University of Southern Denmark
Department of Mechanical and Electrical Engineering

Control of PMAC Motor for an Electric Go-Kart

Report by
Jacob Damgaard Iversen
jaive17 - 13/11/96

Jakob Møller Lorenzen
jalor17 - 02/05/96

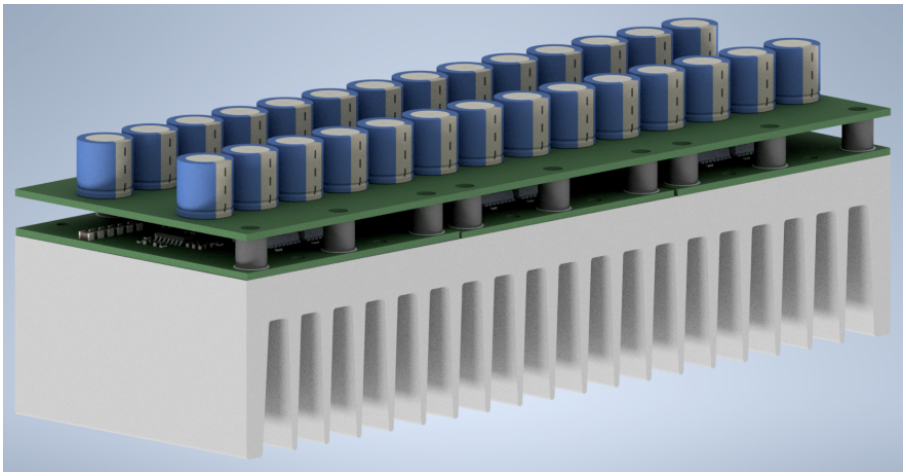
Sebastian Rud Madsen
semad17 - 11/07/97

Supervisor
Rakesh Ramachandran

Course Code
EK-PRO1

Project Period
01/02-2021 to 26/05-2021

Date of Hand-in
25/05-2021



Jacob Damgaard Iversen

Jakob Møller Lorenzen

Sebastian Rud Madsen

1 Abstract

This project revolves around the design of a controller for a brushless, permanent magnet motor. This controller should replace the existing motor controller in an electric go-kart and must therefore fit into the already existing system.

The paper goes through the design, thermal considerations, and simulation of a three-phased inverter with parallel MOSFETs, the design of the related DC-link, and an analog interface board, which handles the sensor signals from the go-kart.

The inverter is digitally controlled with field-oriented control using space vector modulation. A simulation with Clarke-Park transformations and PI controllers designed with pole-placement is performed to verify the digital control.

The control and processing of sensor and encoder signals uses a Zybo board and is implemented in programmable logic and processing system using VHDL and C, respectively. Communication with a PC is established to set and get relevant parameters of the control.

The project results in a controller designed for a continuous RMS current of 80 A, and simulations show that when running at a temporary peak RMS current of 220 A, the temperature is kept below 75°C. The assembled motor controller is, due to time constraints and limited laboratory access, only partly tested. The tests show that each component of the controller is working individually, but the controller still has to be verified during various load scenarios and when inserted into the go-kart.

2 Preface

This report is part of the output of the semester project on the first semester of the Master in Electronics at the Technical Faculty of the University of Southern Denmark. The product is designed to fit seamlessly within the current go-kart system. The report documents design choices for the product, many being based on compatibility with the go-kart.

The report is written for an audience with fundamental knowledge of electronics, VHDL, and C programming acquired throughout the undergraduate courses and first semester of the Master.

Thanks to Rakesh Ramachandran for supervising the project.

3 Reading Guide

The report is split into eight parts. The first of which is an introduction regarding the problem statement, project delimitation and a description of the system, upon which the project is based. The second part is concerned with the hardware of the project, namely the inverter, DC-link, and analog interface board. The third part revolves around the control and simulations of the system and the fourth part describes the embedded system, which consists of both VHDL and C programming. The fifth part is comprised of the testing of the different parts of the system. Part six and seven contains the discussion and conclusion respectively, followed by the appendices in part eight.

Abbreviations are written in parenthesis after the phrase they abbreviate, after which the abbreviation is used interchangeably. A list of all abbreviations can be found on page 58. Symbols are written in italic and units are written in roman. The symbols are explained when first used, and a list of symbols is found on page 59.

Figures, tables, and listings are numbered in order of introduction and equations are numbered with the section number and order of introduction in said section.

Literature is referred to as [number of reference, page number], and the bibliography can be found on page 60.

All software, simulations, KiCad files, 3D renderings, and datasheets can be found on the project's GitHub repository at https://github.com/SR-Madsen/ElectricGoKart or in the ZIP file.

Table of Contents

Part I
Introduction

Brushless, permanent magnet (PMAC) motors have seen an increase in popularity in the automotive industry, as they are more desirable than the brushed counterpart. Compared to brushed motors, the PMAC motors have increased torque, efficiency, robustness as well as lower noise and EMI generation. This makes the PMAC motors more appealing when dealing with for example electric vehicles. However, the commutation of the PMAC motors is more complicated, which necessitates an electric motor controller. The controller is used as interface between a DC battery source and the stator of the motor, where an AC current is needed to generate a rotating magnetic field, which then drives the rotor.

4 Problem Statement

The goal of the project is to design a motor controller for a PMAC motor to replace the current Sevcon Gen4 controller in SDU's electric go-kart. The project has several requirements. The controller must fit within the current go-kart, requiring no changes to the external system. The controller may consist of an interface board for low-current signals, and should contain an inverter board with transistors for high-current switching. The Zybo board must be used for program execution and communication with a PC. The overall project costs must not exceed 3000 DKK.

5 Project Delimitation

In order to keep the project manageable within the time scope of the project, the project extent has been limited. The motor controller should only be able to drive the motor in the forward direction, and the solution does not need to support regenerative breaking, traction control, torque vectoring, or other complex functionalities. Furthermore, an interface board with isolated auxiliary supplies and digital connections to be used between the controller and the go-kart is part of the materials for the project. However, the interface board does not include interfacing of analog signals.

6 System Overview

A simplified system block diagram can be seen in Figure 1, where the green block outlines the contents of this project. The coming subsections will describe the system in the electric go-kart, which the motor controller must fit seamlessly within when replacing the Sevcon Gen4. A full overview of the current go-kart can be found in Appendix B.

The system contains fuses for both the motor and controller supply of 160 A and 5 A respectively. The controller supply is passed through two switches, which can be disengaged in an emergency and thus shut down the system. The motor supply is passed through a relay, which is activated and deactivated by the controller. A circuit ensures that the motor can only be activated when the battery voltage is within acceptable range.

In order for the go-kart to move, the drive switch and foot switch must also be enabled, after which the torque pedal sensor is used for setting motor torque.

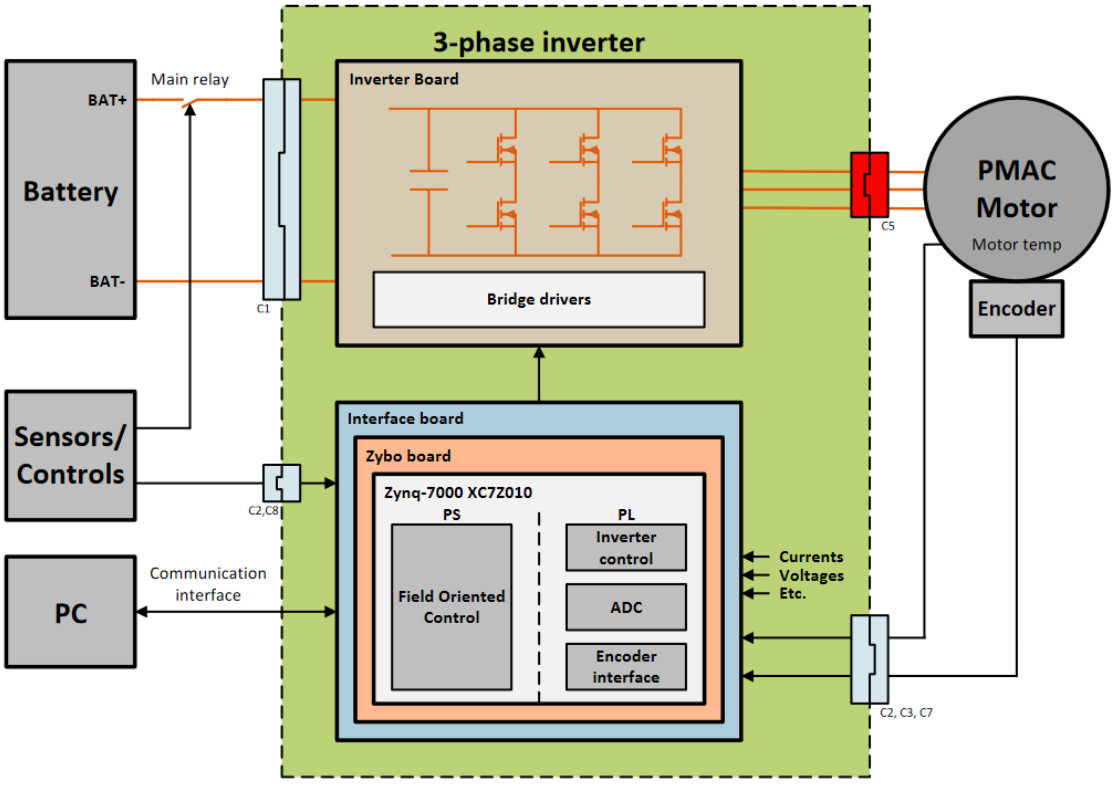


Figure 1: The simplified project diagram. The green block outlines the contents of this project. From project description

6.1 Motor and Battery

The go-kart uses the ME1117 Permanent Magnet AC (PMAC) motor from Motenergy, which is supplied with a voltage from 40 V to 57.6 V when used for endurance, nominally 52.8 V, from a LiFePO4 battery pack. The motor characteristics are listed in Table 1, as provided on the manufacturer's web-page [1].

Characteristic	Value
Pole pairs	4
Voltage rating	0 - 72 VDC
Continuous current	80 A RMS
Peak current	220 A RMS (1 minute)
Peak rotational speed	5000 RPM
Torque constant / Back-EMF constant	0.13 Nm/A or V/(rad/s)
Phase-to-phase resistance	0.013 Ω
Phase-to-phase inductance	0.1 mH
Armature inertia	52 kg/cm^2

Table 1: Characteristics of the used go-kart motor ME1117

6.2 Encoder

Mounted on the motor is an encoder of type RMB28MD from the manufacturer RLS. This encoder requires a 5 V input supply, and has multiple outputs. For acquiring the absolute position, a serial interface is used. This takes a clock with a maximum frequency of 900 kHz in, and outputs 8 bits of data specifying the position. When running, it is also possible to use incremental, digital outputs A, B, and Z with a resolution of 64 pulses per revolution. A timing diagram from the RMB28 datasheet [2, p. 9] showing the incremental outputs can be seen in Figure 2. The Z impulse is high for one count when a full revolution has been completed.

The figure also shows the equation for calculating the resolution for one cycle. For the RMB28MD encoder with 256 counts per revolution, this equation results in a cycle of

$$\text{cycle} = \frac{360° \cdot 4}{256} = 5.625° \tag{6.1}$$

Due to the four pole pairs, the resolution per count for the electrical angle is equal to the resolution per cycle for the mechanical angle. For this reason, any smaller variation than 5.625° of the electrical angle cannot be measured.

