embedded MultiMediaCard). A variety of parameters were looked at when deciding on this board. The driving considerations for choosing this FPGA with a carrier card were the 152 General Purpose Input/Output pins (GPIO-pins), the availability, and the use of a Zynq 7000 series SoC. The FPGA + carrier card can also provide 3.3V VCC for GPIO. The Te0703-07 carrier board has an integrated RJ45 GbE connector that was used for communication while testing. The use of a Xilinx

SoC was important as my previous experiences with the toolchain created familiarity with the workflow and libraries. As for the 152 IO-Pins, they opened up the possibility of binding and testing the IO required for driving multiple motors connected to the system.

Figure 3.1: Trenz TE0720-03-61C33MAS Starter kit [25]

3.1.2 BLDC-Motor

The BLDC motor used for testing the implementation of a FOC was the T-MOTOR MN4004-21 KV400 [38]. This is an out-runner BLDC motor with 18 electromagnets and 24 magnets, giving it 12 pole pairs. Table 3.1 shows a list of the core specifications for the motor. The phases are connected in a wye configuration (figure 2.3a).

Motor Dimensions	Ø44.35 x 19mm	Weight (incl. cable)	53 g
Lead	50 mm	Internal Resistance	359 $m\Omega$
Stator Diameter	40 mm	Configuration	18N24P
Idle Current(22V)	0.2 A	Rated Voltage(Lipo)	4-6S
Max. Power(180s)	300W	Peak Current(180s)	12A

Table 3.1: T-Motor MN4004-21 KV400 Specifications

Figure 3.2: T-motor MN4004-21 KV400 [38]. The electromagnets (copper coils) are placed on the stator within the motor, while the outer shell of the motor is the rotor, where the permanent magnets are placed.

3.1.3 Driver

To power our BLDC motors, the BOOSTXL-DRV8305EVM [5] from Texas Instruments was used as the driver board. The driver board includes the 3-phase smart gate driver DRV8305 [13] with current shunt amplifiers and SPI (Serial Peripheral Interface). The SPI interface was not used in this setup. The driver board supports voltages between 4.4 and 45 while delivering currents up to 20A peak. It has 6x CSD18540Q5B N-Channel MOSFETs [12] and Low-side current shunt sensing for each half-bridge through the DRV8305's triple Current-Shunt amplifiers.

The current measured with the Current-Shunt amplifiers is outputted as voltages between 0 and 3.3V. This voltage is centered around 1.65V, where 0V equals a current flow of -20A, 1.65V = 0A, and 3.3V equals 20A flowing through the respective phase/coil. Voltage sense for the supply bus and each phase output are reported as scaled voltages for 3.4-45V operation. (The voltage output is named VSENPVDD, VSENA, VSENB, VSENC in figure 3.3b).

The driver has 6 PWM input pins (fig 3.3b) A_{High} = PWMHA, A_{Low} = PWMLA, B_{High} = PWMHB, B_{Low} = PWMLB, C_{High} = PWMHC, C_{Low} = PWMLC that was controlled by our PWM - scheme. Said input pins support 3.3V with PWM frequencies up to 200KHz. The driver supports three different driving modes for the PWM pins: 6-PWM mode described in table 3.2, 3-PWM mode described in table 3.3 and 1-PWM mode where the PWM signal is sent on pin A_{High}, and stored in an internal 6-step block commutation table where the combination of A_{Low}, B_{High} and B_{Low} encodes what index to write the PWM value to.

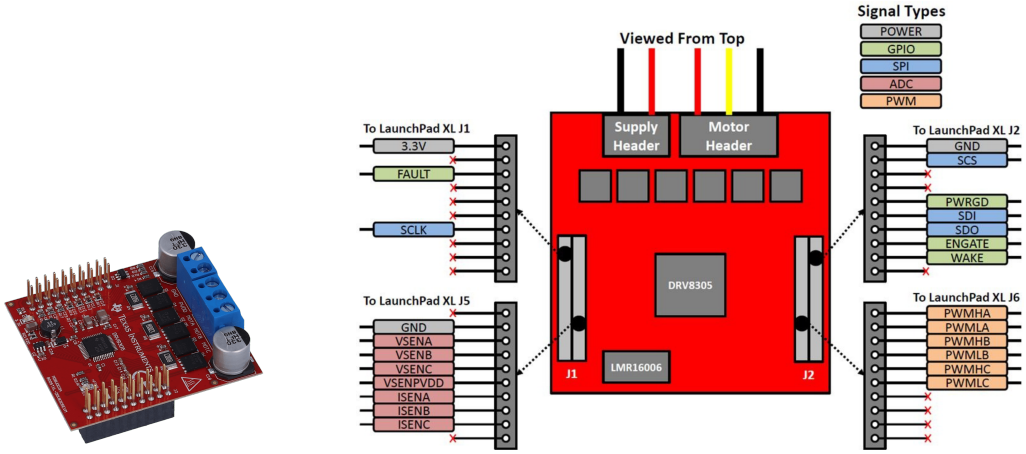
The H-Bridge contained within The DRV8305 is protected by the integrated state machine (TDRIVE). The TDRIVE protects against excessive currents on the gate drive outputs and shoot-through in the MOSFETs on the driver board. This is done through internal handshaking when switching from the low to the high side and vice versa on the MOSFETs. Thus preventing a High and Low signal from shorting out the MOSFETs. One example of this safety working can be seen at $\mathrm{PWM}_{High} = 1$, $\mathrm{PWM}_{Low} = 1$ in table 3.2. To enable the driver and shunt amplifiers, the pin ENGATE in figure 3.3b must be pulled high, or the device stays inactive.

PWM_{High}	PWM_{Low}	$Gate_{High}$	$Gate_{Low}$
1	1	L	L
1	0	H	L
0	1	L	H
0	0	L	L

Table 3.2: 6-PWM mode, where the PWM_{High} and PWM_{Low} signals are inputs for A, B or C

PWM_{High}	PWM_{Low}	$Gate_{High}$	$Gate_{Low}$
1	X	H	L
0	X	L	H

Table 3.3: 3-PWM mode, where the PWM_{High} signals are inputs for A, B or C. While PWM_{Low} is not connected



(a) Image of the physical board, image from [5]

(b) Pinout for the BoostXL-DRV8305EVM, image from EVM User's guide at [5]

Figure 3.3: BoostXL-DRV8305EVM

3.1.4 Angular Encoder

During FOC, the motor's angle (θ) needs to be known to apply the appropriate PWM signal. There are different methods to sense the position of the rotor. This can be done by using an encoder, resolver, hall effect sensors, or sensorless. During our experiment to test the validity of the core, the incremental rotary encoder AEDT-9810-Z00 [31] with a 625 CPR ROP 11 GMNA056 5189-1092 wheel was used. The Encoder uses a voltage of 5V and can be used without pullup resistors on the output pins. The AEDT-9810-Z00

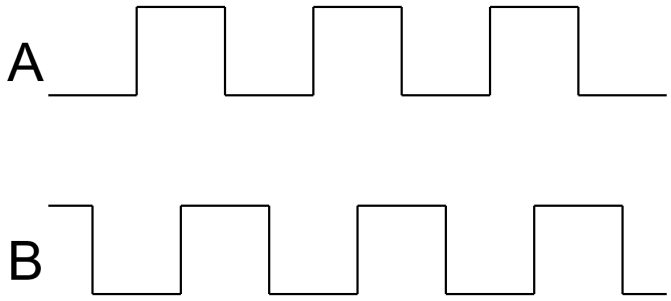


Figure 3.4: Channel A and B follow each other when rotating clockwise.

is an optical encoder with three channels for incremental measurement. This encoder detects light passing through a disc with three optical channels, where the light can either be blocked or allowed to pass.

The three channels are named A, B, and I. Channel I is blocking once every 360^o and indicates that the encoder has gone one full revolution. 625 CPR stands for Counts per Revolution, indicating how many pulses we will get on tracks A and B per full rotation. The channels A and B are offset by half a pulse from each other. When rotating clockwise, Channel A is leading on Channel B (figure 3.4). It is possible to calculate the position of the encoder by looking at what transitions the channels make compared to each other. If 'A becomes "NOT B" or B becomes A', we have a positive rotation, while if 'B becomes "NOT A" or A becomes B', we should decrement our position. The encoder includes undisclosed internal logic that increases the 625 CPR to 5000 points per revolution per optical channel. This gives an accuracy of $\frac{360^o}{5000} = 0.072^o$ physical degrees per channel. Combining the two optical channels makes it possible to achieve a measurement accuracy as fine as $\frac{0.072^o}{4} = 0.018^o$, representing the maximum attainable precision with this particular encoder. The maximum output frequency is 1000 KHz, which gives us a maximum speed of $\frac{1000}{5000} * 1000 = 200$ rounds per second, or $200 * 60 = 12\ 000$ RPM. In our case, an accuracy of 0.072^o and a maximum speed of 12 000 RPM is adequate for the physical assurance testing of the setup.

3.1.5 5V level shifter

Since the IO on the FPGA3.1.1 uses 3.3V while the angular encoder 3.1.4 has 5v on the output pins, a level shifter was required. The Adafruit 8-channel Bi-directional Logic Level Converter [33] with the chip TXB0108 [52] was used in the physical setup. The Level shifter converts a digital signal from one voltage to another. In this case, the level

shifter converted 5V to 3.3V. The shifter can convert up to 8 channels of signals with a data rate of up to 100 Mbps per channel. Since the encoder 3.1.4 runs at 100KHz, the data rate of 100 Mbps is sufficient. Due to the nature of this being a bi-directional converter, a series of resistors had to be added on the 3.3V side of the signal. The resistors were added to ensure the correct direction of signal conversion. Omitting the resistors caused the level shifter to reverse the signal conversion and amplify noise from the FPGA 3.1.1 pins.

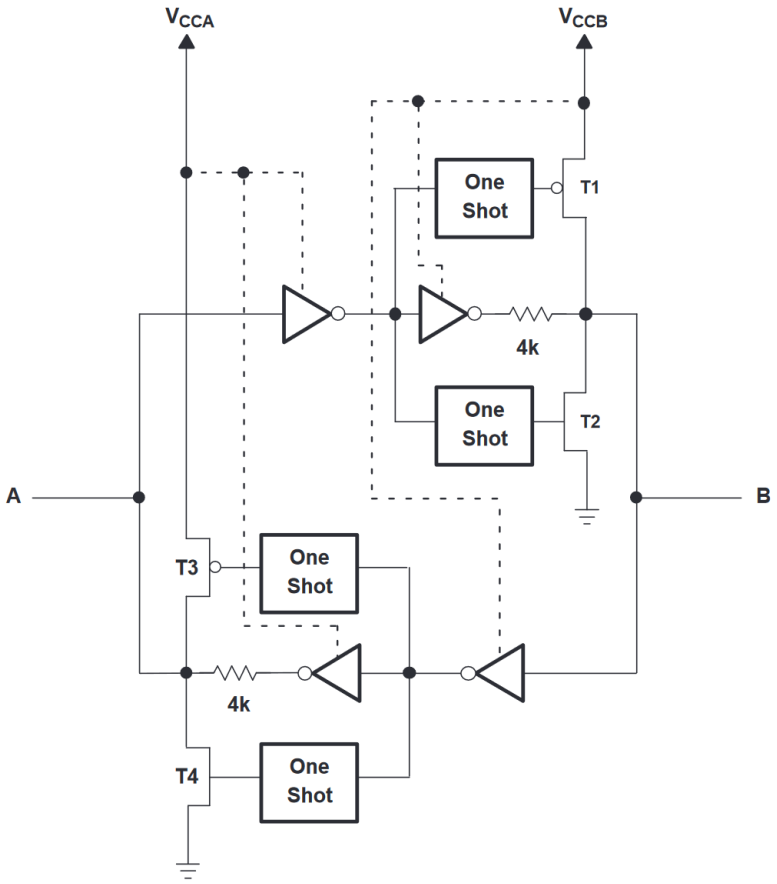


Figure 3.5: Architecture of TXB0108 I/O Cell. The physical setup placed a resistor between A and the FPGA 3.1.1 connected to A. Side B was connected to the encoder 3.1.4. Figure from [52]

3.1.6 ADC

When measuring the current inside phases of the BLDC motor 3.1.2, the currents were scaled and converted to a voltage between 0 and 3.3 V by 3.1.3. To use this voltage in our calculations inside the FOC, the use of an Analogue to Digital Converter (ADC)

was necessary. The EVAL-AD5592R-PMDZ-ND 8 channel, 12-bit ADC [22] was the chosen ADC for this purpose. The EVAL-AD5592R-PMDZ-ND contains the AD5592R [3] ADC/DAC chip, which has 8 configurable channels that could be set to Analogue to Digital converters (ADC) or Digital to Analogue converters (DAC). Each channel has a resolution of 12 bits and goes through the same multiplexed ADC. The internal ADC has an input range of 0 to V_{ref} or 0 to $2 * V_{ref}$ configurable by the user.

For communication between the ADC/DAC and the FPGA, the Serial Peripheral Interface (SPI) was used as it is offered with native support on the chip. The communication works by exchanging 16 bits of information when the master (FPGA) selects the ADC/DAC. The first messages to the AD5592R are used to set up the IO pins for ADC/DAC data.

The ADC can be set to two different modes of operation. In the first mode, the FPGA sends an instruction containing what pin to read the analog value from. This value is sampled on the falling edge $\overline{\text{SYNC}}$ signal. This sampling takes 500 ns. The response for the requested ADC signal is sent on the exchange of bits between the ADC and the FPGA following the next falling edge of $\overline{\text{SYNC}}$ after the sampling.

If more than one channel is to be converted, the second mode of operation can be used. Here, multiple channels and a repeat flag can be selected, and the data will then be measured on falling edges of $\overline{\text{SYNC}}$ in descending order and sent consecutively to the FPGA. To initiate each operation for a response, the FPGA can send "No operation" (NO_{ops}) instructions over SPI. The repeat flag makes the ADC repeat the measurement cycle and send it on repeat.

The maximum speed of the SPI is 20 MHz, sending the data from one ADC conversion will then take a minimum $50ns * 16bit = 800$ ns. Since The AD5592R is a multiplexed ADC, the measurements of the phases are not done at the exact same time. This can create problems when the duty cycle of the PWM signal is too high for the ADC to send a response for all three phases, as it would take $800 * 3 = 2400$ ns.

When using multiple motors and multiple ADCs, it is possible to shorten this time, as it is possible to run every PWM cycle with an offset from each other. This means a motor can have its three phases connected to different ADCs, and the measurements can be taken simultaneously. With the PWM offset between each of the motors, the time of 800 ns can be used as the minimum off-time for the duty cycle. In the physical setup used for this thesis, one EVAL-AD5592R-PMDZ-ND was used, and the repetition

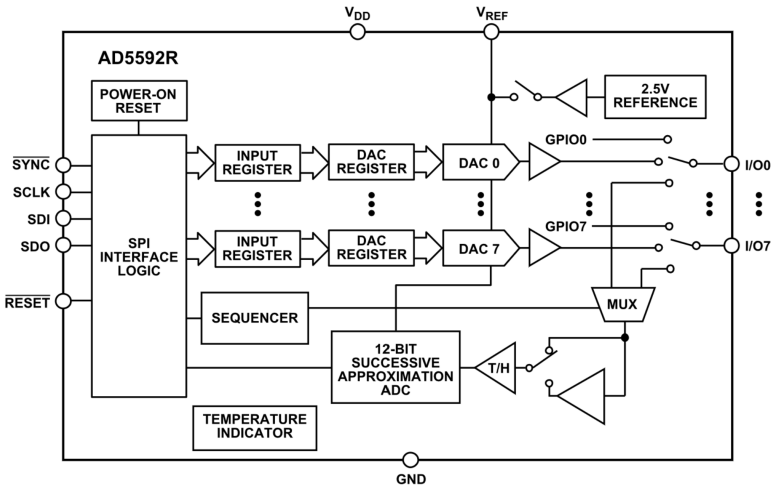


Figure 3.6: Functional Block Diagram of the AD5592R. Image from [3]

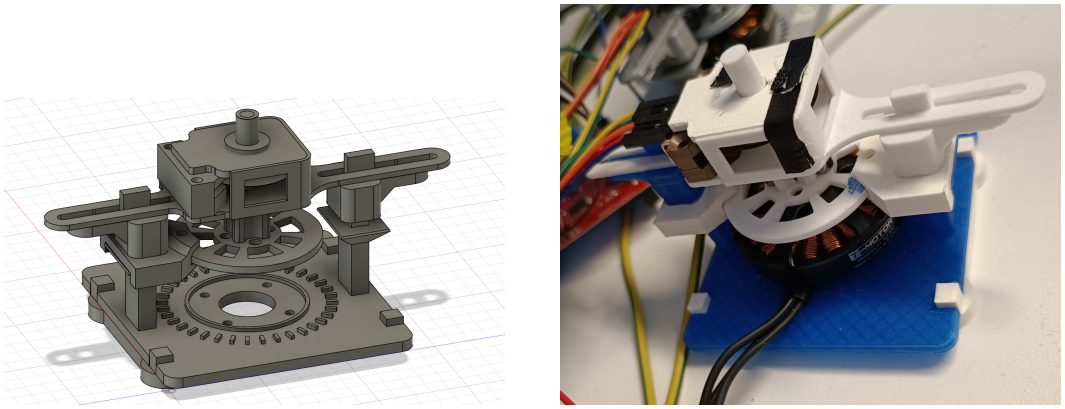
of reading was enabled. The measurements were consecutively read, and if they were taken during the off-period of the PWM signal, they were sent to the FOC-core.

3.1.7 Power

A variable power supply was used to power the driver 3.1.3 and BLDC motor 3.1.2. This was the Manson SPS9600 1-15V 60A [50] as it was provided by the ROBIN group [45]. This power supply powered two drivers with one corresponding BLDC motor each and was set to 9V during the testing of the FOC-core. As for powering the angular encoder 3.1.4 and the level shifter 3.1.5, a generic 230V 50-60 Hz AC to 5V DC adapter was used. As the adapter was only used for powering the encoder and level shifter, a generic low-wattage, low-ampere adapter was sufficient.

3.1.8 CAD models

A set of simple 3D models (figure 3.7a) were constructed to hold the angular encoder and BLDC motor in place. During the design of the components, the fixtures for placements were made with some tolerance, making it possible to move the components a tiny bit by hand and allow them to flex into place. All components were designed with computer aided design (CAD) and 3D printed using fused deposition modelling (FDM) technology. The components were connected using M3 bolts, duct tape, and friction (figure 3.7b).



(a) Image of the 3D-model for the setup surrounding the motor

(b) Image of the physical motor with setup

Figure 3.7: Two images of the motor setup. One of them being the digital models while the other one being the physical version of the model

3.1.9 Wiring

To develop and test the FOC-core 3.2, the components described in section 3.1 were connected together as shown in figure 3.9. The setup consisted of 2x driver boards 3.1.3, 2x BLDC motors 3.1.2, 2x angular encoders 3.1.4, 1x ADC 3.1.6, 1x level shifter 3.1.5, 1x breadboard, 1x TE0720-04-61C33MAS FPGA 3.1.1, 6x 4.7 $k\Omega$ resistors and two external computers connected to the FPGA. One computer was connected via USB to program the device, while the other was connected via Ethernet to send instructions to the FOC-core. Jumper wires were used in conjunction with a breadboard when transferring signals between components. The breadboard was mainly used to house the level shifter and split up the low-voltage power lines for the low-power components (3.3V, 5V, and GND). The six 4.7 $k\Omega$ resistors were placed on the breadboard and connected between the FPGA and the level shifter on the 3.3V side. As for powering the motors and driver boards, some more robust wires were used with a Wago connector to split the power between the Driver boards.

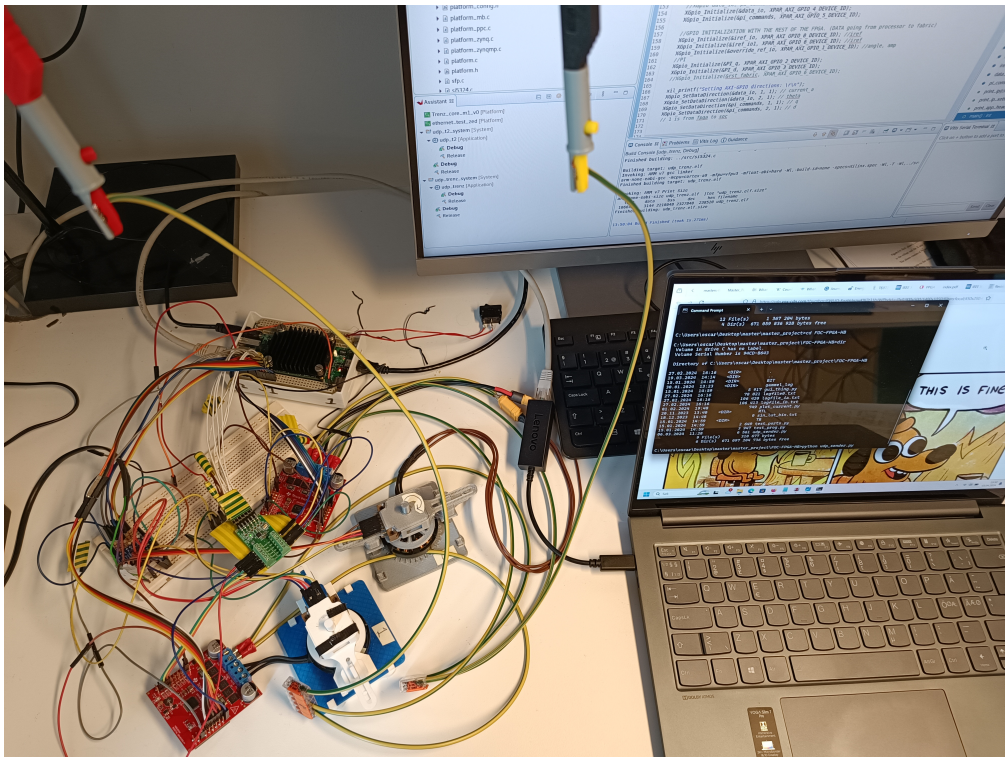


Figure 3.8: The physical setup of the electronics.

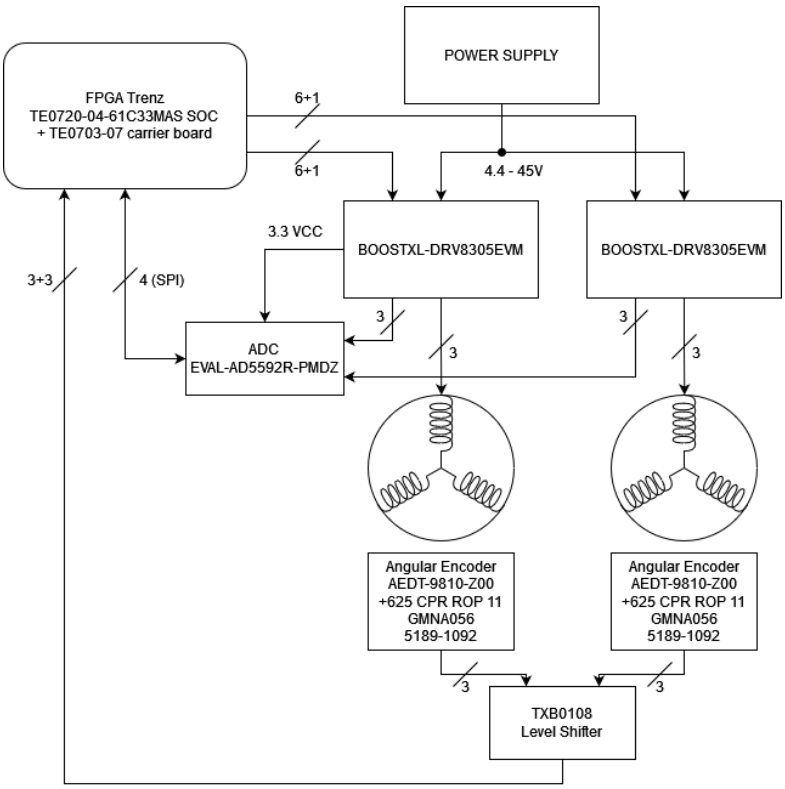


Figure 3.9: Electrical diagram of physical setup with the aforementioned components.

3.2 FOC-FPGA-HB core

Several HDL files had to be created to implement a pipelined FOC core. The Hardware Description Language of choice was VHDL 2008. The multiple modules included in the design have their own respective file and have some configuration possibilities through a configuration VHDL file. The inputs and outputs from the core have some special data types that scale their width with the number of motors selected in the configuration file. Those data types mainly consist of arrays of signed/unsigned numbers and arrays of logic vectors. The input and output width from the processing system and the sensors can be defined in a configuration file. The FOC-core and all its constituent modules can be found at [4], representing my original work.

3.2.1 Configuration file

The main configuration module is placed within a file called config.vhd. This file is mainly used for constants in the implementation like the width of vectors, number of motors, PWM frequency, the different datatypes used when interfacing with the top module, and a procedure for setting PWM signals depending on which MOSFETs one would want active for powering the electromagnets. The table 3.4 contains some of the PWM states configured in the configuration file. This procedure is used by the SVPWM module 3.2.8.

Configuration name	A_{High}	A_{Low}	B_{High}	B_{Low}	C_{high}	C_{Low}
AB_c	1	0	1	0	0	1
A_bc	1	0	0	1	0	1
BC_a	0	1	1	0	1	0
B_ac	0	1	1	0	0	1
AC_b	1	0	0	1	1	0
C_ab	0	1	0	1	1	0
A_b	1	0	0	1	0	0
A_c	1	0	0	0	0	1
B_a	0	1	1	0	0	0
B_c	0	0	1	0	0	1
C_a	0	1	0	0	1	0
C_b	0	0	0	1	1	0
null_ABC	1	0	1	0	1	0
null_n_abc	0	1	0	1	0	1

Table 3.4: The naming scheme used for the Mosfet configuration during PWM cycles. The configurations used during SVPWM are in red. Blue contains SPWM configurations, while gray includes two null configurations. There exist 13 more null configurations that are not shown in this table. Figures for the configuration can be found in the Appendix figure A.2 and A.1. The naming scheme follows these rules:

1) All null configurations include a null prefix in their name.

2) Active high signals are then named alphabetically in upper case.

3) The low signals comes after the high signals with small letters.

4) A "_" sits in the transitions between rule 1,2 and 3.

3.2.2 Clarke transformation

Clarke_transform.vhd is a module created to perform a Clarke transformation 2.5 on the three input currents i_a, i_b, i_c to the α, β-domain. The module uses two clock cycles and is pipelined so multiple transformations can be done sequentially. Inside this module, the Clarke transformation is done with a few approximations used in the calculations.

$$\frac{1}{\sqrt{3}} \approx 295/2^9 \tag{3.1}$$

$$\frac{1}{3} \approx 85/2^8 \tag{3.2}$$

When trying to multiply by $\frac{1}{\sqrt{3}}$ following equation 3.1, one would multiply said number with 295 and shift it down 9 bits to have the correct answer. The simplified set of equations 2.9 was used during the experiments.

3.2.3 Park transformation

Following the Clarke transformation, a Park transformation is performed in the module park_transform.vhd. The park transformation is a multicycle operation. In the first clock cycle, the module stores our alpha, beta, and ϕ values for use in the next cycle. A request for the sinusoidal value of the angle ϕ is sent out to a lookup table 3.2.5 containing the values scaled up to 16 bits (including the sign). Another request is sent for a cosine value in the following clock cycle. When requesting a cosine value, the Park transformation module requests a sine value of the initial angle with an offset of $+ 90^o$ as $cos(x) = sin(x + 90^o)$. Doing this operation with digital angles is the same as taking the two top bits of the number and adding 1 while copying the rest of the bits over to the new angle. When both values from the LUT have been retrieved, the cosine and sine values are multiplied with α and β according to equation 2.12 and 2.13. The output d and q (representing their respective current in the dq-domain (I_q, I_d)) is then shifted down the length of the sinusoidal value so the numbers are in the same order of magnitude as they entered the module. This module can take a new input every other clock cycle as it uses the same LUT for both the sinusoidal and cosines lookup. The rest of the logic in this module is pipelined. The module uses 16 bits for the input and output signals alpha, beta, theta, d, and q. The angle used for requesting sinusoidal and cosine values is 10 bits, while the response from the LUT is 16 bits, including the sign.