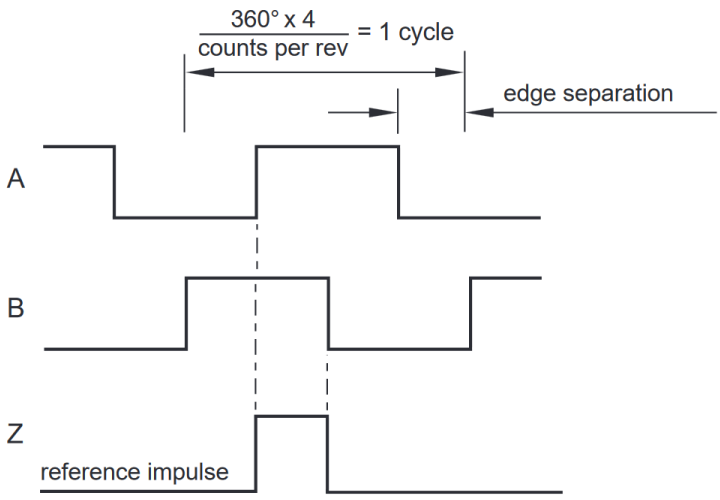


Figure 2: Timing diagram showing incremental outputs A, B, and Z when rotating the encoder, from encoder datasheet [2, p. 9]

7 Zybo Development Board

The core processing part of the motor controller must be the Zybo development board. This board is based around the XC7Z010-1CLG400C, also known as Zynq-7010, which is a dual-core Cortex-A9 Microprocessor (MPU) and Field-Programmable Gate Array (FPGA) on one die. In the Zynq, these are called Processing System (PS) and Programmable Logic (PL) respectively, and use Advanced eXtensible Interface (AXI) ports for interconnection. The Zynq-7010 is a powerful core capable of running complex algorithms and can be used for many types of connections.

The Zybo board also contains memory, serial flash, a programmer, and multiple other features. Some connections have not been routed to modules on the development board, but are connected to expansion ports named Pmod. These allow the Zybo to connect to other boards. The ports contain four miXed-signal Analog-to-Digital Converter (XADC) connections, eight low-speed and 24 high-speed digital connections to the PL, and eight digital connections to the PS as described in the Zybo reference manual [3, p. 25].

Part II
Hardware

The PCBs to be used for controlling the motor must be designed to meet specifications and fit within the system. This includes the inverter itself with transistors and gate drivers, a DC-link with capacitors for preventing transients, and an analog board to handle and filter analog signals.

8 Switching Frequency

When designing the hardware, a switching frequency must be used for design of inverter and filters. While a higher switching frequency results in lower requirements for the filters as well as lower ripple, it increases the transistor losses and processor workload. The frequency is found by using the encoder's resolution of 256 counts per revolution. This ensures that the duty cycles can be changed for every registered movement, with only a calculation delay. The switching frequency f_{sw} must thus be 256 times higher than the motor's peak frequency of 5000 RPM given by Table 1, which results in a frequency of

$$f_{sw,opt} = \frac{5000\,\text{RPM}}{60\,\text{s}} \cdot 256\,\text{cpr} = 21.33\,\text{kHz} \tag{8.1}$$

To reduce switching losses and thus transistor temperature, this frequency is halved to $f_{sw} = 10.66\,\text{kHz}$. This increases the average delay for changes to the duty cycle, which is not expected to impact the mechanical system noticeably, but should be verified by simulation.

9 Inverter

To facilitate the conversion of the DC battery voltage into AC voltage and current that can drive the PMAC motor, a three-phase inverter must be designed. The three-phase inverter consists of three half-bridges connected across the battery voltage as shown in Figure 3. This section will be concerned with choosing the inverter transistors and designing the circuits driving the transistors.

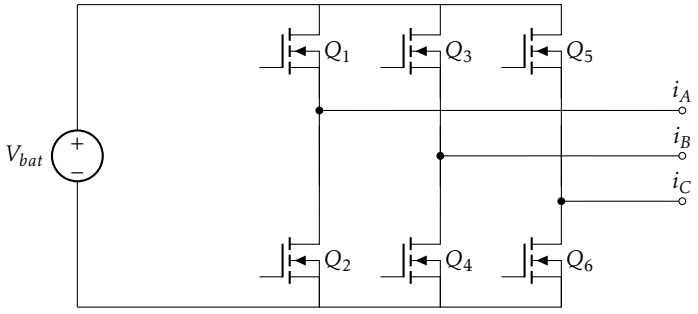


Figure 3: Basic diagram of three-phase inverter

9.1 Transistors

The transistors should be chosen with respect to the needs for voltages and currents within the system, and the losses must be analyzed to ensure a good performance.

9.1.1 Transistor Selection

Often, motor inverters are implemented with either MOSFETs or IGBTs. The pros and cons of each type is listed in this Infineon article [4]. The IGBT is typically chosen for its ability to block high voltages of more than 1000 V and for its ability to handle large currents. However, it suffers from a current tail that makes it unsuitable for switching frequencies above 20 kHz. Furthermore, the IGBT has a negative temperature coefficient, which introduces the risk of thermal runaway and makes it harder to parallel devices.

On the other hand, the MOSFET eliminates some of these downsides by having a positive temperature coefficient and by having good switching capabilities. This allows the inverter to run at faster switching frequencies and to utilize parallel devices. Conversely, the MOSFET is usually not capable of blocking voltages much above 1000 V.

The battery voltage of the car has a maximum value 57.6 V for endurance usage. Based on this voltage, it is chosen to utilize MOSFETs as the switching transistors in the inverter, as this means the great blocking capabilities of the IGBTs will not be necessary. This choice allows for easier paralleling of devices, consequently yielding lower on-resistances and in turn lower conduction losses. The MOSFETs should have a voltage rating of at least 80 V to allow for overshoots during switching, and they should be capable of handling the maximum allowable motor current. This current is, as seen in Table 1, 220 A RMS or 311 A peak. Because of the large current it is beneficial to choose a MOSFET with a very low drain-source on-resistance, $R_{ds,on}$. However, generally when the MOSFET's $R_{ds,on}$ is lowered, its switching capabilities are reduced, resulting in larger switching losses. This drawback must be kept in mind, although switching losses are generally outweighed by conduction losses at 10.66 kHz.

With this in mind, the IAUT300N08S5N012 MOSFET [5] from Infineon is selected. The most important specifications of this MOSFET are summarised in Table 2.

Parameter	Value	Unit
V_{ds}	80	V
I_D	300	A
$R_{ds,on}$ (80°C)	1.3	mΩ
Q_{gd}	56	nC
t_{rise}	19	ns
t_{fall}	55	ns
R_{jc}	0.4	$\frac{K}{W}$
$V_{gs,th}$	3.8	V

Table 2: Specifications of the chosen MOSFET

9.1.2 MOSFET Loss

To verify that the chosen MOSFET is indeed suited for the application, and to dimension a cooling system, it is necessary to estimate the power loss in the MOSFETs and in the inverter. This is done by using the worst-case driving conditions. The power loss will therefore be estimated based on a drain current of $I_{phase,RMS} = 220\,\text{A}$, even though this current is only expected for up to 1 minute at a time. The loss calculations will also take paralleling of the MOSFETs into account, as it is expected that this will be necessary in order to keep the junction temperatures of the MOSFETs at suitable levels.

9.1.2.1 Conduction Loss

The conduction loss in a MOSFET can be calculated using the following equation,

$$P_{con} = R_{ds,on} \cdot I_D^2 \tag{9.1}$$

where I_D is the drain current and $R_{ds,on}$ is the drain-source on-resistance.

As the output of the inverter is AC current the drain current varies with time. Furthermore, the output current flows complimentary through each side of every half-bridge. During the positive half-cycle, the high-side switch conducts, and during the negative half-cycle, the low-side FET conducts. Assuming that the PWM for each FET is symmetrical, it can be assumed that the current through each FET, I_D, is equal to the RMS value of one half-cycle of phase current. Meaning $I_D = \frac{I_{phase,RMS}}{\sqrt{2}}$. Furthermore, the conduction loss is scaled by the number of FETs in parallel squared. This is one of the most significant upsides to paralleling MOSFETs, and is described in the Texas Instruments application note of [6, p. 9]. Applying these modifications to Equation 9.1 yields:

$$P_{con,fet} = \frac{R_{ds,on} \cdot \frac{I_{phase,RMS}}{\sqrt{2}}^2}{N^2} \tag{9.2}$$

where N is the number of MOSFETs paralleled.

Using Equation 9.2 inserting $I_{phase,RMS} = 220\,\text{A}$ and $R_{ds,on} = 1.3\,\text{m}\Omega$ and solving for multiple FETs in parallel yields the results shown in Figure 4. This figure also shows the accumulated conduction loss in all FETs in the inverter to underline the benefit of paralleling the MOSFETs.

9.1.2.2 Switching Loss

Estimating the switching loss in a MOSFET is a complicated matter as it depends on many factors. The switching loss is essentially a result of the period in which both voltage across the MOSFET and current through the MOSFET is present. Switching scenarios where this is the case is called hard-switching. When voltage and current is not present simultaneously the MOSFET is said to be soft-switching.

When current is running from the motor inductor into the half-bridge, freewheeling in the high-side body diode, and the low-side FET is turned on, the current will start

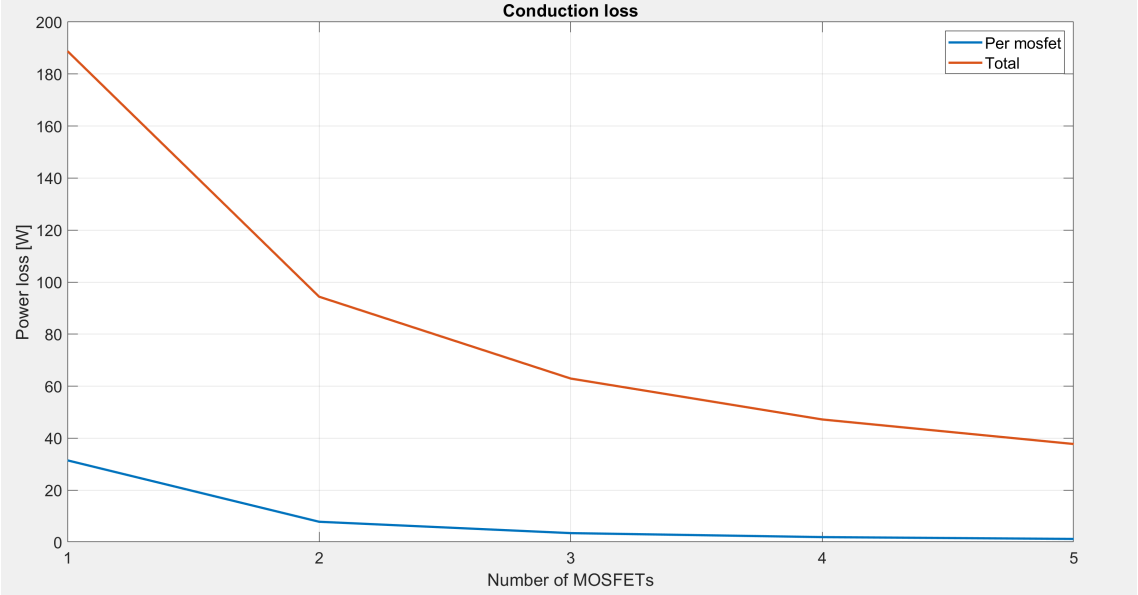
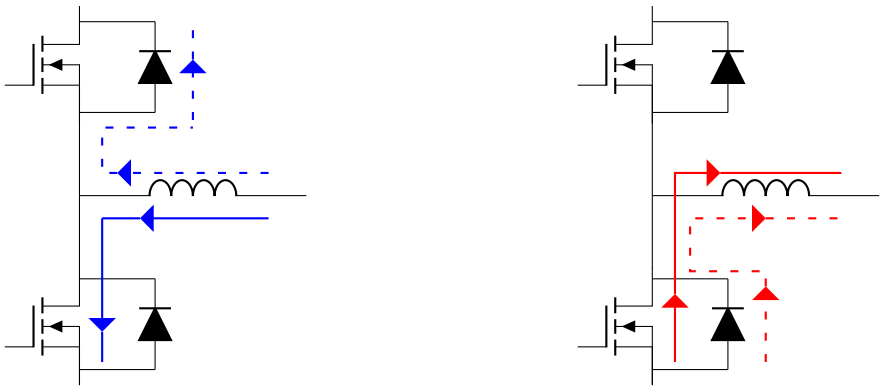


Figure 4: Conduction loss as a function of paralleled MOSFETs

running in the low-side FET as the voltage across it drops. This means current and voltage will be present in the low-side FET simultaneously, meaning a hard turn-on. This scenario is depicted in Figure 5a where the current changes path from the dashed blue line to the solid blue line.