3.2.4 Inverse Park transformation

The inverse park module has almost the same method of operation as the Park_transform.vhd module 3.2.3. This module uses an inverse park transformation section 2.6. The equations 2.14 and 2.15 are used on the inputs (d,q) to convert them into the α, β-domain. A LUT indexed with the angle ϕ contains the sinusoidal values that are multiplied with the d and q values received. A shift operation is then performed to reduce the width of the signal to 16 bits and normalize the result. The module is pipelined and has the same lengths for the input and output variables as the Park module 3.2.3.

3.2.5 Sinusoidal LUT

In this implementation of an FOC-core, a total of 3 lookup tables (LUT) were used to find sinusoidal and cosine values corresponding to the input angle. There are two types of LUTs created for the core. The LUT created for use in the Park and inverse Park transformations stores 14-bit unsigned sinusoidal values between 0 and 90^o degrees. The initial 90^o is expanded to span the whole 360^o space using algorithm 1. Using digital angles, the two top bits can tell what quadrant the angle is in and adjust the output accordingly. The adjustments will be as follows: If it is in the first quadrant (the top bits are "00"), the following bits will be used as the index. If the output is in quadrant number two ("01" as top bits), the index is max_index - θ[top-2:0][1]. When moving over to the third quadrant ("10") as the top bits, the output is the negative of the LUT[θ[top-2:0][1]][2]. For our final quadrant, the output will be the negative of LUT[max_index - θ[top-2:0][1]][2]. The LUT uses two clock cycles to get sinusoidal values for angles between 0 and 360^o.

The other type of sinusoidal LUT used in the core is a sinusoidal LUT that looks up angles strictly placed in the range 0 to 60^o used in SVPWM 3.2.8. The values included in this sinusoidal LUT have been modified to include a third harmonic injection (figure 2.12). This LUT contains 171 16-bit unsigned numbers. No sign conversions are needed in this module since it works as a simple LUT using a single clock cycle.

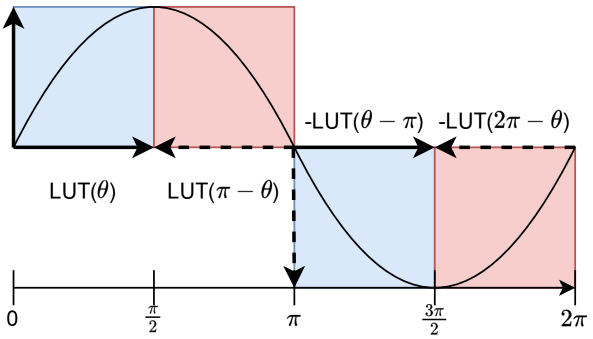


Figure 3.10: A visual representation of algorithm 1 in radians. The blue sections uses ($\theta\%\pi$) as the index, while the red section uses ($\pi - \theta\%\pi$) as the index.

[1]Vector[A:B] returns a vector of size (A-B) and returns the bits from A down to B of the vector.
[2]Indexing with a single element such as an integer returns the element at the corresponding index.

Algorithm 1: Logic to extend a LUT spanning 0 - 90^o to the range 0 - 360^o

Data: θ representing angles between 0 and 360^o scaled to the range 0, 1023

Result: $output_{var}$ represents the scaled to the range -32768, 32767

$\theta_{top} \leftarrow \theta[15:13]$;

$\theta_{bottom} \leftarrow \theta[13:0]$;

if $\theta_{top}[1] = 0$ **then**

  $Sign \leftarrow 0$; ▷ Positive sign on final output

else

  $Sign \leftarrow 1$; ▷ Negative sign on final output

end

if $\theta_{top}[0] = 0$ **then**

  $Lookup\_index \leftarrow \theta_{bottom}$;

else

  $Lookup\_index \leftarrow 255 - \theta_{bottom}$;

end

if $Sign = 0$ **then**

  $output_{var} \leftarrow$'0' & $LUT[Lookup\_index]$;

else

  $output_{var} \leftarrow -$('0' & $LUT[Lookup\_index]$);

end

3.2.6 PI-controller

As mentioned in section 2.7, the FOC-core needs a controller to set pulse widths to induce the desired currents. For this purpose, a set of two Proportional Integral (PI) Controllers were used for each motor in the system. One controller is for the q-axis in the dq-domain, and one is for the d-axis in the dq-domain. The target for i_d is set to 0 as the forces applied here do not result in output torque. The PI controller constructed takes in a strobe enable signal when new data is sent to the controller from the Park transformation 3.2.3. For data input, it takes in two 16-bit signed numbers. The first input is the desired current (i_{ref}), and the other is the current draw in the dq-domain. The module uses two input vectors for the values used for KP and KI in the PI operations (equation 2.16). In the controller declaration, constants have been added to shift the KP and KI values down by a configurable amount of bits. This enables the possibility to scale the output with non-integer values, as Kp could be set to 3 and shifted down 1 after the multiplication, which effectively multiplies Δi by 1.5. The constants for the shift amount are named Total_shift, and the function for its operation can be described with $K_{real} = K * 2^{-Total\_shift}$.

$$\Delta i(t) = i_{ref} - i_{current} \tag{3.3}$$

$$integral(t+1) = integral(t) + (\Delta i * \Delta t) \tag{3.4}$$

$$u(t) = K_p * \Delta i(t) + K_i * integral(t) \tag{3.5}$$

The controller uses the mathematical function described in equation 3.5; however, the multiplication between K_i and integral(t) has been moved to the same clock cycle as equation 3.4 is performed since it makes the timing requirements easier to meet. The new formula for calculating the KI part of the controller becomes 3.6

$$Ki_{integral}(t+1) = Ki_{integral}(t) + K_i * (\Delta i * \Delta t) \tag{3.6}$$

Which leads to the equation 3.7.

$$u(t) = K_p * \Delta i(t) + Ki_{integral} \tag{3.7}$$

In this implementation of a PI controller, the counter used to measure Δt is omitted as the core uses a fixed deterministic time interval between the updates of the controller. During the addition operations, a function for safe addition was created and used. This

safe add operation ensures the controller does not overflow inside the integral sum or on the output. Overflows may cause negative feedback and create a loop where a high positive voltage is applied when a negative voltage is desired. The controller uses 32 internal bits as both the K (K_p, K_i) variables and the current ($i_{ref}, i_{current}$) values uses 16 bits each. The controller uses a total of 5 clock cycles from enable$_{in}$ = 1 to enable$_{out}$ = 1. The module's output uses the range from (15 + Total_shift down to Total_shift) of u(t) as its output.

3.2.7 Cordic

The Cordic module is one of the more complex modules constructed for use in this FOC core. A fair few implementations of FOC cores omit the conversion from the α, β-frame to polar coordinates as an inverse Clarke transformation can return the thee phases V_a, V_b, V_c. However, when third harmonic injection or more complex PWM strategies are in use, a polar reference frame with an angle (ϕ) and an amplitude (ρ) might be desired. The Cordic module constructed for the FOC core uses a total of 10 Cordic iterations from the Cordic algorithm 2.8 and returns both amplitude and angle of the input vector in the α, β-frame to the polar coordinate frame.

The implemented module is pipelined and works by first storing the value of the x sign and converting the value to an absolute positive value. For the 10 next iterations, Cordic computes x' and y' using equation 2.26 and 2.25, swapping the sign of $+/-(2^{-n})$ depending on the sign of the y value. The multiplication with 2^{-n} is done through a bit shift operation. The angle ϕ is calculated by taking $\phi(t+1) = \phi(t) + 2^{-n}$. The values for 2^{-n} are pre-calculated and stored in a 10 x 16-bit array, taking n as the index.

After 10 iterations, the output angle is adjusted in accordance with the initial x sign and the top bit in the calculated angle. If the angle is within 0 - 90^o (aka the sign bit is 0), the output angle has two possibilities. If x had an initial positive value, then ϕ = "00" & abs(ϕ(10))[top-1:0]. If x had an initial negative value and ϕ(10) is in the range 0 - 90^o, ϕ = "01" & abs(180^o - ϕ(10))[top-1:0]. For the two cases when ϕ(10) was in the range 0 - (-90)^o the output angle became either ϕ = "10" & abs(ϕ(10))[top-1:0] or ϕ = "11" & abs(180^o - ϕ(10))[top-1:0] for -x, +x in their respective order.

The final amplitude also needs some correction in the end as the result needs to be scaled with the constant K (equation 2.27). The approximation used for K was multiplying x'(10) with 311 and shifting it down appropriately. Giving a K_{10} value of

0.607421875 (0.0278% bigger than the actual K_{10} value).

3.2.8 SVPWM

SVPWM was the PWM generation of choice for controlling the motor drivers. The SVPWM module is created to handle Clock domain crossings between the PWM clock and the logic clock for the internal logic in the core.

The SVPWM module starts by finding out what hexant the desired angle is placed within. This is done through a series of if statements on whether or not the angle is between 60^o * (n-1) and 60^o * (n), where n is the number of the statement. When the appropriate hexant is found, a ϕ modulo 60^o operation is executed. Here this operation is performed by taking $\phi_{60^o} = \phi - 60^o * (n-1)$ for the positive if statement. Since ϕ is a 16 bit number, the angle 60^o is converted to $2^{16}/6 \approx 10922$. The cycle after the quadrant is found, the order and type of voltage configuration in table 3.4 for SVPWM is calculated by looking up the different sections (fig 2.11a). The conversion from angle and amplitude within a span 60^o to the timing used for the SVPWM vectors is found using a LUT containing 0-60^o 3.2.5. The timing is found by multiplying the amplitude with LUT[ϕ_{60^o}] for state 1, and LUT[60^o - ϕ_{60^o}] for state 2. The time spent in the null state equals the total time in each PWM cycle minus (state 1 + state 2).

Once the timing for each state is calculated, a clock domain crossing between the PWM clock and logic clock is performed to ensure that the PWM states and timings are assigned correctly. A user-configurable downscaling is applied to the amplitude, as the height of the amplitude might exceed the timing of the PWM signal. A safety mechanism is included to remove any overflow introduced in the timing calculations.

The output of this module is a set of PWM wires for A_{high}, A_{low}, B_{high}, B_{low}, C_{high}, C_{low}. They are set through a procedure in the configuration file 3.2.1. As for additional functionality, the PWM module reports when a null state has been entered. This is done by setting either null_state_high or null_state_low to 1. A strobe signal is also sent out whenever the PWM signal has finished.

The module provides high flexibility to customize parameters like when the change of PWM signal happens, the timing for each PWM cycle, and the scaling applied to the amplitude. This module do not support multiple motors, and it is therefore required to add one of this module per motor in the system.

3.2.9 Control-unit

The control module was created to ensure correct startup, reset, override functionalities, and conversion of angles for use in the core. This module ensures that 0 electrical degrees are found during a short startup sequence, that overrides the control system. The startup sequence forces the SVPWM to generate a 50% duty cycle PWM signal at 0^o electrical degrees for a user-configurable amount of time. When the time configured for the override is passed, the offset between the angle sent in from the encoder and 0 degrees is measured and applied. It is also possible to trigger this internal override to any angle and amplitude when this is required in the desired control scheme.

The last function performed by the control unit is the conversion from mechanical angles to electrical angles. This is done through the formula 2.1 with some small modifications to compensate for an initial offset at 0 mechanical degrees (eq 3.8). Where θ_n and ψ_n are the mechanical and electrical angle for motor number n, while $offset_n$ is the initial mechanical angle offset found during the startup sequence.

$$\psi_n = (\theta_n + offset_n) * PolePairs \tag{3.8}$$

The module created uses custom data types for ease of implementation and code scalability. A set of arrays with 16-bit STD_logic_vectors has been used for target angles and amplitudes. As for the mechanical and electrical angles converted inside the module, an array of 16-bit unsigned numbers has been used. All arrays have a length equal to the number of motors used.

3.2.10 Top

The top file serves as the primary file containing the FOC core. The purpose of this file is to package, instantiate, connect, and synchronize all the modules used within FOC. The interface to the core is described in table 3.5. Since most of the modules in the core are shared and pipelined, synchronization needs to be performed. A simple counter sends and multiplexes the input signals to their respective module. The counter updates the multiplexers so the correct signal is sent and gathered at the correct time from the modules.

Name	Direction	Type	Description
clk_main, clk_pwm	in	STD_LOGIC	Clock signal for logic and PWM
rst	in	STD_LOGIC	Reset signal
en_override	in	STD_LOGIC	Overrides FOC with angle and amplitude
id_ref, iq_ref	in	i_reference_array*	Reference target for id and iq
th_override	in	override_th*	Target angle when en_override= '1'
amp_override	in	override_amp*	Target amplitude when en_override='1'
i_a, i_b, i_c	in	current_array*	Current measurements
KP_d, KI_d, KP_q, KI_q	in	Signed[15:0]	PI parameters for PI controllers. *_d for i_d, *_q for i_q
null_state_h	out	STD_LOGIC_VECTOR	Wire at position "n" is active when motor "n" is in the high null state
null_state_l	out	STD_LOGIC_VECTOR	Wire at position "n" is active when motor "n" is in the low null state
pwm_pins	out	pwm_pin_config*	PWM signals coming out of the core
i_out	out	dq_array*	Calculated current along the q axis in dq-domain

Table 3.5: List of input and output parameters used when instancing the core. All data types marked with "*" are described in table 3.6.

Data type	Length	Contains
i_reference_array	motors	signed[15:0]
override_th	motors	unsigned[15:0]
override_amp	motors	unsigned[15:0]
angle_array	motors	unsigned[15:0]
current_array	motors	signed[15:0]
pwm_pin_config	motors	STD_LOGIC_VECTOR[5:0]
dq_array	motors	STD_LOGIC_VECTOR[15:0]

Table 3.6: Custom data-types used in the core interface.

A top module diagram has been created to show the data flow throughout the FOC-core (figure 3.11). As seen in table 3.6, the core operates using 16-bit vectors when transferring data between the modules. To ensure the correct data is processed, a series of multiplexers (mux) and de-multiplexers (de-mux) have been utilized during the synchronization, allowing the non-shared logic to function properly.

The shared modules within the FOC system are the Clarke 3.2.2, Park 3.2.3 and inverse Park 3.2.4 transformations, all LUTs used for sinusoidal values 3.2.5, the CORDIC algorithm 3.2.7 used for conversion between a cartesian frame to a polar frame, and the control-unit. As for non-shared logic, a set of two PI controllers 3.2.6 and one SVPWM-module 3.2.8 is initiated per motor in the system.

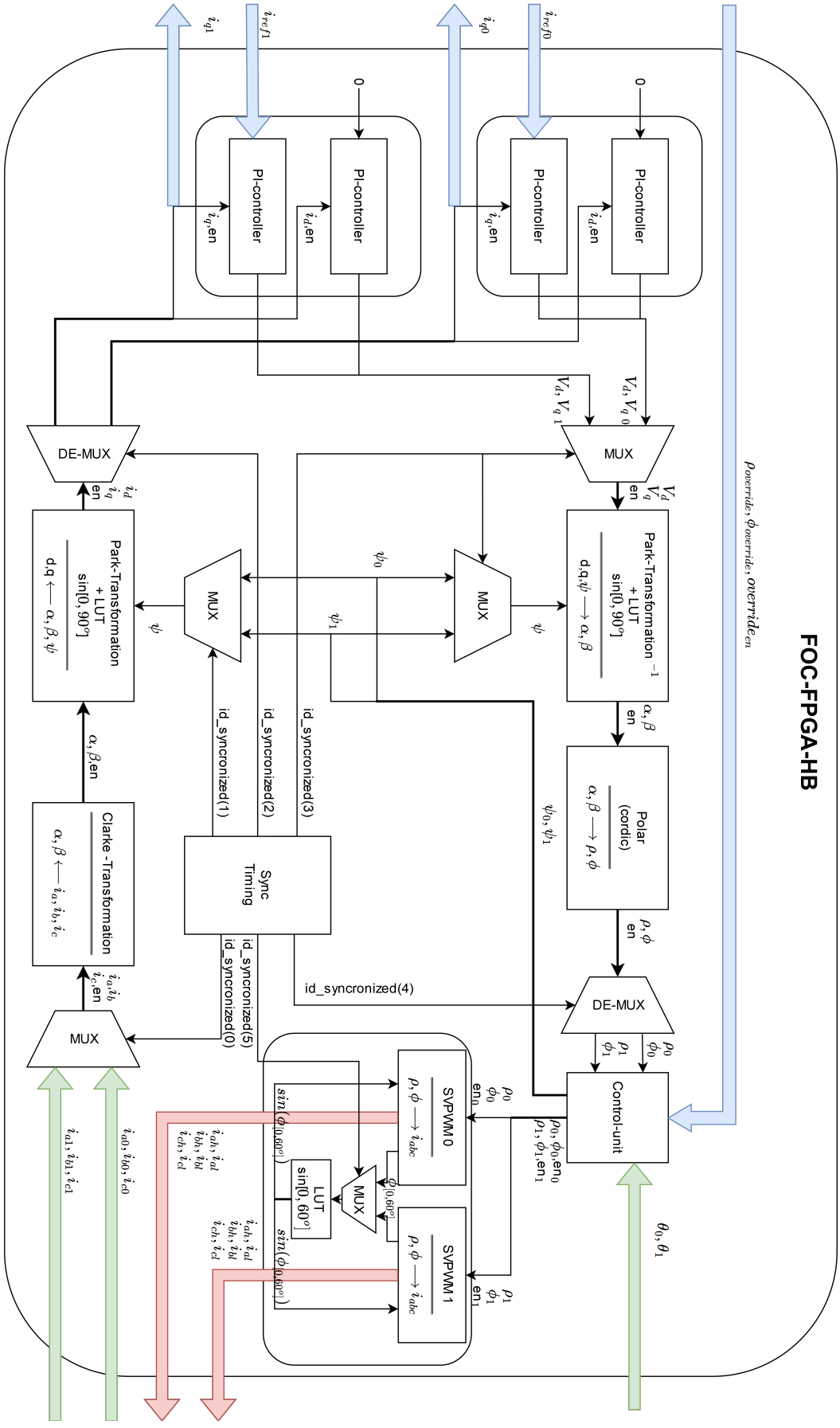


Figure 3.11: A diagram over the FOC core that has been implemented. The symbols used within the diagram are explained in table 3.7

Symbol	Explanation
i_{ref}	Desired current draw
i_q	Current draw along q in d,q - frame
i_d	Current draw along d in d,q - frame
V_q	Voltage along q in d,q - frame
V_d	Voltage along d in d,q - frame
ψ_m	Electrical angle for motor m
α	Position α in α, β - frame
β	Position β in α, β - frame (Electrical)
ρ	Amplitude in polar-coordinate frame
ϕ	Angle in polar-coordinate frame
θ_n	Mechanical angle for motor n
en	Strobed enable signal
$override_{en}$	Enable signal for override. If 1 then $\rho_{override}$ and $\phi_{override}$ is used for control instead of iq
id_syncronized(n)	Syncronization id signal for shared logic. At position n motor m is being calculated
i_{am}	Current in coil a for motor m
i_{bm}	Current in coil b for motor m
i_{cm}	Current in coil c for motor m
i_{ah}, i_{al}	PWM signal for a_{high} and a_{low} output
i_{bh}, i_{bl}	PWM signal for b_{high} and b_{low} output
i_{ch}, i_{cl}	PWM signal for c_{high} and c_{low} output
LUT $\sin[0, x^o]$	LUT containing sinusoidal values for angles between 0 and x^o
Blue arrow in	Data from outer control-system into core for control
Blue arrow out	Data from core to outer control-system
Green arrow	Sensor data
Red arrow	PWM Signals

Table 3.7: Symbols used within the diagram of the FOC core (figure 3.11)

3.3 Compatibility layers

3.3.1 Current sensing

To sense the currents i_a, i_b and i_c in the motors, a SPI interface between the ADC EVAL-AD5592R-PMDZ 3.1.6 and the FPGA 3.1.1 was implemented. The implemented SPI interface runs at 12MHz, but it is possible to run the interface at speeds up to 20 MHz as it is supported by the ADC. Both the Clock Polarity (CPOL) and Clock Phase (CPHA) are user-configurable. However, in the case of the ADC, both have been set to "1", meaning that the clock polarity idles at logical high voltage while the data is sent on the first clock edge of the master clock after the chip has been selected. Having a CPHA value of 0 requires the module to send and receive data immediately after the chip select has been activated. While the chip has been selected one bit is exchanged per clock cycle.

The ADC was set up using an instruction to select all input ports as ADC ports. The following message sent to the ADC was a zero message containing only zeroes, as no read or write to the registers can be done in a short period during the port setup. Thereafter, a message prompting the ADC to start measuring and sending the data was sent. The repeat flag and the pins used for ADC measurements were set to high. The following messages are zero messages used for the exchange of data from the ADC.

A typical set of messages sent during startup can be found in table 3.8.

Message number	MSB 15	Register address 14:11	Flag 10:8	Pin selection 7:0
1	0	0100	000	1111111
2	0	0000	000	0000000
3	0	0010	010	0000111
4	0	0000	000	0000000

Table 3.8: A set of instructions sent from the FPGA to the ADC. The first operation (1) sets the I/O on the ADC/DAC to the ADC configuration. The following message is a no-operation message. Message number 3 sets the starts a sequential readback and measurement on I/O pins. The middle flag signals the ADC to loop the reading of ADC data. Bit 7 downto 0 contains what pins to read the ADC data from.

The selection of what channels to measure is selected by the user with a generic during the instantiation of the module.

The FPGA side of the SPI interface receives a readback message on the format "0" & 3bit channel address & 12-bit ADC result. A typical message for ADC channel 1, with the amplitude of $\frac{255}{4095}V_{ref}$ can be found in table 3.9.

ADC/DAC	Index ADC	Value
15	14:12	11:0
0	001	0000_1111_1111

Table 3.9: Response from ADC readback. Bit 15 is set to "1" if the response contains DAC data, else the response contains ADC data. The 3 index bits contain an integer with the number of the measured pin. The remaining 12 bits contain the amplitude of the measured signal. This number is in the range 0 to 4095 and represents 0 to V_ref volts. The example shows a measured voltage of 255/4095 V_{ref}

The SPI module converts the sequential readback message to a 3-bit index integer and a 12-bit unsigned measurement synchronized to an outgoing enable signal.

3.3.2 Rotary encoder

The encoder module was made to keep track of the motor rotation measured by the rotary encoder 3.1.4. This is done through measuring the phases on channels A and B. Channel A and B have a 90-degree offset from each other. A third channel (I) sends a pulse once a full physical rotation has occurred.

When measuring the rotation, the values for A, B, and I are latched to ensure that no metastability has taken place. This minimizes the possibility of propagation of metastability through the system. Comparing the previous values for A and B with their new values tells us whether or not movement has been made. It also helps determine the direction of the possible movement. The initial position of the encoder is counted as 0 degrees during startup. Any offset between this 0^o and a real-world 0^o angle has to be compensated for outside this module.

During operation, the module counts up/down until channel I first activates. The value of this offset is then stored internally. In subsequent triggers of channel I, the angle is set to this offset. This is done as an extra layer of protection against drifting or errors during counting.

Since the combination of wheel and encoder 3.1.4 provides an effective resolution of 20000 points per full revolution, the output has been scaled to 16 bits[3].

3.3.3 Top level FPGA

To combine the sensor data with the core a module called top_fpga was constructed. This module sets up the core and connects the modules sensing the physical location and currents to it. This compatibility layer is made to keep track of where the currents measured by the ADC are measured and stored. This is done by discarding any measurements done in non-zero PWM periods. An internal array is used for storing the measurements, and the respective values are sent to the FOC-core 3.2. The normalization and extension of bits for the measured currents are done by subtracting 0x800 from the measurement and extending the top bits with the bit indexed at position 11 in the result from the subtraction. The masks used within the ADC SPI modules 3.3.1, as well as the ADC instances, are contained within this file. Modifications of the physical hardware outside the FPGA would require modifications in this file.

3.3.4 Block Design

The SoC on the Trenz FPGA 3.1.1 was used to transfer data to and from an external computer through Ethernet. Therefore a block design was used to add the ARM processor to the design. The ARM processor was added through a ZYNQ7 Processing System (5.5) block. Communication between the RTL code and the ZYNQ7 Processing System was done through an AXI Interconnect (2.1) block and AXI GPIO (2.0) blocks. The signals that did not require connection to the SOC were routed out using external ports.

[3]Scaling is done through multiplying with 839, then right-shifting the number with 8, effectively dividing with 2^8.

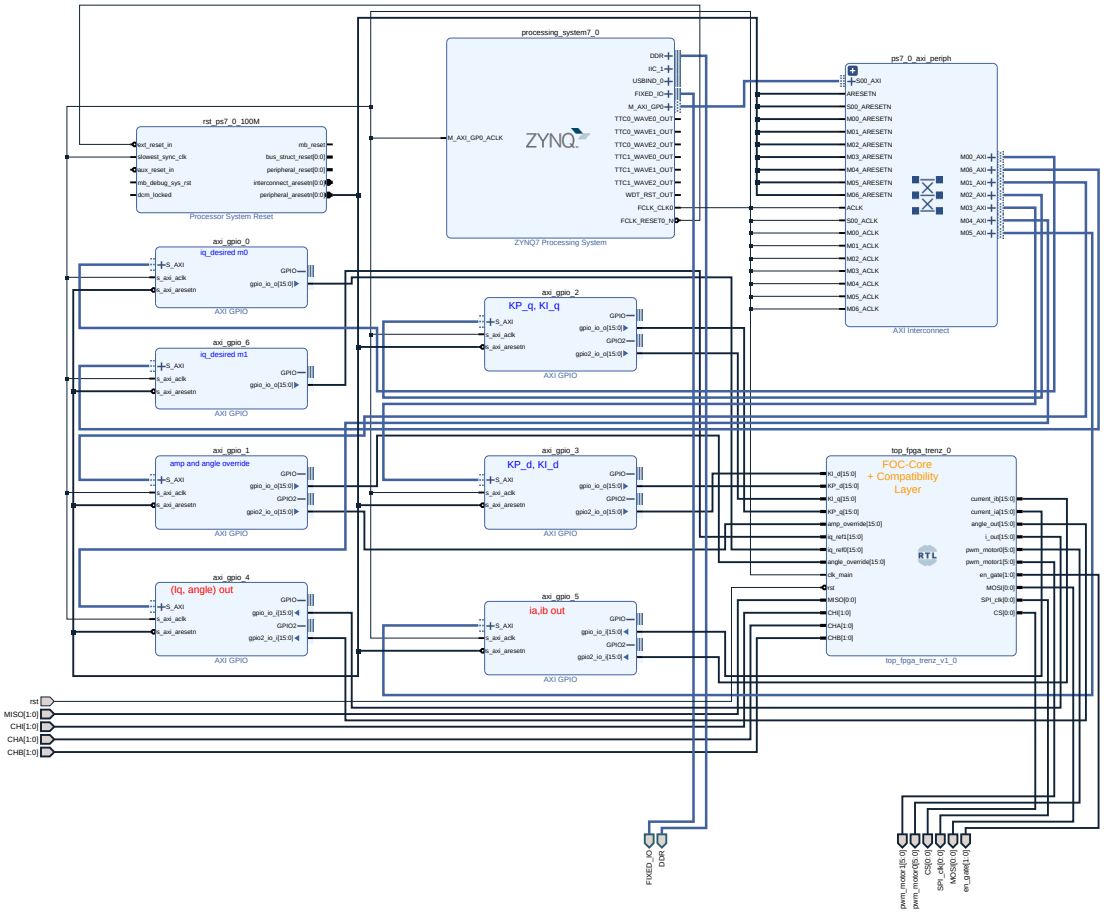


Figure 3.12: Block diagram of the ZYNQ7 Processing System combined with the compatibility layer 3.3.3. The diagram is set up for 2 motors.