When current runs from the half-bridge into the motor inductor and the low-side FET is turned on, current will already be running in the body diode of the low-side FET, ensuring that the drain-source voltage will be the forward-voltage drop of the body diode, meaning a soft turn-on. This scenario is depicted in Figure 5b where the current changes path from the dashed red line into the solid red line. The high-side switch will operate with a similar mechanism, meaning that the switching in the inverter is done in a combination of hard- and soft-switching. It is assumed that switching losses are zero during soft-switching, therefore, the switching losses are scaled by 0.5. [6, p. 16]



(a) Hard switching of low-side FET (b) Soft switching of low-side FET

Figure 5: Current paths when switching the low-side FET

Another consideration is the current magnitude. As the inverter delivers AC current, it is most appropriate to use the RMS current for calculation the switching losses. This current is assumed to split equally between each parallel MOSFET. The switching losses in each FET can then be estimated with the following equation

$$P_{sw,fet} = \frac{1}{2} \cdot \frac{1}{2} \cdot V_{bat} \cdot \frac{I_{phase,RMS}}{N} \cdot (t_{c,on} + t_{c,off}) \cdot f_{sw} \tag{9.3}$$

where $t_{c,on}$ and $t_{c,off}$ is assumed to be the rise and fall times given in the datasheet. This assumption is most likely not correct, however, estimating correct turn-on and turn-off times is a complicated matter, and for this application it is assumed that conduction losses will be the most significant contributor to the power losses.

From Equation 9.3 it is possible to calculate the combined switching loss of all the MOSFETs in the inverter by multiplying with $N \cdot 2 \cdot 3$. This also reveals that paralleling MOSFETs will not help to lower the overall switching losses. The estimation does, however, convey how paralleling MOSFETs decreases the switching losses in each FET. This is shown in Figure 6.

Along with the conduction losses and switching losses, there will also be dissipated power in the MOSFET as a result of the reverse recovery in the body diodes, the diode conduction losses, and driving of the MOSFETs. These losses are considered insignificant and will therefore be ignored. The estimated power loss depending on the number of MOSFETs in parallel is summarised in Table 3.

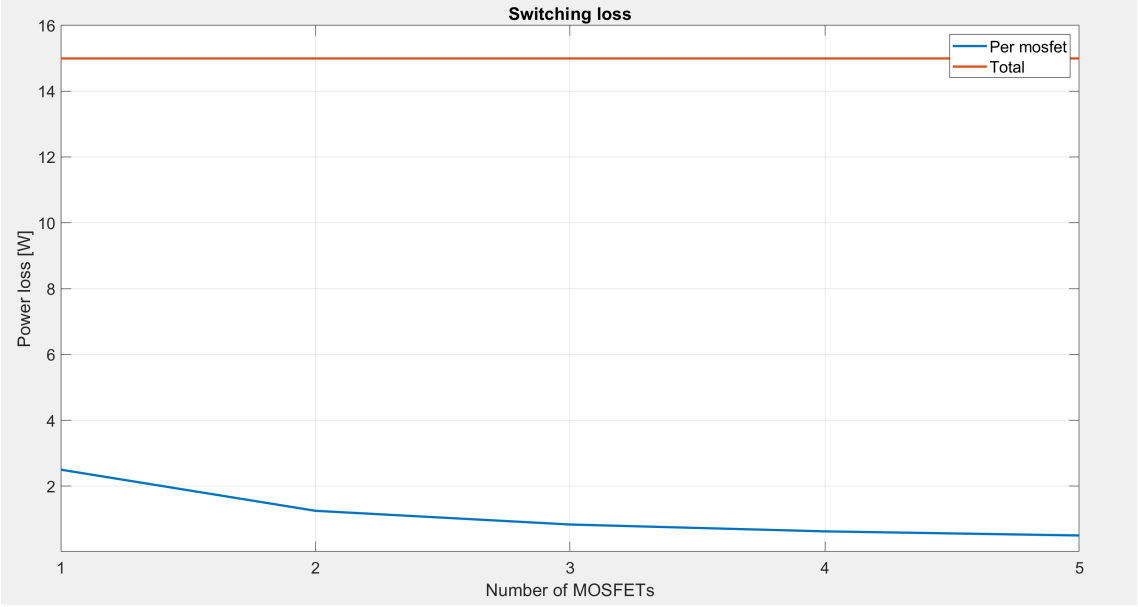


Figure 6: Switching losses in the inverter and each FET dependent on number of FETs in parallel

Number of MOSFETs	1	2	3	4	5
Conduction loss [W]	188.76	94.38	62.92	47.19	37.75
Switching loss [W]	14.99	14.99	14.99	14.99	14.99
Total loss [W]	203.75	109.37	77.91	62.18	52.74

Table 3: Power losses summarised for various configurations of parallel MOSFETs

9.1.3 Thermal Considerations

To better gain an understanding of how many MOSFETs are needed in parallel, the thermal implications are considered based on the power losses estimated in the previous section.

For the inverter board, an aluminium substrate PCB has been chosen. The PCB consists of a 1 oz. copper layer on top, a thin layer of thermally conductive dielectric material with a thermal conductivity of 1 $\frac{W}{m \cdot K}$, and a bottom layer of aluminium substrate. The PCB is then mounted on top of a heat sink with a layer of thermal paste in between.
The build can be seen in Figure 7.

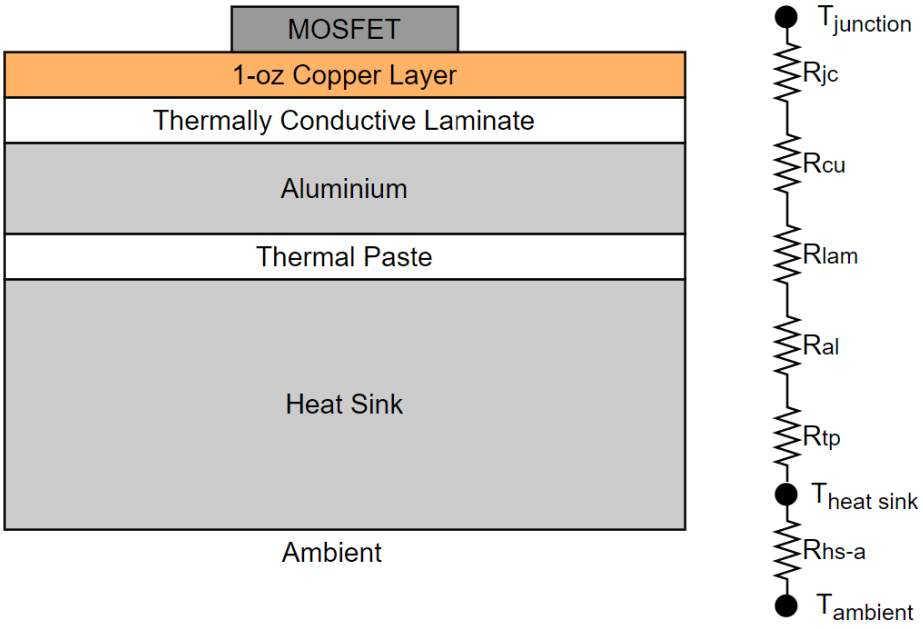


Figure 7: Aluminum PCB stackup with thermal resistances. Inspired by [6, p. 12]

9.1.3.1 Thermal Resistances

To calculate the thermal resistances of each layer, Equation 9.4 can be used. The equation is based on the thermal conductivity and thickness of the material, as well as the area of the "channel" through which the heat will be transferred. This equation assumes that the heat is only transferred vertically through the different layers without spreading laterally.

This assumption is made as taking the lateral spread into consideration is very complex and is usually done by a computational fluid dynamic tool. The assumption means that the thermal resistances will be overestimated. The equation used is

$$R_{th} = \frac{h}{w \cdot l \cdot k} \tag{9.4}$$

with h being the thickness of the layer, w being the width, and l being the length of the MOSFET pad. k is the thermal conductivity of the material composing the layer.

Using Equation 9.4 with the relevant thermal conductivity constants and thicknesses, the thermal resistances are calculated. The thickness of the layers in the PCB is determined by the manufacturer, PCBWay. The copper has a thickness of 35 µm, the dielectric laminate has an average thickness of 114 µm according to the PCBWay blog [7], and the aluminium layer thickness must then be the remaining 1.451 mm of the total PCB thickness of 1.6 mm. The thickness of the thermal paste is estimated to be 0.1mm.

The thermal conductivity for copper and aluminium is $401\frac{\mathrm{W}}{\mathrm{m \cdot K}}$ and $236\frac{\mathrm{W}}{\mathrm{m \cdot K}}$ respectively [8], and the thermal conductivity of thermal paste has been estimated to 3 $\frac{\mathrm{W}}{\mathrm{m \cdot K}}$ based on an average of available thermal paste on RS-Online.

The heat sink chosen for this project is the Fischer SK 47 SA with a length of 200 mm and a thermal resistance of $R_{hs-a} = 0.7\,\frac{^{\circ}\mathrm{C}}{\mathrm{W}}$ [9]. This heat sink was chosen because of its reasonable price and low thermal resistance. A summary of the thermal resistances can be seen in Table 4 and their placement in the PCB stackup can be seen in Figure 7.

Symbol	**Material**	**Thickness [m]**	**Thermal conductivity** $\left[\frac{\mathrm{W}}{\mathrm{m \cdot K}}\right]$	**Thermal resistance** $\left[\frac{^{\circ}\mathrm{C}}{\mathrm{W}}\right]$
R_{jc}	-	-	-	0.4
R_{cu}	Copper	$35 \cdot 10^{-6}$	401	0.0011
R_{lam}	Dielectric laminate	$114 \cdot 10^{-6}$	1	1.47
R_{al}	Aluminium substrate	$1.451 \cdot 10^{-3}$	236	0.08
R_{tp}	Thermal paste	$0.1 \cdot 10^{-3}$	3	0.4296
R_{hs-a}	Heat sink	-	-	0.7

Table 4: Thermal resistances from junction to ambient through the PCB

9.1.3.2 Junction Temperature

Knowing the thermal resistances, it is possible to calculate the junction temperature of the MOSFETs using the equation

$$T_{junction} = T_{amb} + T_{HS-A} + T_{J-HS} \tag{9.5}$$

where T_{amb} is the ambient temperature, T_{HS-A} is the temperature difference over the heat sink and T_{J-HS} is the temperature difference from the junction to the layer of thermal paste. T_{amb} is assumed to 40 °C, as the inverter is placed in a closed environment.

The chosen heat sink has a weight of approximately 1 kg and is made of aluminium, giving a resulting heat capacity of approximately 910 $\frac{\text{J}}{\text{K}}$, as the specific heat capacity for aluminium is 0.91 $\frac{\text{kJ}}{\text{kg}\cdot\text{K}}$ [10]. Considering this, it can be assumed that the power dissipated in the heat sink can be calculated based on the maximum continuous current of 80 A RMS rather that the maximum current of 220 A RMS, as the maximum current can only be applied for one minute.

The temperature rise in the heat sink is calculated as follows

$$T_{HS-A} = P_{80\,\text{A,RMS}} \cdot R_{hs-a} \tag{9.6}$$

where $P_{80\,\text{A,RMS}}$ is the power loss with a current of 80 A RMS. The temperature difference from junction to thermal paste is calculated as follows

$$T_{J-HS} = P_{total,FET} \cdot (R_{jc} + R_{cu} + R_{lam} + R_{al} + R_{tp}) \tag{9.7}$$

where $P_{total,FET}$ is the power dissipated in one MOSFET. Using Equation 9.5 to calculate the junction temperature of the MOSFETs yields the results shown in Figure 8. The results show that using two MOSFETs in parallel results in a junction temperature of approximately 74 °C, which is considered satisfying. Furthermore, the graph shows that there are diminishing returns when increasing the amount of MOSFETs in parallel. In combination with the increases in price, complexity, and gate drive requirements, this means that two or three parallel MOSFETs is the optimal number for this application. This temperature is only an estimate, as it is based on power loss estimations with various assumptions. Therefore, the power loss calculations are verified using a PLECS simulation.

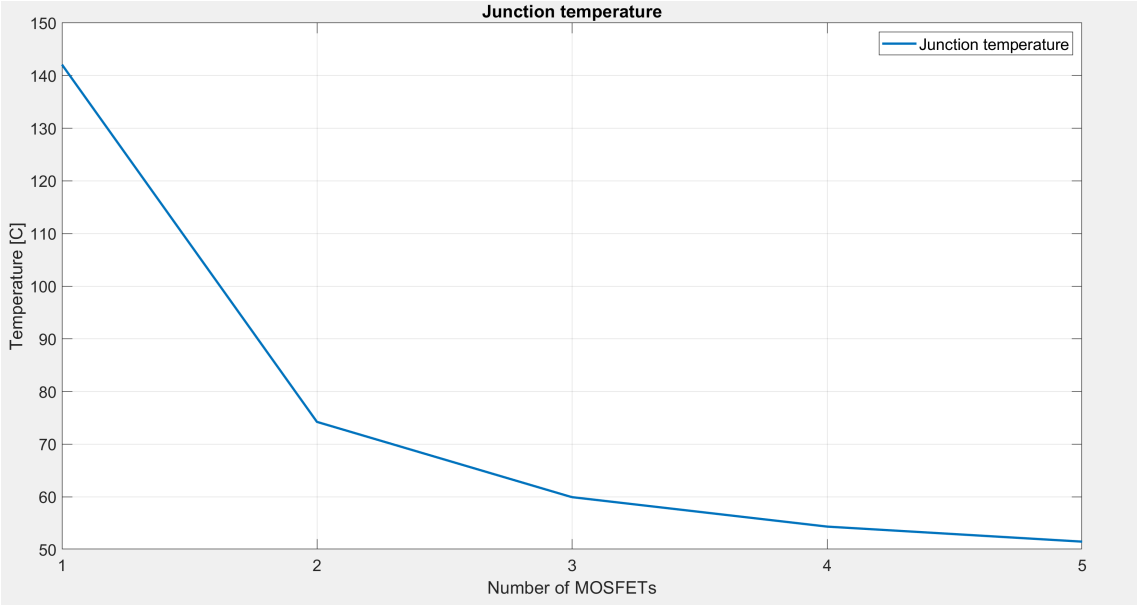


Figure 8: Estimated junction temperature of MOSFETs

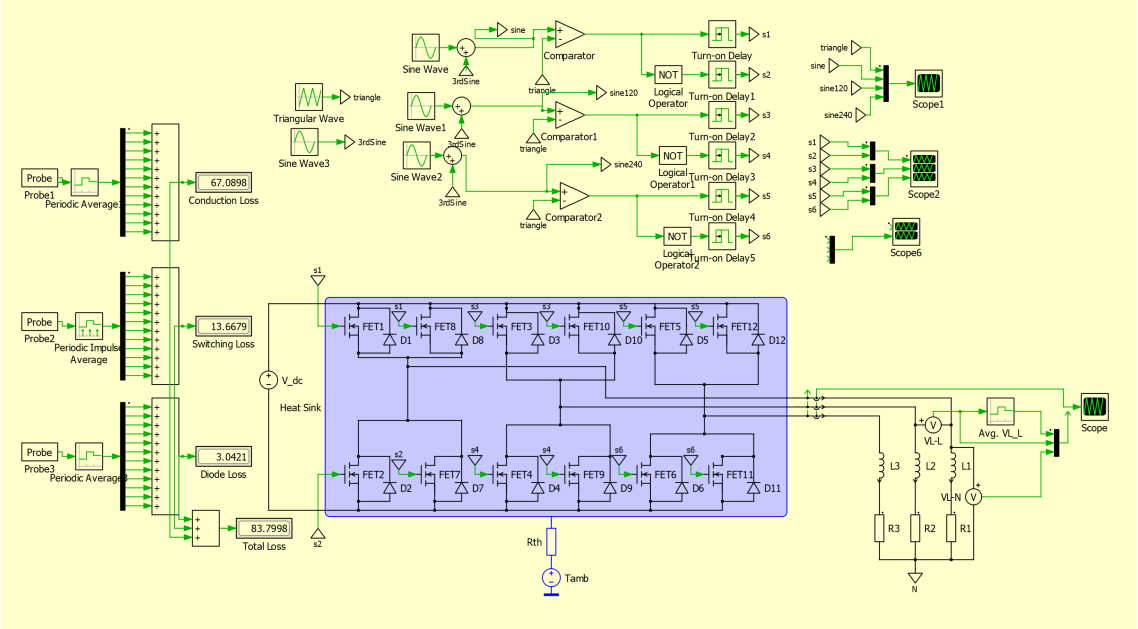


Figure 9: Diagram showing PLECS simulation of three phase inverter with two MOSFETs in parallel

9.1.4 PLECS Simulation

The PLECS simulation is made utilizing thermal models describing the MOSFETs characteristics. The diagram used in the simulation is shown in Figure 9. The simulation is made with a load equal to the stator resistance and stator inductance of the motor and results in phase currents with an RMS value of 223 A. A plot of the simulated currents can be seen in Figure 10.

From Figure 9, it is evident that the conduction loss is approximately 67 W and the switching loss is approximately 14 W. While the switching loss is very close to the es-

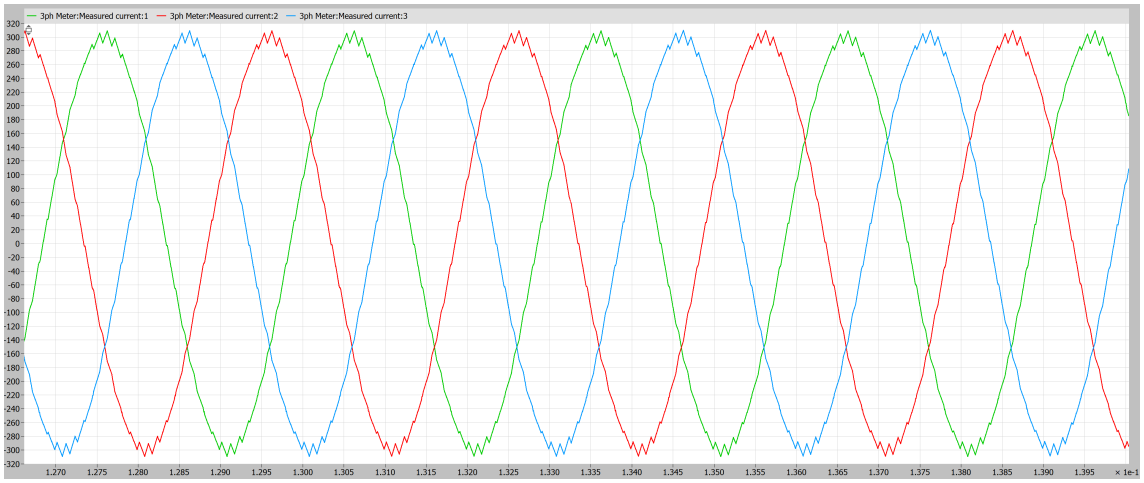


Figure 10: Phase currents simulated in PLECS

timated value of 14.99 W, the conduction loss is approximately 27 W less than the estimated value of 94.38 W. This difference could be down to several factors. One of the factors could be that the PLECS model does not take into account that $R_{ds,on}$ increases with temperature. Another reason for the final differences could be that the approximation of the current in Equation 9.2 is not precise enough. Ultimately, the simulation does not indicate that conduction losses would be larger than estimated in Figure 4, and a design with two MOSFETs in parallel will therefore be implemented.

9.2 Gate Driver

As the Zynq cannot provide adequate current or voltage for switching the MOSFETs, and to isolate the Zynq from the battery voltage, a gate driver must be used.

Choosing a gate driver for the MOSFETs is a choice between a variety of different driver types. There are for example gate drivers for controlling individual sides of the half-bridge, isolated drivers, and non-isolated drivers that control the entire half-bridge with a single PWM-signal. Controlling individual sides requires twice as many drivers and isolation is not deemed necessary. The chosen gate driver is L6494 from ST [11], as this driver is the cheapest of the ones considered, only requires a single PWM input, and offers sink/source capabilities of 2.5 A/2 A, adjustable deadtime, and an integrated bootstrap diode.

An RC lowpass filter is added at the PWM signal to filter out noise. The cutoff frequency should be placed far enough from the switching frequency to avoid affecting the PWM signal, but low enough to have an effect on the noise. A cutoff frequency of $f_c = \frac{1}{2\pi \cdot 1\,\text{k}\Omega \cdot 1\,\text{nF}} = 160\,\text{kHz}$ is chosen.

9.2.1 Bootstrap Capacitor

The integrated bootstrap diode makes it obvious to implement a bootstrap capacitor for handling the floating potential for the high-side gate driver. An alternative is using a supply, which can provide a steady voltage without the need of being charged, but at a higher price.

The bootstrap capacitor, C_B, should be dimensioned to keep a steady voltage when it supplies the voltage for the high-side MOSFETs. The design takes basis in the datasheet for the gate driver [11, p. 13]. The MOSFETs can be seen as external capacitors with capacitance, C_{EXT}, related to the total gate charge, Q_{gate} as such

$$C_{EXT} = \frac{Q_{gate}}{V_{gate}} = \frac{231\,\text{nC}}{12\,\text{V}} = 19.25\,\text{nF} \tag{9.8}$$

where V_{gate} is the gate voltage.