3.4 Software

In the complete system, the Arm dual-core Cortex-A9 MPCore was programmed using Vitis [54]. The Vitis project was based upon the lwIP[15] Echo Server template combined with the GitHub project "Zynq_UDP" [30]. The main file "main.c" was changed to record data within the core and transmitted through 1024-byte UDP packages 3.11. A UDP package was sent once the package was full or an internal counter (TcpSlowTmrFlag) was triggered.

Byte number	1	2	3	4	5	6	7	8	9
Message	motor_number	angle[15:8]	angle[7:0]	I_q[15:8]	I_q[7:0]	I_a[15:8]	I_a[7:0]	I_b[15:8]	I_b[7:0]

Table 3.10: The table shows how a single message within the UDP package is constructed. The message is packaged within a UDP package 3.11

Byte number	1	2-10	11-19	...	991-999	1000-1024
UDP pkg	Number of messages	Message 1	Message 2	...	Last Message	Zeros

Table 3.11: The setup of a full UDP message sent with lwIP. The data within the messages are shown in 3.10

As for receiving packages, the file "echo.c" was modified to handle and decipher incoming messages. The received UDP packages 3.13 consisted of the number of operations in the message and the operation headers with payload to publish on the AXI GPIO pins.

Operation	Header[39:32]	Payload [31:16]	Payload[15:0]	Description
No_{op}	0	0	0	No operation
Override control	1	ϕ	ρ	Angle and Voltage amplitude when overriding the core
Unused/No_{op}	2	Na	Na	Not in use
Tune PI i_q	3	Kp	Ki	Kp and Ki values for i_q PI controller
Tune PI i_d	4	Kp	Ki	Kp and Ki values for i_d PI controller
Set i_q target motor 0	5	i_q ref	i_d ref	Target current motor 0
Set i_q target motor 1	6	i_q ref	i_d ref	Target current motor 1
Unused/set i_q target motor n	5+n	i_q ref	i_d ref	Target current motor n. Not in use
Reset/No_{op}	255	Na	Na	Reset. Not in use

Table 3.12: The table shows the buildup of an operation message received within a UDP package (table 3.13). Each Operation consists of 5 bytes. The first byte contains the header, while the following four bytes contain a Payload split into two 16-bit numbers.

Byte number	1	2-6	7-11	...	1017-1021	1021-1024
UDP pkg	Number of Operations	Operation 1	Operation 2	...	Operation 240	Zeros

Table 3.13: The table shows the buildup of a UDP message received by the FPGA. The package contains a maximum of 240 operation messages that can be sent at once. The first byte tells the system the number of operations sent within the message. Operations not included in the first-byte number are ignored. The buildup of the operation message is shown in table 3.12.

The Software was only used to verify and test the implementation, as the main focus of this thesis is the scalability and use of pipelining in the core.

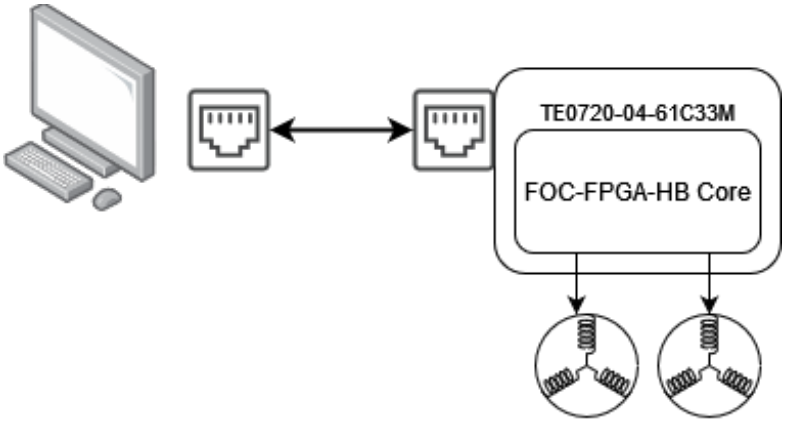


Figure 3.13: Simple image of computer connected to the Trenz TE0720 3.1.1 communicating over Ethernet via UDP packages (table 3.13 and 3.11). The SoC on the FPGA board was solely utilized for sending and receiving UDP packets. In future implementations, the SoC may be utilized in the control system or for advanced operations.

Chapter 4

Experiments & Results

The following two sections will contain the experiments 4.1 and the Results 4.2. The Experiments section provides an overview of the desired measurements and how the respective measurements were obtained. The Results section 4.2 will contain 3 different types of plots, a table, and four images. The first three plots show the usage of DSP, LUT, and registers through the implementations. The following plots are for resource consumption and are divided into "Shared" vs "Non-Shared" logic. The table shows the IO usage, and the last graph shows a real-world example of the implementation resource usage on the Trenz FPGA 3.1.1. The images are diagrams of the resource placement on the Trenz FPGA chip for different motor configurations.

4.1 Experiments

This section will contain an overview and explanation of the experiments conducted to produce the results listed in the Results section 4.2.

The experiments conducted in this thesis consisted of creating a pipelined core for FOC. The core was then mapped to physical IO pins on an FPGA to constrain the design into valid real-world implementations. After an implementation in Vivado had been created, a series of reports were created: Power, Utilization, and Timing reports. An image of the implementation on the device was taken. For each subsequent implementation, the number of motors controlled by the FOC was increased by one. An increase by one motor meant an additional 16-bit AXI GPIO interface needed to be added to control the i_q parameter. The IO required by the motor was also mapped to physical pins. The workflow of the experiments followed a cycle containing the steps 1-8:

1. Edit the top compatibility layer to a set number of motors.
2. Add additional interface for ADC if necessary
3. Edit the constraints file to include IO mapping for all physical IO pins.
4. Edit the Block design in Vivado to include the updated design and add AXI GPIO interfaces for i_q targets.
5. Generate implementation in Vivado.
6. Generate and export reports for the implemented design.
7. Take a picture/screenshot of the graphical interface showing the design placed on the device.
8. Go to 1 and repeat the steps with more motors.

The steps were repeated until all 150 external IO ports in the Trenz FPGA 3.1.1 were used.

The implementations for 1 and 2 motors were tested in the real world to ensure they functioned correctly. The real-world sanity test consisted of programming the SoC and FPGA and connecting an external computer to send instructions. The external computer sent instructions for setting K_p=3000, $K_i = 0$, and $i_q = 200$ to the SoC. Note that the K_p, K_i values used in the PI controller have not been tuned, nor should they be used in a final product. The sanity test was conducted by visually inspecting the motor's rotation and sending the values i_q= -200 and $i_q = 400$ to the core. Both motors were tested simultaneously at different velocities/currents. The velocities were limited by internal friction and current draw. Implementations with quantities of motors exceeding two have not been tested due to time and availability limitations. The Utilization report was exported to spreadsheet (.xlsx) files for analyzing the consumption of slice LUTs, Slice Registers, and DSPs. The timing report was only used to verify that the logic could run at 100MHz, which was specified in the timing constraints. The exact number of IO pins mapped in the implementation can be seen in table 4.1 in the Results.

4.2 Results

In this section, a series of results from the implementation of a pipelined FOC-Core will be presented. The graphs are created through the python3 library Matplotlib [36]. The core can run at the targeted 100MHz. Further timing analysis is not included as Vivado 2.2 may do some timing optimization beyond reaching closure. "The Vivado Design Suite routes global resources first, such as clocks, resets, I/O, and other dedicated resources. This default priority is built into the Vivado router. The router then prioritizes data signals according to timing criticality. " [46].

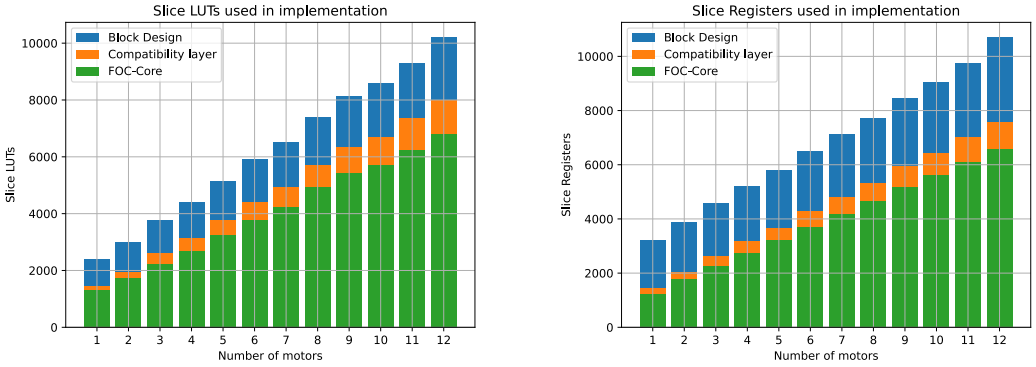
The total height of the bar plots is the total resource usage within the mentioned design. The Block Design mentioned in figure 4.1a, 4.1b and 4.2a shows the resources used for AXI interfaces and SoC. The compatibility layer consists of the components mentioned in 3.3 such as the conversion by a rotary encoder 3.3.2, an ADC SPI interface 3.3.1 and the conversion and mapping of currents.

"**Shared logic**" in plot 4.2b, 4.3a and 4.3b contains modules: Clarke 3.2.2, Park 3.2.3, Inverse Park 3.2.4, Cordic 3.2.7, Control unit 3.2.9, all the Sinusoidal LUTs 3.2.5 and the FOC-Top module 3.2.10. The "**Non-Shared logic**" contains the SVPWM modules 3.2.8 and the PI controllers 3.2.6 as these modules needed 1 and 2 instantiations per motor in the system. The total of Shared + Non-Shared logic is contained within the FOC-Core (Green in plot 4.1a, 4.1b and 4.2a).

Figure 4.4 illustrates the resource usage in the Trenz FPGA 3.1.1, reported as a percentage. A usage of 100% indicates the utilization of all available resources within the specified resource type. The FPGA contains 53200 slice-LUTs, 106400 Slice registers, 220 DSPs, and 150 IO ports.

The table 4.1 shows the calculation of physical IO pins utilized. For each motor, a total of 10 pins are allocated, comprising 6 for PWM, 3 for Encoder, and 1 for *en_gate*. The increment in total ADC pins follows the formula $ceil(\frac{Motors*3}{8}) * 4$. This is because the SPI protocol uses 4 pins, and every ADC 3.1.6 measures up to 8 channels, whereas 3 channels (i_a, i_b, i_c) are required per motor.

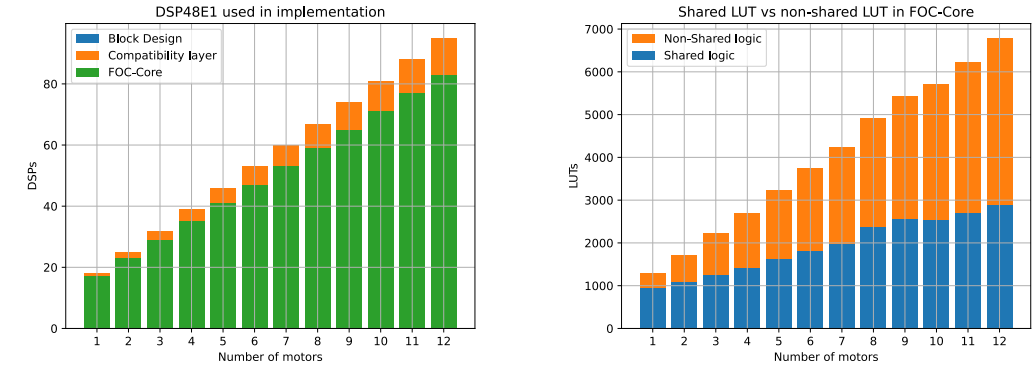
At the final page of this chapter a total of four images are provided 4.5 and 4.6. The images show implementations of 1, 4, 8, and 12-axis FOC-cores placed on the Trenz FPGA 3.1.1. The images are included to indicate how the resources are mapped in the FPGA fabric when the system's motors are increased.



(a) Slice LUTs used in implementation of FOC

(b) Slice Registers used in implementation of FOC

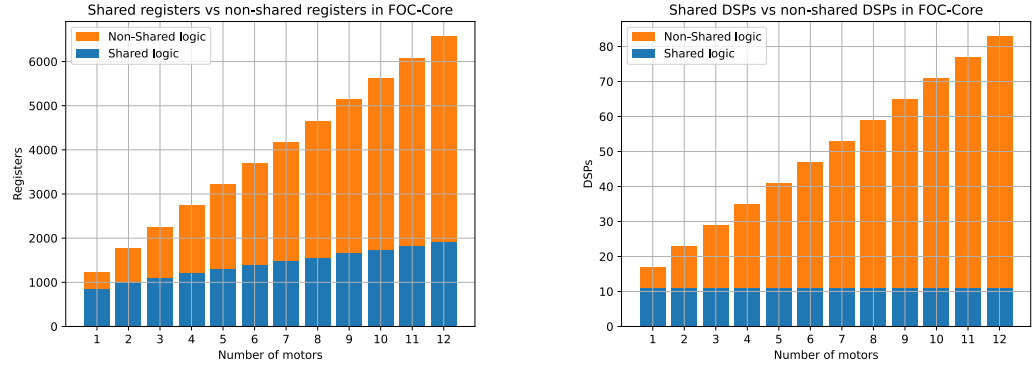
Figure 4.1: Resource usage in implementation



(a) DSP48E1 usage in implementation of FOC

(b) LUT usage in FOC-Core is divided into shared and Non-Shared logic.

Figure 4.2: DSP usage in implementation (Left), Shared vs. Non-shared logic LUT usage (Right)



(a) Register usage in FOC-Core from shared logic vs non-shared logic

(b) DSP usage in FOC-Core from shared logic vs non-shared logic

Figure 4.3: Shared vs Non-shared logic.

Motors	PWM	θ Encoder	en_gate	ADC	RST	Total Pins	% usage of IO
1	6	3	1	4	1	15	10%
2	12	6	2	4	1	25	16.67%
3	18	9	3	8	1	39	26%
4	24	12	4	8	1	49	32.67%
5	30	15	5	8	1	59	39.33%
6	36	18	6	12	1	73	48.67%
7	42	21	7	12	1	83	55.33%
8	48	24	8	12	1	93	62%
9	54	27	9	16	1	107	71.33%
10	60	30	10	16	1	117	78%
11	66	33	11	20	1	131	87.33%
12	72	36	12	20	1	141	94%

Table 4.1: Total usage Physical IO pins on Trenz FPGA 3.1.1. The table shows the number of pins and their purpose given a number of motors. The maximum number of Physical IO pins on the Trenz FPGA is 150.