As two FETs are paralleled, the capacitance as seen from the gate driver is $2 \cdot C_{EXT}$. Following $C_B >>> C_{EXT}$, the chosen bootstrap capacitor is 1 µF.

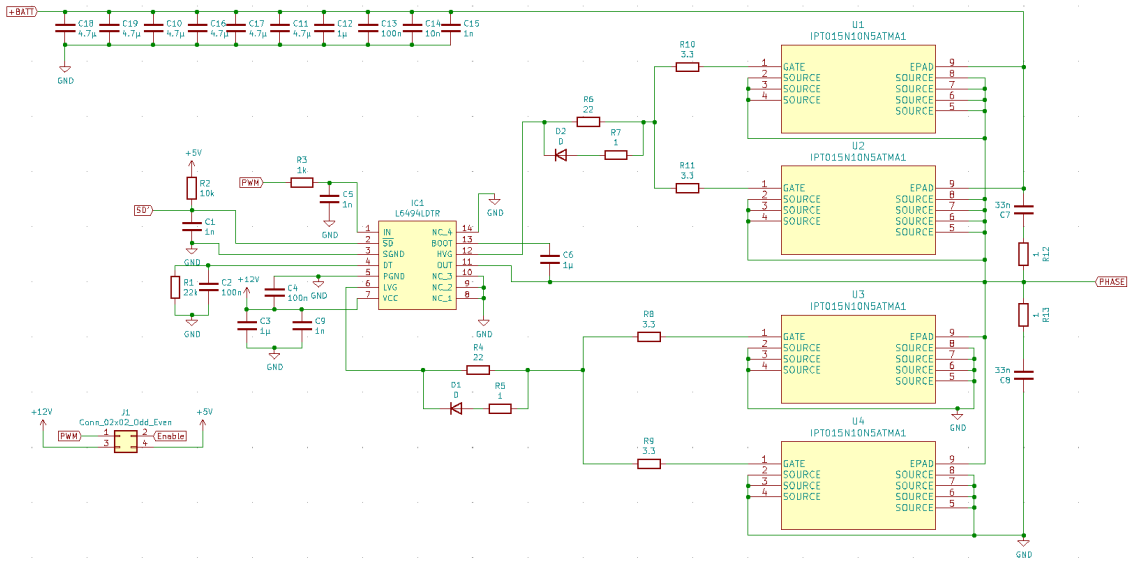


Figure 11: The schematic of the inverter's gate driver circuitry

9.2.2 Gate Resistors

Placing MOSFETs in parallel creates a current path through the gates, sources, and ground. The parasitic inductances will not be equal for the MOSFETs, so an equal $\frac{di}{dt}$ from switching will cause unequal voltages to ground and a current will be circulating, as mentioned in [6, p. 7]. This current can cause gate ringing and must be mitigated. Adding individual resistors close to the gates, R8 through R11 in Figure 11, can help reduce this ringing. Resistors common for the parallel MOSFETs are added to control the turn-on and turn-off times.

The gate resistors will be dimensioned to achieve a turn-on time of $\approx 250\,\text{ns}$ and a turn-off time of $\approx 50\,\text{ns}$. The slow turn-on time is to avoid ringing and the fast turn-off time is to allow for a shorter deadtime without crossconduction. Looking at the low-side MOSFET, the series combination of resistor R4 and an individual resistor, for example R9, will determine the turn-on time, while the series combination of R5||R4 and R9 will determine the turn-off time.

The turn-off time depends on the gate-to-drain charge, $Q_{GD} = 36\,\text{nC}$ [5] and the sink capabilities of the gate driver. The current needed per FET for $t_{turn\_off} = 50\,\text{ns}$ is

$$I_{G,sink} = \frac{Q_{GD}}{t_{turn\_off}} = \frac{36\,\text{nC}}{50\,\text{ns}} = 0.72\,\text{A} \tag{9.9}$$

This is possible, as the gate driver can sink up to 2.5 A.

Then R9 can be determined by

$$R9 = \frac{V_{MILLER}}{I_{G,sink}} - R_{G,int} = \frac{4.5\,\text{V}}{0.72\,\text{A}} - 2\,\Omega = 4.25\,\Omega \tag{9.10}$$

where V_{MILLER} is the Miller plateau voltage of the FET and $R_{G,int}$ is the internal gate

resistance assumed to be 2 Ω. This assumption is based on the $R_{G,int}$ of other FETs in the series, that states the value in their datasheets.

Splitting the calculated resistance into $R9 = 3.3\,\Omega$ and $R5 = 1\,\Omega$ results in $t_{turn\_off} = 50.05\,\text{ns}$. However, the forward voltage of the diode, which will slow the turn-off time down, is omitted in this estimation.

The estimation of R4 takes the other resistances into account by

$$R_{EQ} = \frac{R9 + R_{G,int}}{N} = \frac{4.25\,\Omega + 2\,\Omega}{2} = 2.65\,\Omega \tag{9.11}$$

where N is the number of MOSFETs in parallel.

By finding the necessary source current for the gate driver the same way as before $I_{G,source} = \frac{Q_{GD}}{t_{turn\_on}} = 0.144\,\text{A}$ and including the applied gate voltage $V_G = 12\,\text{V}$, R4 can be determined

$$R4 = \frac{V_G - V_{MILLER}}{I_{G,source} \cdot N} - R_{EQ} = \frac{12\,\text{V} - 4.5\,\text{V}}{0.144\,\text{A} \cdot 2} - 2.65\,\Omega = 23.39\,\Omega \tag{9.12}$$

Using $R4 = 22\,\Omega$ results in $t_{turn\_on} = 237\,\text{ns}$.

9.3 Layout

The economic choice when ordering the PCB is to split the three half-bridges of the inverter onto separate, smaller PCBs, as the price for PCBs increases with the size of the board. Ordering three PCBs instead of one will not change the price, as the boards are to be ordered from PCBWay, where the lowest quantity for an order is 5 PCBs.

A lot of power will be dissipated in the inverter, which is why a heat sink is added. To maximise the effect of the heat sink, it is important to ensure an efficient heat path between the inverter and heat sink. PCBs with a core of aluminium transfers and dissipates heat more efficiently than standard PCBs [12]. In continuation of this, the inverter PCB will be 1-layered as it allows the heat sink to be in closer contact with the metal of the PCB. However, this necessitate the use of a separate PCB for the through-hole capacitors of the DC-link, which furthers the distance between the DC-link and the MOSFETs. To handle the immediate transients, several ceramic decoupling capacitors are added to the circuit, as they can be placed on the aluminium PCB close to the MOSFETs.

When paralleling MOSFETs, it is important that they switch simultaneously. If one FET turns on slightly before the other, a large current will flow through it until the other FETs are turned on enough to conduct current. Therefore, there is a risk of a FET conducting more current than it is rated for. This can happen if the gate paths to the FETs are different. Thus, the MOSFETs are placed close to each other. This also makes it more likely, that they have the same temperature and therefore operate more equally.

A 3D rendering of the inverter PCB can be seen in Figure 12.

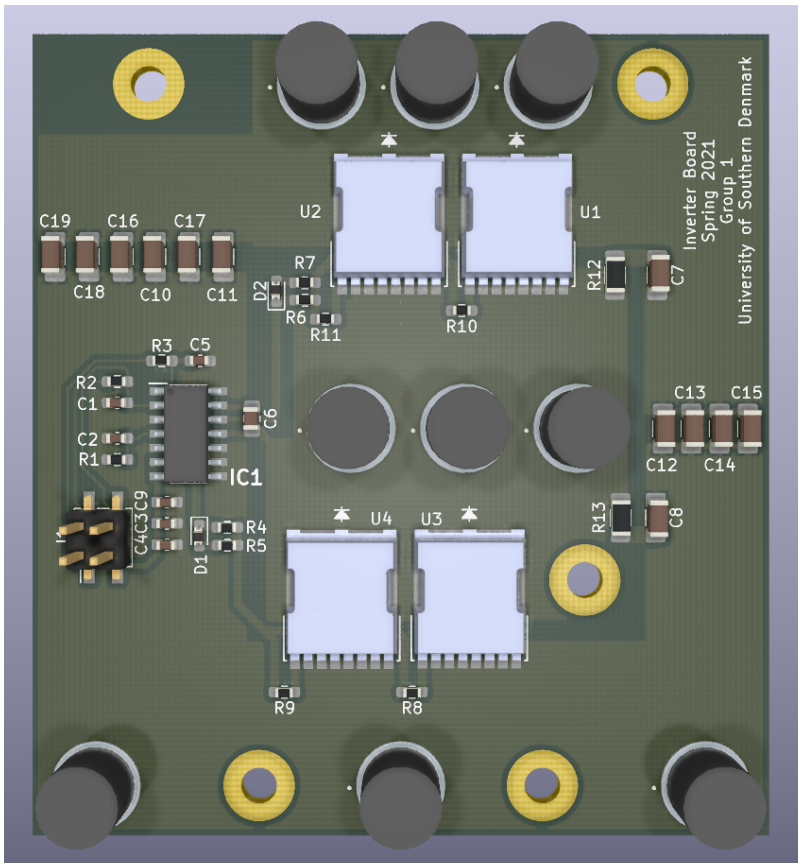


Figure 12: KiCad 3D rendering of the inverter PCB

10 DC-Link

The DC-link appears between the battery and the inverter to create a smooth transition from battery to load and reduce ripples. It typically consists of a capacitor bank, that will prevent transients due to switching in the inverter from reaching the battery.

The two primary factors that determine the size of the DC-link is the current stress on the capacitors and the voltage ripple affected by the total capacitance. When dealing with large currents, it often is the current stress that influences the DC-link the most. Therefore, in order to dimension the capacitors, the current stress on the DC-link is initially analysed.

As the DC-link fundamentally is a capacitor bank, it only experiences the AC components of the current. It is assumed that the current stress is primarily caused by the variations in the current due to the inverter. As mentioned in the paper of [13, p. 86], the RMS value, $I_{C,RMS}$, of the current in the DC-link can be written as

$$I_{C,RMS} = I_{phase,RMS}\sqrt{2M\left(\frac{\sqrt{3}}{4\pi} + cos^2(\phi)\left(\frac{\sqrt{3}}{\pi} - \frac{9}{16}M\right)\right)} \tag{10.1}$$

where $I_{phase,RMS}$ is the phase current, M is the modulation index and $cos(\phi)$ is the power factor.

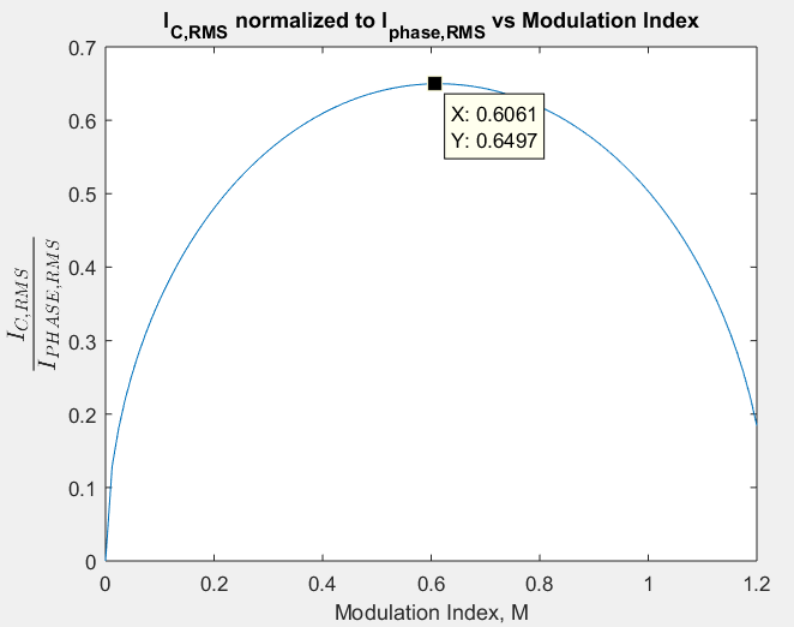


Figure 13: Current in the DC-link as a function of modulation index

To find the worst-case current stress, the power factor is set equal to 1. The modulation index can then be found by normalizing $I_{C,RMS}$ to $I_{phase,RMS}$ and plotting it against the modulation index. As shown in Figure 13 the modulation index that causes the largest current stress is $M = 0.61$.

Using the RMS phase current of 220 A, the resulting current stress on the DC-link is found to be

$$I_{C,RMS} = 220\,\text{A} \cdot \sqrt{2 \cdot 0.61 \left(\frac{\sqrt{3}}{4\pi} + 1^2 \cdot \left(\frac{\sqrt{3}}{\pi} - \frac{9}{16} \cdot 0.61 \right) \right)} = 142.94\,\text{A} \tag{10.2}$$

When dimensioning the DC-link based on the current stress and the rated ripple current of the capacitors, the following equation can be used for applications with a switching frequency greater than 10 kHz [13, p. 88]

$$I^2_{C,RMS,dim} = 0.45 \cdot I^2_{C,RMS} \tag{10.3}$$

where $I_{C,RMS,dim}$ is the current used for the dimensioning.

The chosen capacitor should have a DC voltage rating of at least 80 V, as the maximum voltage on the connected battery is 57.6 V, and preferably a high ripple current rating. The rated ripple current of a single capacitor is compared to $I_{C,RMS,dim}$ to determine the necessary amount of the chosen capacitor. The capacitor "UBY1K391MHL1TO" from Nichicon is an 80 V 390 µF electrolytic capacitor [14] with a rated ripple current of

$$\Delta I_{rated} = 2520\,\text{mA} \cdot 0.94 \cdot 1.5 = 3.55\,\text{A} \tag{10.4}$$

where the factor 0.94 is the frequency correction factor for frequencies between 10 kHz and 100 kHz. As there are no requirements for the lifetime of the DC-link, it is possible

to use a factor 1.5 of ΔI_{rated}.

Using Equation 10.2 through Equation 10.4, the necessary amount of capacitors to withstand the current stress is found to be

$$N_C = \frac{\sqrt{I_{C,RMS}^2 \cdot 0.45}}{\Delta I_{rated}} = \frac{\sqrt{(142.94\,\text{A})^2 \cdot 0.45}}{3.55\,\text{A}} = 26.99 \tag{10.5}$$

The DC-link should thus consist of 27 "UBY1K391MHL1TO" capacitors.

Another aspect of the DC-link is reduction of the voltage ripple due to the capacitance. A reasonable choice of an acceptable voltage ripple on the DC voltage, V_{DC}, is $p_{ripple} = 1\,\%$. The capacitance, C_{req}, required to achieve this with a given switching frequency, f_{sw}, is

$$C_{req} = \frac{1}{8 f_{sw}} \cdot \frac{I_{C,RMS,dim}}{p_{ripple} \cdot V_{DC}} = \frac{1}{8 \cdot 10.66\,\text{kHz}} \cdot \frac{\sqrt{(142.94\,\text{A})^2 \cdot 0.45}}{0.01 \cdot 57.6\,\text{V}} = 2\,\text{mF} \tag{10.6}$$

Therefore, the capacitance of the DC-link should be greater than 2 mF.

The total capacitance, C_{tot}, of the 27 390 µF capacitors is

$$C_{tot} = 27 \cdot 390\,\mu\text{F} = 10.5\,\text{mF} \tag{10.7}$$

As $C_{tot} \geq C_{req}$, the voltage ripple is kept below 1 %.

A pre-charge relay is used to prevent large in-rush currents when the main relay is activated by slowly charging the DC-link capacitors prior to activation. The circuit supplies the capacitors at battery voltage, and is restricted by resistors equivalent to 10 Ω. Knowing the DC-link capacitance of 10.5 mF, the time constant τ_{pre} can be calculated as $\tau_{pre} = \text{RC} = 10\,\Omega \cdot 10.5\,\text{mF} = 0.105\,\text{s}$. To ensure that the capacitors are almost fully charged, five periods must pass, equal to $\tau_{full} = 5 \cdot 0.105\,\text{s} = 0.525\,\text{s}$. At this point, the main relay can be activated and the pre-charge relay deactivated.

11 Analog Interface Board

An interface board is provided as part of the project materials. The board contains power supplies, circuits for pre-charge and main relay control, as well as signal level-shifting and sensor measurements. It also has an isolated circuit for battery voltage measurement, with a 1-to-62 voltage ratio. As the ADC has a resolution of 12 bit, the battery voltage is calculated from the ADC conversion as

$$V_{bat} = \frac{62\,\text{V}}{(2^{12} - 1)\,\text{LSB}} \cdot \text{bits}_{\text{ADC}} \tag{11.1}$$

The board also contains connectors used for mounting the analog interface board, which must be designed, and the Zybo board. Connections for the Zybo board can be found in Appendix E. A summary of the analog board connectors and their signals, as designated on the interface board schematic, is shown in Table 5.

The Zynq's XADC can be configured to simultaneously sample two signals using fixed channels, preserving their phase relationship. This is important for the currents, and they must thus use the auxiliary ADC channels (AUX) 7 and 15, while the torque sensor and battery voltage channels are connected to AUX 6 and AUX 14 respectively. As the conversion timing is important, the ADC is event-driven, in which a signal activates the conversion. This will ensure that the conversion always happens at the same point in a cycle. Whenever all analog signals have been converted once, the XADC will output an End-of-Sequence (EOS) signal, which can be used for interrupting the processing system.

The analog interface will contain the circuits used for handling the current and torque sensor signals before they are converted in the ADC. The sensors to be used are already part of the go-kart system, and cannot be changed.

	J301	**J302**	**J304**	**J305**
Pin 1	5 V	Phase 1 Current Measurement	ADC 1	Digital 1
Pin 2	Ground	Ground	Ground	Ground
Pin 3	5 V Reference	Phase 2 Current Measurement	ADC 3	Digital 2
Pin 4	Ground	Ground	Ground	Ground
Pin 5	3.3 V Reference	Torque Pedal Measurement	ADC 4	Digital 3
Pin 6	Ground	Ground	Ground	Ground
Pin 7	+15 V	NC	NC	Digital 4
Pin 8	Ground	Ground	Ground	Ground
Pin 9	-15 V	NC	NC	Digital 5
Pin 10	Ground	Ground	Ground	Ground

Table 5: Summary of connectors used for mounting the analog interface board

11.1 Current Sensor

Two of the phase currents will be measured by LF 205-S current transducers [15]. These are supplied by the interface board and output a measurement current, I_s, with a ratio of 1:2000. With a maximum RMS phase current, $I_{ph,RMS}$, of ±220 A, as specified in Table 1, the peak phase current is

$$I_{ph,peak} = \pm 220\,\text{A} \cdot \sqrt{2} = \pm 311.13\,\text{A} \tag{11.2}$$

To ensure that the maximum current can be measured, as well as faulty overcurrents, this is adjusted to ±325 A. The peak sense current is then

$$I_{s,peak} = \frac{\pm 325\,\text{A}}{2000} = \pm 0.1625\,\text{A} \tag{11.3}$$

To use the full range of the ADC for optimal accuracy, this current must result in a voltage of ±0.5 V. This is ensured by using a resistor with size

$$R_{Is} = \frac{\pm 0.5\,\text{V}}{\pm 0.1625\,\text{A}} = 3.08\,\Omega \tag{11.4}$$

The closest real value is 3.09 Ω, which will be used. To absorb damaging transients, a TVS diode is used. As the voltage is below any Zener diode's breakdown voltage, a high-

side resistor must be placed, as illustrated in Figure 14. Using a diode with a working voltage of 5 V, which must be the maximum voltage, results in 4.5 V across R_H and thus a value of R_H calculated with voltage division as

$$R_H = 9 \cdot R_{Is} = 9 \cdot 3.09\,\Omega = 27.81\,\Omega \tag{11.5}$$

A parallel combination of 54.9 Ω and 56 Ω results in a resistance of 27.72 Ω.

The resulting power loss in the resistors is at most 0.73 W and 0.08 W for the high-side and low-side resistor respectively.

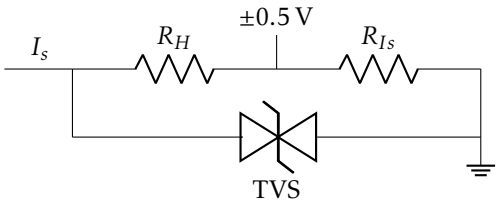


Figure 14: Transient-protected circuit for converting sense current into voltage

The signal must be offset by 0.5 V to result in a value of 0 - 1 V, which is measurable by the ADC. This can be achieved by using a differential amplifier with an offset generated by a reference supply. As accuracy is important for the current measurements, an instrumentation amplifier will be used. While more expensive than a common differential amplifier, it provides higher accuracy and removes noise due to the high Common-Mode Rejection Ratio (CMRR). The amplifier's power supply must be stable to further improve accuracy. The 0.5 V offset is generated by using a reference diode. The instrumentation amplifier to be used is the AD8420ARMZ with high CMRR, low offset, and reasonable bias current and bandwidth [16].

To remove high-frequency signals, a low-pass network is placed at the instrumentation amplifier's inputs. Using a filter introduces a delay to the measurement. Accepting a 6° phase shift at the maximum current frequency of $\frac{5000\,RPM}{60\,s}$ = 333 Hz, and assuming 1° shift due to the amplifiers and parasitic effects, the following equation must be used

$$\varphi_{LP,333} = -\arctan(2 \cdot \pi \cdot 333\,\text{Hz} \cdot R \cdot (2 \cdot C_c + C_g)) = -5° \tag{11.6}$$

in which R is the filter resistor, C_c is the common capacitor between inputs, and C_g is the capacitor from input to ground. The resistors and capacitors are then chosen to fulfill the requirement, giving a differential cutoff frequency of 3.8 kHz and common cutoff frequency of 79.5 kHz. This will effectively remove any high-frequency noise, and a small amount of switching noise. The full circuit is shown in Figure 15.