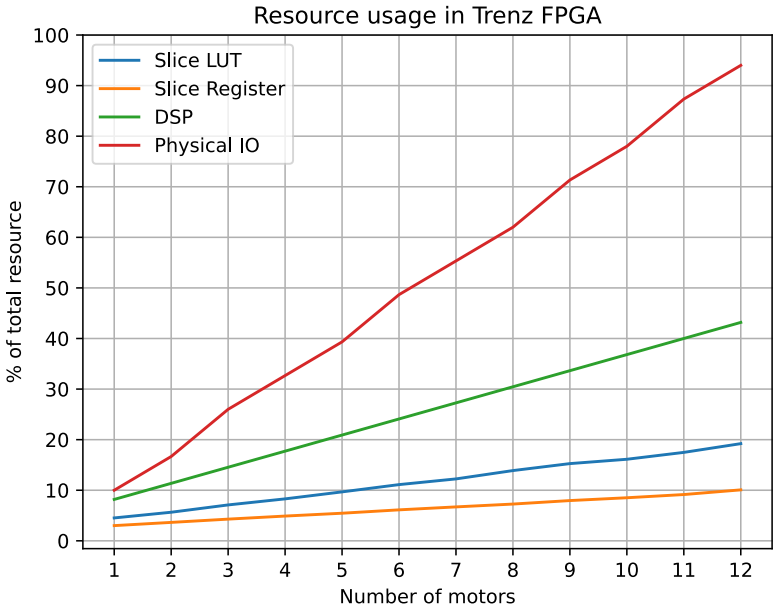


Figure 4.4: Quantity of resources consumed by the design on the Trenz FPGA 3.1.1. The quantity is given as a percentage. The total number of resources is described at the beginning of the Results section 4.2

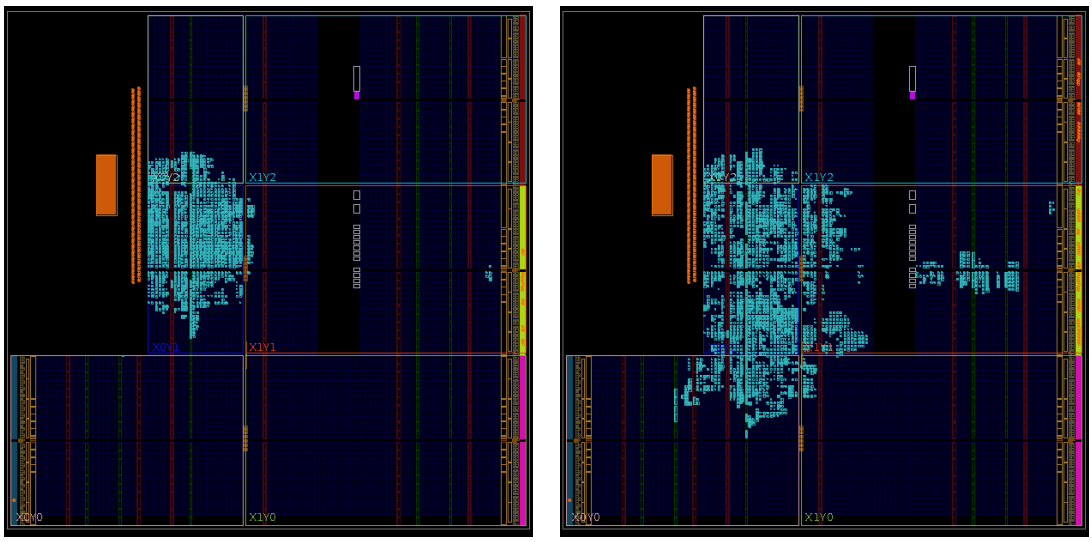


Figure 4.5: Diagram over the physical usage of FPGA generated by Vivado. The implementations are for 1 motor (left) and 4 motors (right)

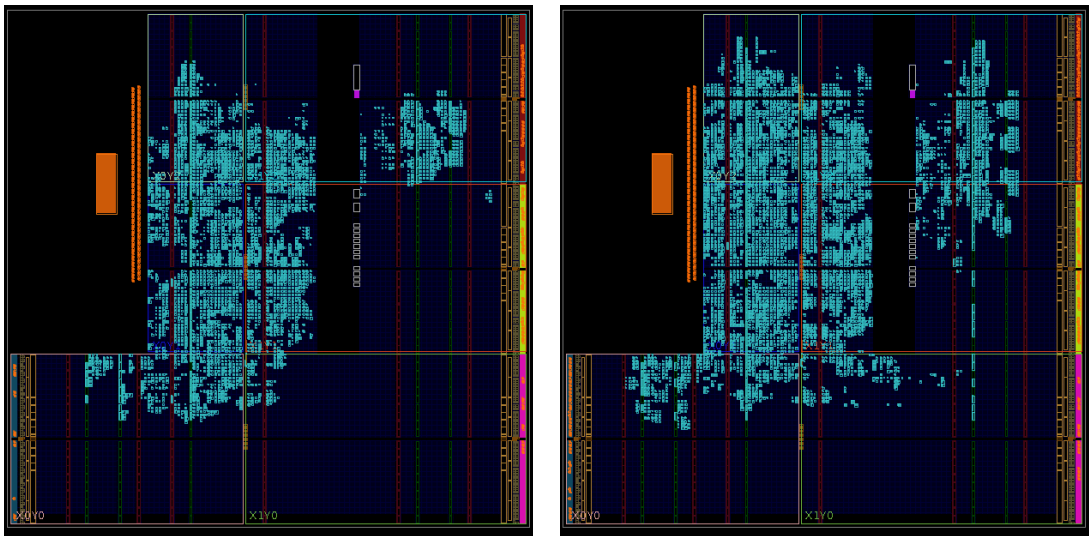


Figure 4.6: Diagram over the physical usage of FPGA generated by Vivado. The implementations are for 8 motors (left) and 12 motors (right)

Chapter 5

Discussion

This chapter is reserved for discussing and examining the results and graphs obtained in the results section 4.2 through the experiments described in 4.1. One of the main focuses throughout the discussion will be on the resource consumption of the FOC-Core 3.2. The limiting factor of the current system if placed on the Trenz FPGA 3.1.1 will be discussed. As the functionalities and implementation methodologies may vary between FOC-core implementations, the core will be compared to itself in a single motor configuration.

5.1 Analysis

The graph 4.1a, 4.1b and 4.2a shows a linear relationship between the addition of motors and the resource consumption. When looking at the previously mentioned graph, it is important to notice how the FOC-Core scales between the number of motors in the system. The cost of the Block design and Compatibility layer would remain if the FOC core were to be replaced with n numbers of cores. If we look at the graph displaying the Slice LUT usage 4.1a, we can see that the amount used for the implementation supporting 1 motor is approximately 1300 LUTs (1299). Comparing this to the usage for 2 motors (1723) gives an increase of 424 LUTs between the two implemented designs. The average increase of slice LUTs between a number of motors n, and the following number n+1 is ≈ 505.6[1]. However, if a new full implementation of the core 3.2 were to be added for each motor added to the system, the resource usage would increase with $\approx$ 1300 LUTs per motor. Comparing an implementation with multiple cores driving one motor each to the proposed system, a decrease of $\frac{1300-505}{1300} * 100\% \approx 61\%$ in resource

[1]Calculated using linear regression and Least Squares method through [43]

consumption can be seen.

This trend continues when looking at slice Registers, where the implementation of 1 motor has the initial cost of 1224 registers, and an average increase in size of 482.5[1] registers per motor. Comparing the pipelined approach with the standard multiplication of cores, gives a reduction of $\frac{1224-482}{1224} * 100\% \approx 60.6\%$ in register consumption for our pipelined approach.

Moving over to DSP usage within the core, we can see an increase of 6 DSPs per additional motor in the system and an initial cost of 17 DSPs for a system with one motor. Even though it could be argued that a reduction of $\frac{17-6}{17} * 100\% \approx 64.7\%$ in DSP consumption can be found, the natural opposing argument as to why this is unrealistic is that removing the pipeline from the implemented modules like 3.2.2, 3.2.3, 3.2.4 and 3.2.7 could optimize out some of the initial architectural cost.

When analyzing the resource consumption of the core, it is important to notice what contributions originate from the shared logic, compared to the non-shared modules, such as the PI controllers 3.2.6 and the SVPWM 3.2.8 module. In the Shared LUT vs. non-shared LUT graph 4.2b, it can be seen that the non-shared logic has a huge impact on the resource consumption within the FOC-core 3.2.10. For an implementation of 1 motor, the so-called "shared" logic takes up ≈72% of the total resource cost. However, when controlling 5 motors, about half the LUT usage in the core originates from the controllers and SVPWM modules, while the other half originates from the shared logic and synchronization.

The same trend can be seen in figure 4.3a. It is therefore evident that the cost of shared logic will be overshadowed by the cost of non-shared logic for more extensive systems. The implementation with 1 motor starts with a consumption of 69% of the used registers. Moving over to an implementation with 3 motors, the non-shared logic has surpassed the initial cost and is now consuming more resources. As for the last plotted implementation with 12 motors, the shared logic takes up less than 30% of the total FOC-core cost, while the non-shared modules use the rest.

Moving over to the DSP usage in the FOC-core, it can be seen that the shared logic has a fixed cost when it comes to the DSPs (fig 4.3b). The Shared logic uses a total of 11 DSPs while the increase in DSPs is 6 DSPs per motor[2]. This means an implementation

[2]The cost of the 6 DSPs originates from 2 DSPs per PI controller 3.2.6 = 4 DSPs, and 2 DSPs for the SVPWM module 3.2.8.

with 2 motors or more will have most of the DSP usage within the non-shared logic. When looking at an implementation with 12 motors, it will have ≈87% of its DSP usage contained within the controllers and SVPWM modules alone.

The previously mentioned results from graph 4.2b, 4.3a and 4.3b may suggest that implementing a shared controller and trying to optimize the SVPWM module to share logic could yield better results in figure 4.1a, 4.1b, 4.2a and a smaller core when controlling multiple motors. Moving over to an implementation using one pipelined controller, placing the results in one array could save resources in terms of DSP cost.

The graph labeled 4.4 shows the percentage of resources consumed by the FOC system[3] on the Trenz TE0720 FPGA 3.1.1. From this graph, the limits of the total system can be extracted. As seen in the aforementioned graph 4.4, the Physical IO ports exhibit a notably accelerated ascent compared to LUTs, registers, and DSPs. The limitation of maximum resource utilization is placed at 12 motors. Here, the design consumes 94% of all the available IO pins. A theoretical increase of one motor would yield 151 IO pins, which is 100.67% of the total resource. When it comes to other resources consumed by the implementation, the DSPs use ≈ 43% of the 220 DSPs in the FPGA, while a total of ≈ 20% of the total LUTs and ≈ 10% of total registers have been utilized in the implementation with 12 motors. The images of the implemented design 4.5 and 4.6 give a visual representation of the total resource consumption of FPGA fabric for 1, 4, 8 and 12 motor implementations.

Using multiple cores instead of one single core scaling with the number of motors could be constrained by the DSP usage, as the approximate usage of DSPs for 1 motor is ≈ 8.2%. This would bring an implementation controlling 12 motors with one core for each up to ≈ 98.2% usage. A 98.2% DSP usage would lead to little to no extra resources for more complex models, system expansions, or operational monitoring on the Trenz FPGA. Another problem arises when approaching a 100% utilization of an internal resource. The timing in Vivado may become hard or impossible to pass. Achieving timing closure will become harder as more effort would need to be applied during routing of the design between resources [46].

A reduction in IO consumption could be achieved by, for example, using the 3-PWM

[3]The system includes FOC-core3.2 and the compatibility layer 3.3 (FPGA top layer 3.3.3, AXI GPIO connections and the Block design 3.3.4)

mode (table 3.3) in the driver board 3.1.3 and optimizing the SPI interface. This would increase the number of motors driven by the proposed system, as the IO is the current limiting factor, and not the internal resources. Optimizing the IO for the motors could reduce IO consumption from 6+4+1 down to 3+4+0 per motor when both the 3-PWM mode and a global enable signal are used for all the motors. To reduce the in and outgoing SPI signals for the ADCs, 3 of the 4 wires can be shared in the interface. Either by sharing everything except the Chip select signal or through breaking the SPI convention and share everything except the MISO signal[4].

5.2 Possible Core replacements

Although an FPGA implementation has its benefits, implementing it through CPU software may suit some use cases better. Often, the implementations have been done on micro-controllers as they are relatively cheap, and many implementations can be found on the Internet like Arduino SimpleFOC [23][48] or through Github [8]. "However, the processing demands and determinism required for motor control makes it difficult to schedule other processing tasks in the remaining clock cycles. Optimising code at assembly level is often an additional challenge"[40]. Therefore, using an FPGA has benefits, like determinism in latency and timing.

Another contender is a combined CPU and FPGA system that may use the CPU to calculate the sinusoidal values used within the FOC-core 2.6, 2.9. However, using the processing system might introduce some delay or remove the deterministic timing in the FPGA, as we now need to await a response from the CPU.

The last contender is an integrated circuit with FOC. One example of this is the AMT49406 [10] by Allegro. Solutions like this exist for easy implementation of FOC for motors. Unfortunately, standalone chips like these offer little to no access to the internal components used within the FOC. They also do not allow for modifications in the hardware after the chip has been created. Therefore, an FPGA implementation may be preferred when researching the inner workings of motor control.

[4]The CS, MOSI, and clock signal will be shared. Meanwhile, the incoming MISO signal will have separate wires from each ADC to the master. This is possible as the Slaves synchronize to the master.

5.3 Use in Robotic manipulator and quadrupedal walkers

As the proposed pipelining of an FOC-core can control 12 motors or more[5] on the Trenz FPGA, it could be used for a 12 degrees of freedom (DOF) quadrupedal robot like the Open Dynamic Robot Initiative (ODR) [28]. Another benefit of this implementation running purely on the FPGA fabric is that the Cortex A9 arm core within the Trenz FPGA 3.1.1 can be used for higher-level functions and behavior control of the robot. The low utilization of LUTs (20%) and Registers (10%) and the moderate use of 43% DSPs for 12 motors 4.4 creates an opportunity to handle additional tasks in the FPGA fabric like network handling or video filtering.