The measured voltage will be 0.5 V when 0 A is measured, and moves towards 0 V or 1 V depending on the direction of the current. The difference between minimum and maximum current is $2 \cdot I_{ph,peak} = 650\,\text{A}$. Knowing the conversion result, the current can

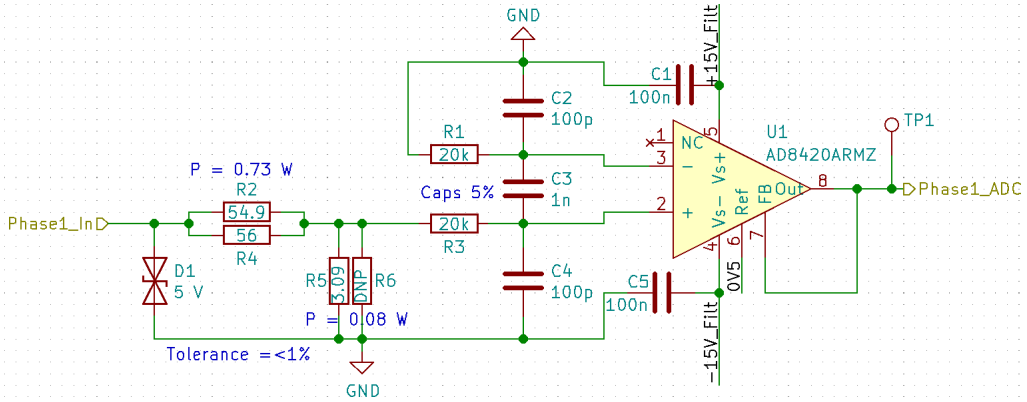


Figure 15: Circuit used for converting measurement current into 0 - 1 V signal

then be calculated with the equation

$$I_{calc} = \frac{650\,\mathrm{A}}{(2^{12}-1)\,\mathrm{LSB}} \cdot \mathrm{bits}_{\mathrm{ADC}} - 325 \tag{11.7}$$

11.2 Torque Sensor

The torque sensor is a 7.5 kΩ potentiometer and is mechanically controlled. For the full sensor range to be measurable, the peak voltage across the potentiometer must be close to 1 V. This is achieved by using a voltage divider as shown in Figure 16.

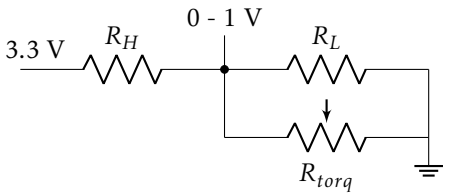


Figure 16: Voltage divider circuit for measuring torque level from 0 - 1 V

The output voltage is then calculated as

$$V_{torq} = 3.3\,\mathrm{V} \cdot \frac{R_{torq} \| R_L}{R_{torq} \| R_L + R_H} \tag{11.8}$$

Setting the maximum voltage to $V_{torq} = 0.9\,\mathrm{V}$ and calculating the maximum value by using $R_{torq} = 7.5\,\mathrm{k\Omega}$ and $R_L = 10\,\mathrm{k\Omega}$, the high-side resistor can be isolated, resulting in $R_H = 11.43\,\mathrm{k\Omega}$. If the torque sensor is disconnected, the resulting voltage is not high enough to damage the XADC. If the measured voltage is above 0.9 V, such as when the sensor is disconnected, an error has occurred and the motor must be stopped for safety reasons.

In an effort to reduce noise in the measurement, a second order Sallen-Key low-pass filter will be placed. The operational amplifier will also work as buffer for the signal. The filter must not attenuate the mechanical signal, but should attenuate the switching frequency

as best possible. A cutoff-frequency of 500 Hz is chosen for the filter. Due to the Sallen-Key filter's imperfections, it will not attenuate as heavily at higher frequencies, which is described in the Texas Instruments article of [17, p. 6]. To remedy this, an RC filter with a high cut-off frequency is placed after it.

The maximum continuous phase current is 80 A RMS, a peak value of 113.14 A, which also results in a peak I_q = 113.14 A. This is what the maximum torque sensor value should be translated to. The equation for I_q as a function of torque pedal conversion result, knowing the maximum voltage is 0.9 V, is

$$I_q = \frac{113.14\,\mathrm{A}}{(2^{12}-1)\,\mathrm{LSB}} \cdot \frac{\mathrm{bits_{ADC}}}{0.9\,\mathrm{V}} \tag{11.9}$$

A button is implemented on the analog interface board which, when pressed, allows time-limited overcurrent of the motor for high torque during initial acceleration. During this period, the maximum phase current is 220 A RMS, a peak value of 311.13 A. The equation to be used is then

$$I_q = \frac{311.13\,\mathrm{A}}{(2^{12}-1)\,\mathrm{LSB}} \cdot \frac{\mathrm{bits_{ADC}}}{0.9\,\mathrm{V}} \tag{11.10}$$

11.3 Layout

A two layer PCB is designed for the analog board signals. The current resistors and Zener diodes are placed as close to the input connectors as possible, while filters are placed as close to the output connector as possible. A large ground plane is placed on both sides, with vias spread out to ensure optimal connectivity.

Subconclusion

In this part, the inverter was designed to operate with a continuous RMS current of 80 A and a peak RMS current of 220 A. The inverter utilises two paralleled MOSFETs to keep conduction losses low, is driven by a bootstrapped gate driver, and is mounted on an aluminium PCB with a heat sink to keep the temperature sufficiently low. The related DC-link is designed to handle the current stress induced by the inverter and keep the voltage ripple below 1 %.

The analog interface board is designed to manage the current sensor and torque sensor signals. An instrumentation amplifier with low phase shift is used for reducing common-mode noise on the current measurement, while a Sallen-Key low-pass filter is used for the torque sensor signal. An overcurrent button is added to allow the motor to operate at peak current for a limited time.

Part III
Control

To control the motor currents, models and approaches must be specified for the motor and controllers. The result must be verified by simulation in Simulink to ensure adequate bandwidth and stability.

12 Field-Oriented Control

The method used for controlling the motor is Field-Oriented Control (FOC), which makes use of digital PI controllers to adjust the currents responsible for orthogonal and parallel electromagnetic forces. A block diagram showing FOC is seen in Figure 17. Only two of the three phase currents, I_a and I_b, need to be measured, and the final phase current is calculated with the equation $I_c = -(I_a + I_b)$, as they are assumed to be balanced.

As time-varying signals are difficult to control and process in a microprocessor, the three phase currents must be transformed into two stationary currents by using the Clarke and Park transformations. The Clarke transformation transforms balanced three-phase quantities into orthogonal two-phased quantities in the α-β plane. The Park transformation then changes the stationary reference frame to a rotating reference frame in the d-q plane. Signals in the various planes can be seen in Figure 18, showing that signals in the d-q plane are stationary [18].

The three measured phase currents are first Clarke-transformed, with the resulting currents called I_α and I_β. These are calculated as

$$\begin{bmatrix} I_\alpha \\ I_\beta \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} I_a \\ I_b \\ I_c \end{bmatrix} \tag{12.1}$$

The Park transformation requires knowing the current electrical angle of the motor, θ_{el}, to rotate the frame with them, thereby creating stationary currents. The stationary cur-

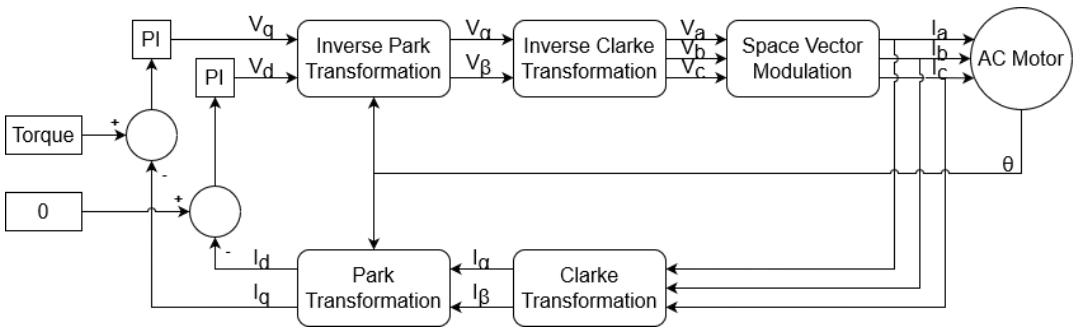


Figure 17: Block diagram of the general workings of field-oriented control

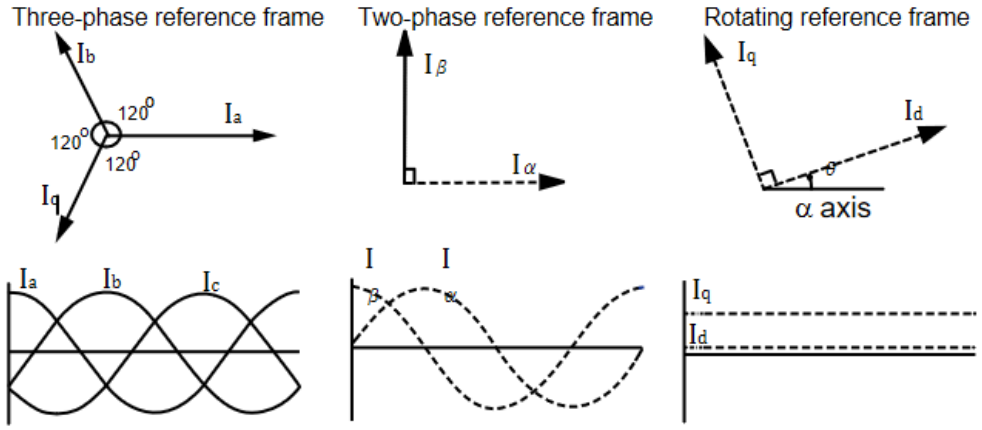


Figure 18: The reference frames and the related signals, from [18, p. 5]

rents are I_q and I_d, and are calculated as

$$\begin{bmatrix} I_d \\ I_q \end{bmatrix} = \begin{bmatrix} \cos(\theta_{el}) & \sin(\theta_{el}) \\ -\sin(\theta_{el}) & \cos(\theta_{el}) \end{bmatrix} \cdot \begin{bmatrix} I_\alpha \\ I_\beta \end{bmatrix} \tag{12.2}$$

As can be seen in Figure 17, these currents are used as input to the PI controllers. The controllers output stationary voltages, which must be transformed back into three rotating voltages by using the inverse transformations. The inverse Park transformation turns V_q and V_d into V_α and V_β with the calculation

$$\begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} = \begin{bmatrix} \cos(\theta_{el}) & -\sin(\theta_{el}) \\ \sin(\theta_{el}) & \cos(\theta_{el}) \end{bmatrix} \cdot \begin{bmatrix} V_d \\ V_q \end{bmatrix} \tag{12.3}$$

The inverse Clarke transformation turns V_α and V_β into V_a, V_b, and V_c, calculated as

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} \tag{12.4}$$

The Zynq Processing System can perform these calculations quickly due to its relatively high clock frequency and built-in Floating-Point Unit (FPU).

The three phase voltages attained by using the inverse Clarke transformation must be realized across the motor by Pulse-Width Modulation (PWM) of the transistors. This can either be through sine-PWM, which uses a sinusoidal reference to determine duty cycle, or Space Vector Modulation (SVM), which overlays a triangular waveform to the sine. The SVM approach has the advantage of 15% better utilization of the supply than sine-PWM, as the line-to-line voltage increases, at the cost of being more computationally heavy [19, p. 208]. SVM is chosen, as the drawback is believed to be minimal, and increased supply utilization results in better performance.

The imposed triangular value is calculated as [19, p. 209]

$$V_{tri} = -\frac{1}{2} \cdot (\max(V_a, V_b, V_c) + \min(V_a, V_b, V_c)) \tag{12.5}$$

in which $\max(V_a, V_b, V_c)$ and $\min(V_a, V_b, V_c)$ are the maximum and minimum value of the three phase voltages respectively.