[5]To control more than 12 motors, it would be necessary to implement a more efficient IO utilization scheme or acquire hardware with additional IO ports

Chapter 6

Conclusion

6.1 Summary

This thesis explored the benefits of scaling a single FOC-core during multi-axis motor control on FPGA. The scalability of the system was achieved through pipelining and sharing of logic between the motors. An implementation of such a system was constructed 3.2 and tested through the generation of 12 Vivado 2.2 implementations 4.2.

During this exploration, some of the potential limits when scaling an FOC system on an FPGA were found. The results showed a clear linearity between the number of motors included in the system and the utilization of resources 4.4. During the discussion 5, it was shown that scaling one single core for n motors used fewer resources than using n cores for the same purpose. This research indicates that pipelining an FOC core may be suitable in research applications with multi-DOF robots.

6.2 Answering the research questions

A set of research questions was constructed in section 1.2. It is now time to answer those questions as the work is being concluded.

6.2.1 Research question 1

> **1.2.1** When scaling an FOC-core through pipelining, how is the resource consumption affected by the addition of multiple motors?

In the results section 4.2, the figures 4.2b, 4.3a and 4.3b give clear indications that the pipelining of a FOC-core will scale linearly when adding additional motors in the

system. The graphs indicate that the transitional cost from n motors to n+1 motors is lower than that of a system with 1 motor. However, a system like the one proposed in this thesis has some overhead as signals must be synchronized to the pipeline. The discussion 5 shows that an average increase of 505.6 LUTs, 482.5 registers, and 6 DSPs can be seen when adding additional motors to the system. However, some of this cost can be omitted by sharing controllers and optimizing the SVPWM module 3.2.8.

6.2.2 Research question 2

1.2.2 When is the proposed architecture capable of outperforming an n-number duplication of a single core in terms of resource usage on FPGA?

The pipelined architecture uses fewer LUTs, registers, and DSPs when multi-axis motor control is desired. As seen in graph 4.1a, 4.1b, and 4.2a for any number n>1, it is more efficient to use the pipelining within the core for multiple motors instead of adding additional FOC-cores to the FPGA. Nevertheless, as elaborated upon in the discussion 5, eliminating the pipelining within the FOC-core modules can shift the threshold at which the multiplication of FOC-cores becomes more advantageous than pipelining.

Another resource to consider when implementing the design is the timing within the core. If we look at the time delay within the core, it might be beneficial to introduce more modules to shorten the longest signal path. In this design, with a clock frequency of 100MHz, no critical timing errors within the FOC- modules were found. However, moving over to faster clock speeds, the design may be harder to synthesize.

6.2.3 Research question 3

1.2.3 What are the limiting factors for FOC on multiple motors with FPGA?

As for our last research question, it can be answered by looking at the graph 4.4. Here, it is possible to conclude that the limiting factor using the hardware described in 3.1 is the IO connections. In the hypothetical scenario where the number of IO connections can be doubled, the IO would still be the limit. The DSP limit, which in this case is the next constraint that could potentially be reached, can be utilized for an implementation with a total of 34 motors. This is close to three times the motors controlled when reaching the IO limit. However, utilizing one FOC-core per motor would lead to reaching the DSP limit when operating 12 motors, as discussed in section 5.1.

Even though the IO limits us to 12 motors, control of motors exceeding a number of 12 may introduce problems outside the FPGA. Some of the limits introduced are latency in wires, voltage drops, signal degradation, signal noise, and complex wiring. To limit the effect of the previously mentioned challenges, the use of multiple microcontrollers or FPGAs may be beneficial. For some implementations with many motors in a confined space, like quadrupedal robots and octopod walkers, the use of a single FPGA can be viable. However, if the distances get bigger or in high-powered applications, the previously mentioned challenges might come into play.

6.3 Main contributions

This thesis presents an approach to effectively control multiple synchronous BLDC motors. The following contributions have been made to advance the field of motor control:

- Creation of a modular and customizable FOC-core suited for simultaneous control of multiple BLDC motors.
- Gathered and presented data regarding resource consumption of LUTs, Registers, and DSPs in a pipelined FOC-core.
- Optimization through pipelining of FOC-core so FPGA resources can be utilized on precision models or the addition of functionalities beyond FOC.

6.4 Future Work

Research into motor control on FPGA is tightly coupled with robotics. Ergo, the continuation might include robotics, the use of electric manipulators, or the field of multi-axis motor control. As for the research in this thesis, some of the most noteworthy areas to continue the work are:

Physical implementation in the real-world: A promising direction for future work would be to deploy the proposed FOC-core on a physical robot, such as the quadrupedal ODR [28], to assess its performance and feasibility in real-world scenarios.

To use the proposed core on the ODR one would need to modify the configuration file to fit the current setup. A complete rewrite of the top level with its corresponding features, including the communication with external computers would also be necessary.

Further optimization of the FOC-core: The proposed FOC-core has reduced scalability when it comes to the implementation of non-shared logic[1]. Further research into optimizing and minimizing resource consumption by said logic may introduce improvements in the complexity and capabilities of the system.

A logical initial step is to pipeline the controllers to share the computational logic while the results are stored in registers. As for optimization techniques for the SVPWM module, the calculation can be shared, while the counters for the PWM signals can be set as non-shared logic. The synchronization within the core may also need adjustments to accommodate the controller's timing.

Compare PWM strategies: Further research on different PWM strategies in FOC is warranted. A comparison of SPWM, SVPWM, and a hybrid system combining both is an intriguing topic of interest. This research is motivated by the recognition that applying the maximum torque is not always necessary, presenting an opportunity to explore optimal PWM strategies. These strategies can be implemented using generics when programming the core or in real-time to select the most suitable PWM strategy dynamically. In the case of a real-time

[1]The "**Non-shared logic**" refers to the modules SVPWM 3.2.8 and PI-controllers 3.2.6

implementation, it would be worthwhile to investigate the potential of including both strategies within the core or explore the feasibility of utilizing partial reconfiguration [55] as an alternative approach.

Augment the capabilities: The proposed system has a relatively low chip consumption of resources in terms of LUT, Registers, and DSPs. This allows for the allocation of remaining resources towards enhancing the system's capabilities and accommodating more sophisticated modeling of behaviors, such as implementing "Finite-Set Model predictive Current Control" [58].

To introduce controllers or external logic to enhance the capabilities of the system, the core module might need modifications to accommodate the addition or replacement of logic. However, if the FPGA were to accommodate external systems like image processing or behavioral control, little to no modifications[2] would be necessary in the FOC-Core.

[2]Except modifications in the configuration file.

Appendix A

PWM state figures

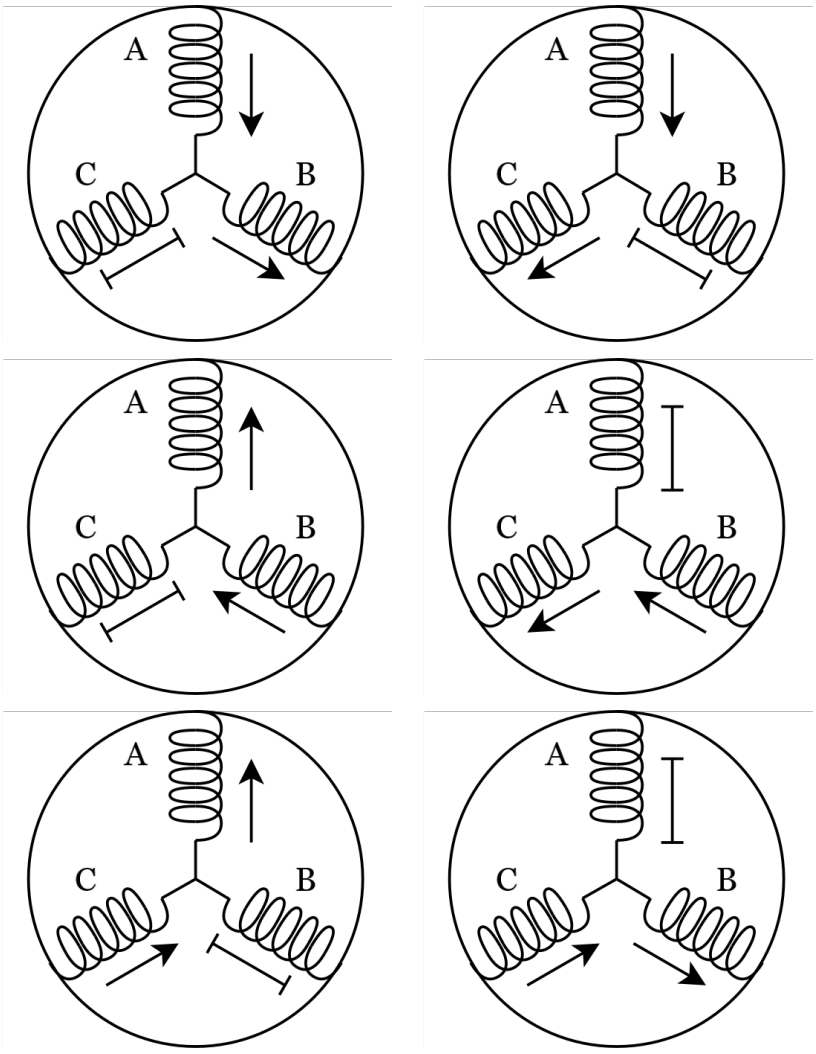


Figure A.1: The active power states used in SPWM. The arrows show the direction of the current.

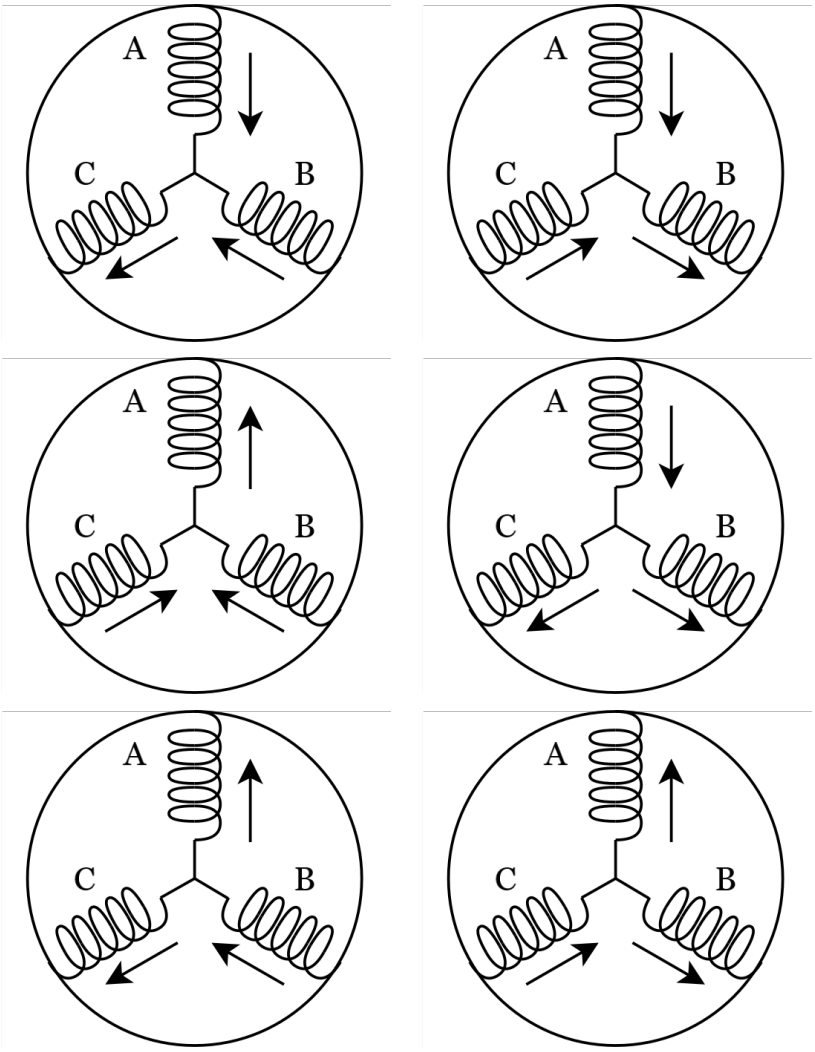


Figure A.2: The active power states used in SVPWM. The arrows show the direction of the current.

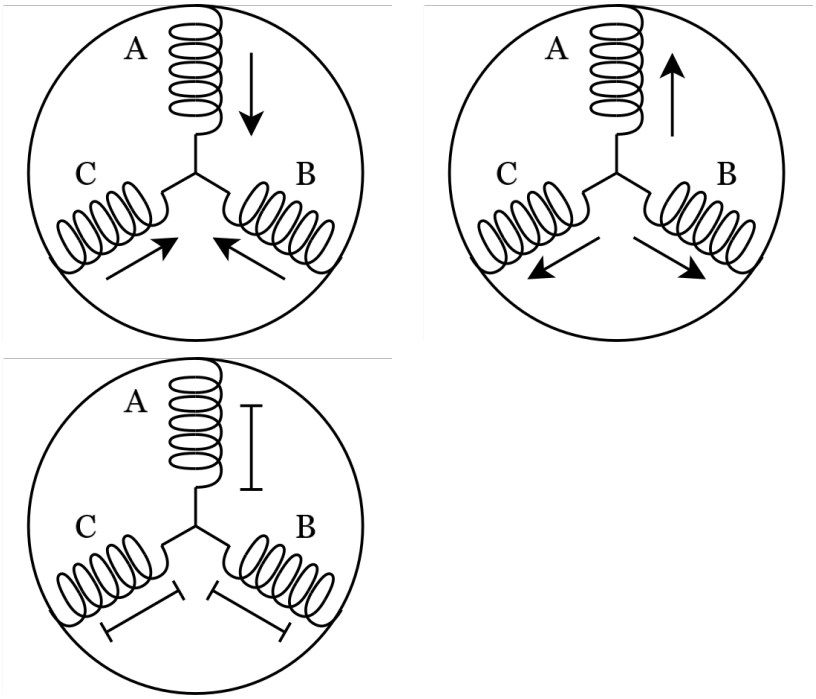


Figure A.3: Powerstates used for no power draw in PWM modulation. There exist 18 more zero states that are not shown here. Their configuration consists of 1 / 2 coils having the same polarity and the last 2 / 1 coils being not connected. The arrows show the direction of the current.

The entirety of the code encompassing the FOC core and its compatibility layers is available in the GitHub repository, accessible at [4].

Bibliography

[1] *61599 - Vivado Implementation - Discussion of tool repeatability*. URL: https://support.xilinx.com/s/article/61599?language=en_US (visited on 23/04/2024).

[2] *About XDC Constraints • Vivado Design Suite User Guide: Using Constraints (UG903) • Reader • AMD Technical Information Portal*. URL: https://docs.amd.com/r/en-US/ug903-vivado-using-constraints/About-XDC-Constraints (visited on 29/04/2024).

[3] *AD5592R Datasheet and Product Info | Analog Devices*. URL: https://www.analog.com/en/products/ad5592r.html (visited on 02/04/2024).

[4] Oscar H. Bjørnstad. *oscarhbj/FOC-FPGA-HB*. original-date: 2024-05-03T11:36:30Z. May 2024. URL: https://github.com/oscarhbj/FOC-FPGA-HB (visited on 03/05/2024).

[5] *BOOSTXL-DRV8305EVM Evaluation board | TI.com*. URL: https://www.ti.com/tool/BOOSTXL-DRV8305EVM (visited on 28/03/2024).

[6] Daniel Brandt. *CAS - CERN Accelerator School: Digital Signal Processing: Sigtuna, Sweden 31 May - 9 Jun 2007. CAS - CERN Accelerator School: Course on Digital Signal Processing*. Tech. rep. Backup Publisher: CERN. Geneva: CERN, 2008. DOI: 10.5170/CERN-2008-003. URL: https://cds.cern.ch/record/1003726 (visited on 02/05/2024).

[7] *Brushless DC electric motor*. en. Page Version ID: 1216355029. Mar. 2024. URL: https://en.wikipedia.org/w/index.php?title=Brushless_DC_electric_motor&oldid=1216355029 (visited on 01/05/2024).

[8] *Build software better, together*. en. URL: https://github.com (visited on 22/04/2024).

[9] *ChatGPT*. en-US. URL: https://chat.openai.com (visited on 25/04/2024).

[10] *Code-Free FOC Sensorless BLDC Motor Controller - AMT49406 | Allegro MicroSystems*. URL: https://www.allegromicro.com/en/products/motor-drivers/bldc-drivers/amt49406 (visited on 24/04/2024).

[11] *Counter-electromotive force*. en. Page Version ID: 1163624870. July 2023. URL: https://en.wikipedia.org/w/index.php?title=Counter-electromotive_force&oldid=1163624870 (visited on 01/05/2024).

[12] *CSD18540Q5B data sheet, product information and support | TI.com*. URL: https://www.ti.com/product/CSD18540Q5B (visited on 29/03/2024).

[13] *DRV8305 data sheet, product information and support | TI.com*. URL: https://www.ti.com/product/DRV8305 (visited on 29/03/2024).

[14] W. C. Duesterhoeft, Max W. Schulz and Edith Clarke. 'Determination of Instantaneous Currents and Voltages by Means of Alpha, Beta, and Zero Components'. In: *Transactions of the American Institute of Electrical Engineers* 70.2 (July 1951). Conference Name: Transactions of the American Institute of Electrical Engineers, pp. 1248–1255. ISSN: 2330-9431. DOI: 10.1109/T-AIEE.1951.5060554. URL: https://ieeexplore.ieee.org/document/5060554 (visited on 29/02/2024).

[15] Adam Dunkels and Leon Woestenberg. *lwIP: Overview*. URL: https://www.nongnu.org/lwip/2_1_x/index.html (visited on 12/04/2024).

[16] Pierre Duysinx and Michel Geradin. 'AN INTRODUCTION TO ROBOTICS: MECHANICAL ASPECTS'. In: (). URL: https://www.researchgate.net/publication/268424577_AN_INTRODUCTION_TO_ROBOTICS_MECHANICAL_ASPECTS.

[17] *Dynamical systems - Latest research and news | Nature*. URL: https://www.nature.com/subjects/dynamical-systems (visited on 20/03/2024).

[18] Boston Dynamics. *Manufacturing*. en-US. URL: https://bostondynamics.com/industry/manufacturing/ (visited on 26/04/2024).

[19] Boston Dynamics. *Spot*. en-US. URL: https://bostondynamics.com/products/spot/ (visited on 26/04/2024).

[20] Renesas Electronics. *BLDC Motor Control Algorithms | Renesas*. en. URL: https://www.renesas.com/us/en/key-technologies/motor-control-robotics/bldc-motor-control-algorithms (visited on 26/04/2024).

[21] Joakim Eriksson and Luciano Hermandsen. *Rapid prototyping -development and evaluation of Field Oriented Control using LabView FPGA*. Jan. 2011. URL: https://www.diva-portal.org/smash/get/diva2:392940/FULLTEXT01.pdf (visited on 25/04/2024).

[22] *EVAL-AD5592R-PMDZ Evaluation Board | Analog Devices*. URL: https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad5592r-pmdz.html#eb-overview (visited on 02/04/2024).

[23] *Field Oriented Control*. en-US. URL: https://docs.simplefoc.com/foc_theory (visited on 22/04/2024).

[24] *Field-programmable gate array*. en. Page Version ID: 1220658191. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=Field-programmable_gate_array&oldid=1220658191 (visited on 26/04/2024).

[25] Trenz Electronic GmbH. *TE0720-04-61C33MAS Starter Kit*. en. URL: https://shop.trenz-electronic.de/en/TE0720-04-61C33MAS-TE0720-04-61C33MAS-Starter-Kit (visited on 26/03/2024).

[26] Damond Goodwin. *Field-oriented Control (Vector Control) for Brushless DC Motors - Technical Articles*. en. Mar. 2023. URL: https://control.com/technical-articles/field-oriented-control-vector-control-for-brushless-dc-motors/ (visited on 29/04/2024).

[27] *GPT UiO*. URL: https://gpt.uio.no/ (visited on 25/04/2024).

[28] Felix Grimminger et al. 'An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research'. In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 3650–3657. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.2976639. URL: https://ieeexplore.ieee.org/document/9015985/ (visited on 21/04/2024).

[29] Ajay Gunalan. *Ajay Gunalan -Basics of a Motor*. en. URL: https://ajaygunalan.github.io/blog/notes/motor/motor (visited on 01/05/2024).

[30] Del Hatch. *delhatch/Zynq_UDP*. original-date: 2018-05-20T22:57:54Z. Jan. 2024. URL: https://github.com/delhatch/Zynq_UDP (visited on 12/04/2024).

[31] *https://no.mouser.com/datasheet/2/678/V02-4924EN_DS-AEDT-981x-2018-01-31-909402.pdf*. URL: https://no.mouser.com/datasheet/2/678/V02-4924EN_DS-AEDT-981x-2018-01-31-909402.pdf (visited on 02/04/2024).

[32] Inductiveload. *English: A schematic of a en:CORDIC, of the bit-parallel, unrolled variety.* May 2009. URL: https://commons.wikimedia.org/wiki/File:CORDIC_(Bit-Parallel,_Unrolled,_Circular_Rotation).svg (visited on 13/03/2024).

[33] Adafruit Industries. *8-channel Bi-directional Logic Level Converter*. en-US. URL: https://www.adafruit.com/product/395 (visited on 02/04/2024).

[34] *Instruction pipelining*. en. Page Version ID: 1220412214. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=Instruction_pipelining&oldid=1220412214 (visited on 26/04/2024).

[35] Mahak Jain. *Differences between Black Box Testing vs White Box Testing*. en-US. Section: Software Engineering. Dec. 2018. URL: https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/ (visited on 29/04/2024).

[36] *Matplotlib — Visualization with Python*. URL: https://matplotlib.org/ (visited on 17/04/2024).

[37] Stuart McFarlane. *LabVIEW FPGA Programming: Pros and Cons*. en-US. URL: https://www.viewpointusa.com/ie/ar/labview-fpga-the-good-the-bad-and-the-ugly/ (visited on 25/04/2024).

[38] *MN4004 Antigravity Type 4-6S UAV Motor KV400 - 2PCS/SET_Antigravity Type_Motors_UAV Power_T-MOTOR Official Store - UAV Power System, Robot Power System, Model Power System*. URL: https://store.tmotor.com/goods-439-MN4004++Antigravity+Type+4-6S+UAV+Motor+KV400+-+2PCSSET.html (visited on 27/03/2024).

[39] *MOSFET*. en. Page Version ID: 1219529887. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=MOSFET&oldid=1219529887 (visited on 29/04/2024).

[40] *Off-loading the complexity of motor control – how intelligent peripherals simplify Field-Oriented-Control (FOC) implementations*. URL: https://toshiba.semicon-storage.com/content/dam/toshiba-ss-v3/emea/en_gb/semiconductor/design-development/reference-model/TCM0456A_ENG.pdf (visited on 22/04/2024).

[41] R. H. Park. 'Two-reaction theory of synchronous machines generalized method of analysis-part I'. In: *Transactions of the American Institute of Electrical Engineers* 48.3 (July 1929). Conference Name: Transactions of the American Institute of Electrical Engineers, pp. 716–727. ISSN: 2330-9431. DOI: 10.1109/T-AIEE.1929.5055275. URL: https://ieeexplore.ieee.org/document/5055275 (visited on 11/03/2024).

[42] *Pulse-width modulation*. en. Page Version ID: 1208971714. Feb. 2024. URL: https://en.wikipedia.org/w/index.php?title=Pulse-width_modulation&oldid=1208971714 (visited on 29/04/2024).

[43] *Quick Linear Regression Calculator*. URL: https://www.socscistatistics.com/tests/regression/default.aspx (visited on 20/04/2024).

[44] Niranjana R. *FPGA-Based Robotics and Automation*. en-US. Section: Blog. Aug. 2023. URL: https://fpgainsights.com/blog/fpga-based-robotics-and-automation/ (visited on 25/04/2024).

[45] *Robotics and Intelligent Systems (ROBIN) - Department of Informatics*. en. URL: https://www.mn.uio.no/ifi/english/research/groups/robin/index.html (visited on 02/04/2024).

[46] *Routing Priorities • Vivado Design Suite User Guide: Implementation (UG904) • Reader • AMD Technical Information Portal*. URL: https://docs.amd.com/r/2020.2-English/ug904-vivado-implementation/Routing-Priorities (visited on 25/04/2024).

[47] Kim Sang-Hoon. *Sinusoidal Pulse Width Modulation - an overview | ScienceDirect Topics*. 2017. URL: https://www.sciencedirect.com/topics/engineering/sinusoidal-pulse-width-modulation (visited on 29/04/2024).

[48] Antun Skuric et al. 'SimpleFOC: A Field Oriented Control (FOC) Library for Controlling Brushless Direct Current (BLDC) and Stepper Motors'. en. In: *Journal of Open Source Software* 7.74 (June 2022), p. 4232. ISSN: 2475-9066. DOI: 10.21105/joss.04232. URL: https://joss.theoj.org/papers/10.21105/joss.04232 (visited on 27/04/2024).

[49] Yngve Solbakken. *Space Vector PWM Intro*. en-US. May 2017. URL: https://www.switchcraft.org/learning/2017/3/15/space-vector-pwm-intro (visited on 29/04/2024).

[50] *SPS-9600 | Manson*. URL: https://www.manson.com.hk/product/sps-9600/ (visited on 02/04/2024).

[51] Burak Tufekci et al. 'Efficient FPGA Implementation of Field Oriented Control for 3-Phase Machine Drives'. In: *2020 IEEE East-West Design & Test Symposium (EWDTS)*. ISSN: 2472-761X. Sept. 2020, pp. 1–5. DOI: 10.1109/EWDTS50664.2020.9224884. URL: https://ieeexplore.ieee.org/document/9224884 (visited on 26/03/2024).

[52] *TXB0108 data sheet, product information and support | TI.com*. URL: https://www.ti.com/product/TXB0108 (visited on 02/04/2024).

[53] *Vector control (motor)*. en. Page Version ID: 1188966450. Dec. 2023. URL: https://en.wikipedia.org/w/index.php?title=Vector_control_(motor)&oldid=1188966450 (visited on 29/04/2024).

[54] *Vitis Unified Software Platform*. en. URL: https://www.xilinx.com/products/design-tools/vitis.html (visited on 13/03/2024).

[55] *Vivado Design Suite User Guide Partial Reconfiguration*. URL: https://docs.amd.com/v/u/2019.1-English/ug909-vivado-partial-reconfiguration (visited on 24/04/2024).

[56] Jack Volder. 'The CORDIC Computing Technique'. English. In: IEEE Computer Society, Dec. 1959, pp. 257–257. DOI: 10.1109/AFIPS.1959.57. URL: https://www.computer.org/csdl/proceedings-article/afips/1959/50540257/12OmNqN6QXX (visited on 13/02/2024).

[57] Zishen Wan et al. 'A Survey of FPGA-Based Robotic Computing'. In: *IEEE Circuits and Systems Magazine* 21.2 (2021). Conference Name: IEEE Circuits and Systems Magazine, pp. 48–74. ISSN: 1558-0830. DOI: 10.1109/MCAS.2021.3071609. URL: https://ieeexplore.ieee.org/abstract/document/9439435 (visited on 26/03/2024).

[58] Sebastian Wendel, Armin Dietz and Ralph Kennel. 'FPGA based finite-set model predictive current control for small PMSM drives with efficient resource streaming'. en. In: *2017 IEEE International Symposium on Predictive Control of Electrical Drives and Power Electronics (PRECEDE)*. Pilsen, Czech Republic: IEEE, Sept. 2017, pp. 66–71. ISBN: 978-1-5386-0507-3. DOI: 10.1109/PRECEDE.2017.8071270. URL: http://ieeexplore.ieee.org/document/8071270/ (visited on 24/04/2024).

[59] *What is NI LabVIEW? Graphical Programming for Test & Measurement*. en. URL: https://www.ni.com/en/shop/labview.html (visited on 25/04/2024).