The voltages are then normalized for the battery voltage by division, and the duty cycles can be calculated as [19, p. 210]

$$d_a = \frac{1}{2} + \frac{V_{tri}}{V_{bat}} + \frac{V_a}{V_{bat}} \tag{12.6}$$

$$d_b = \frac{1}{2} + \frac{V_{tri}}{V_{bat}} + \frac{V_b}{V_{bat}} \tag{12.7}$$

$$d_c = \frac{1}{2} + \frac{V_{tri}}{V_{bat}} + \frac{V_c}{V_{bat}} \tag{12.8}$$

The resulting duty cycles for one phase voltage, when seen over a full rotation, is shown in Figure 19. The figure is from [19, p. 210] and shows a comparison between sine-PWM and SVM for the same output voltage. This shows that a lower duty cycle is required with SVM, indicating that a larger maximum voltage can be attained. The figure also shows the triangular waveform imposed upon a sine to get the resulting SVM duty cycle value.

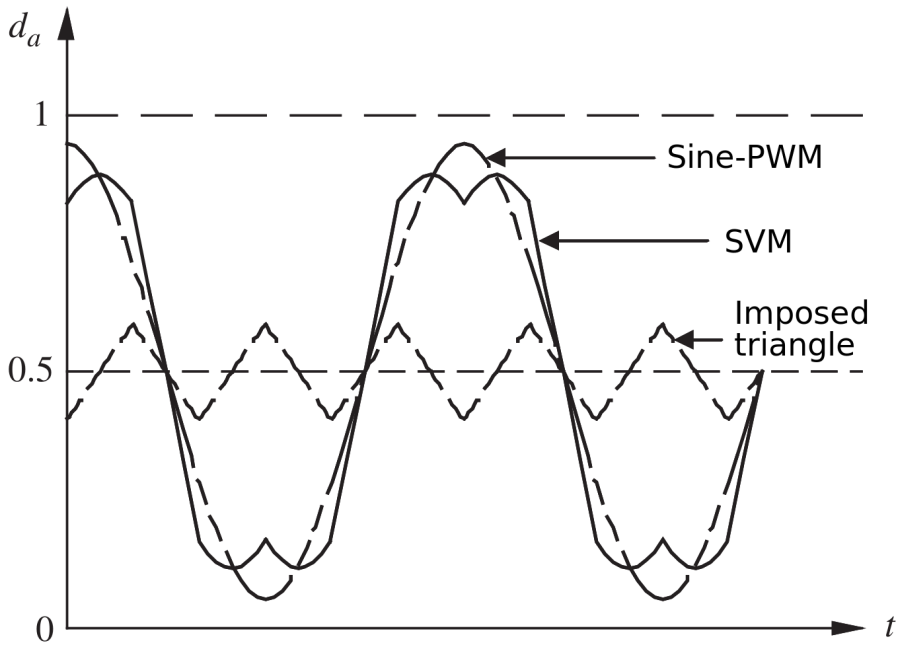


Figure 19: Comparison of duty cycles with sine-PWM and SVM for same output voltage, from [19, p. 210]

13 Motor Model

In order to simulate the motor, a time-domain equation for the voltages in the d- and q-direction, V_d and V_q, must be found. V_d is first found by looking at the equivalent circuit for the motor winding in the d-direction, seen in Figure 20a. This circuit contains the winding resistance and inductance, as well as the induced voltage due to the current in the q-direction. From this circuit, the following equation is derived

$$v_d = R_d \cdot I_d + L_d \frac{dI_d}{dt} - L_q \cdot \omega \cdot I_q \tag{13.1}$$

in which R_d is the motor resistance in the d-direction, I_d is the current in the d-direction, L_d is the inductance in the d-direction, L_q is the inductance in the q-direction, ω is the motor's rotational speed, and I_q is the current in the q-direction.

Similarly, a circuit for the q-direction can be created. This must include the voltage induced by the current in the d-direction, as well as the back-EMF induced by the motor. This circuit is shown in Figure 20b.

The following equation is then derived for the voltage in the q-direction

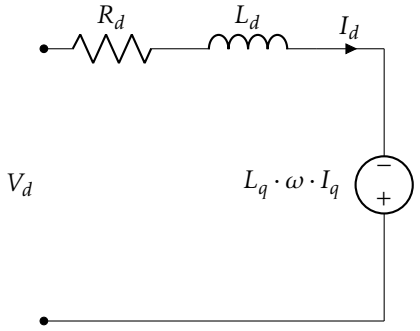
$$V_q = R_q \cdot I_q + L_q \frac{dI_q}{dt} + L_d \cdot \omega \cdot I_d + \omega \cdot \lambda_{MAG} \tag{13.2}$$

in which R_q is the motor resistance in the q-direction, and λ_{MAG} is the motor's flux linkage.

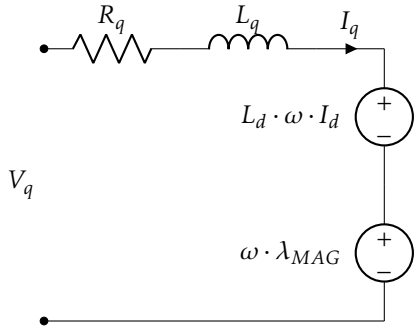
The interaction between current and motor must then be defined. Only the current I_q results in a torque on the motor, dependent of factor $\frac{3}{2}\lambda_{MAG}$. The friction, dependent on rotational speed ω and factor K_f, results in a torque in the opposite direction, as does the load τ_L. Using Newton's second law for rotational systems, the full equation is

$$J_m \cdot \frac{d\omega}{dt} = I_q \cdot \frac{3}{2}\lambda_{MAG} - K_f \cdot \omega - \tau_L \tag{13.3}$$

The motor parameters have previously been measured and have been provided as part of the project materials. These can be seen in Table 6.



(a) Equivalent circuit for the motor winding in the d-direction



(b) Equivalent circuit for the motor winding in the q-direction

Figure 20: dq equivalent circuits

Parameter	Description	Value	Unit
L_q	q-direction inductance	52.5	µH
L_d	d-direction inductance	52.5	µH
R_s	Stator resistance	0.013	Ω
λ_{MAG}	Flux linkage	0.032	Vs/rad
K_f	Friction coefficient	0.1	Nms/rad
J_m	Motor inertia	0.00052	$\mathrm{kg} \cdot \mathrm{m}^2$

Table 6: Measured parameters for go-kart motor

14 PI Controller

To control the motor, a digital PI controller must be designed. The controller will be designed for torque control by adjusting the current, which requires knowing the Laplace-domain transfer function of the circuits in Figure 20a and Figure 20b, those being

$$\frac{I_d}{V_d}(s) = \frac{1}{sL_d + R_d} \tag{14.1}$$

$$\frac{I_q}{V_q}(s) = \frac{1}{sL_q + R_q} \tag{14.2}$$

Using pole placement to design the PI controllers makes it possible to specify settling times for I_d and I_q. Using the transfer function in Equation 14.2, the transfer function for the system when using a PI controller is

$$G_{sys}(s) = \frac{sK_p + K_i}{s^2 L_q + sR_q + sK_p + K_i} \tag{14.3}$$

The desired settling time of the system is chosen to $t_{settling} = 50\,\mathrm{ms}$ as a responsive torque is desired in the racing go-kart. The placement of the poles is determined by the settling time according to

$$s_p = 1.5 \cdot (n+1) \cdot \frac{-1}{t_{settling}} = 1.5 \cdot (2+1) \cdot \frac{-1}{50\,\mathrm{ms}} = -90 \tag{14.4}$$

where s_p is the location of the poles and n is the order of the system, which in this case is 2.

Therefore, the characteristic equation that ensures the chosen settling time is

$$\alpha_c(s) = \left(s - s_p\right)^n = (s - (-90))^2 = s^2 + 180s + 8100 \tag{14.5}$$

The characteristic equation of $G_{sys}(s)$ is

$$\alpha_c(s) = s^2 L_q + sR_q + sK_p + K_i \tag{14.6}$$

$$= s^2 + \frac{1}{52.5 \cdot 10^{-6}} \cdot (6.5 \cdot 10^{-3} + K_p)s + \frac{1}{52.5 \cdot 10^{-6}} \cdot K_i \tag{14.7}$$

Equating the two characteristic equations of Equation 14.5 and Equation 14.7 and isolating for K_p and K_i results in

$$K_p = 0.0029 \tag{14.8}$$

$$K_i = 0.4253 \tag{14.9}$$

As the plant transfer functions are the same, identical PI controllers will be used for both I_q and I_d.

15 Simulink Simulation

To verify different aspects of the control, like the SVM and PI controllers, a Simulink simulation is created. The simulation centers around the motor model developed in Section 13, which is implemented as seen in Appendix C. The PI controllers are implemented with the previously calculated coefficients from Section 14 and have a limited output of $\pm V_{bat}$ as seen in Appendix D.

The PI controllers and the motor model have then been combined into a complete system, which can be seen in Figure 21. The I_d and I_q outputs from the motor model are inverse Park and inverse Clarke transformed, yielding three phase currents. Two of the three phase currents are then fed back through Park and Clarke transforms, to the controller input acting as negative feedback for the system.

Zero-Order Hold (ZOH) delays are inserted to simulate the sampling in the real system. Furthermore, transport delays are inserted after the PI controller acting as calculation delays. These delays and ZOHs ensure that the effects, if any, of sampling the currents and implementing the controls digitally will be seen in the simulation results.

The simulation is executed with parameters from Table 1 and Table 6. The friction coefficient has been multiplied with 10 and the moment of inertia has been multiplied with 1000 to better simulate a situation where the motor is mounted in a go-kart.

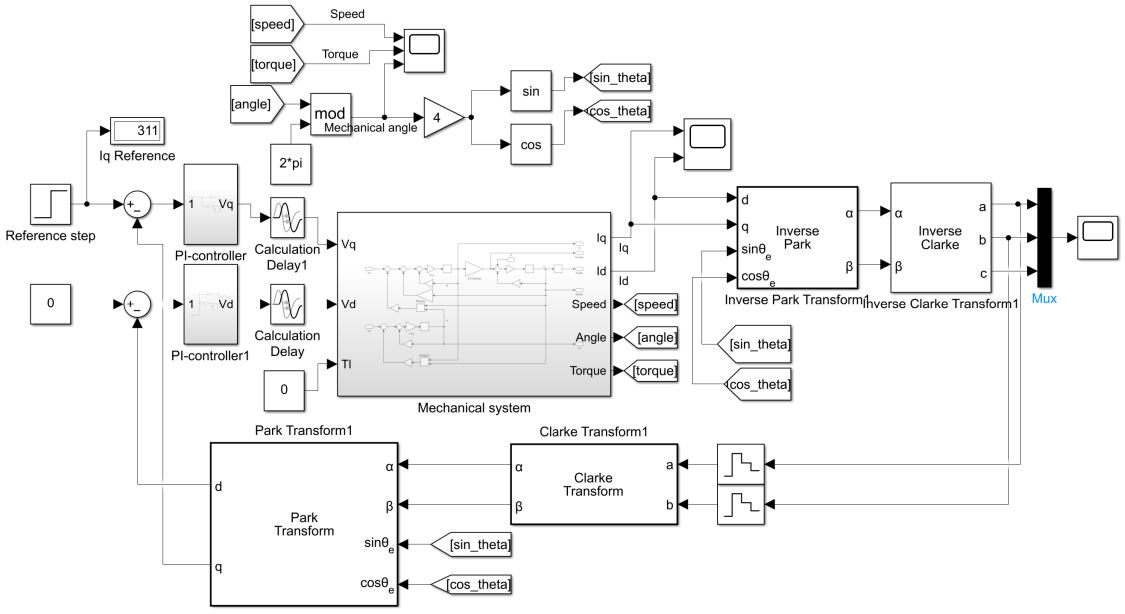
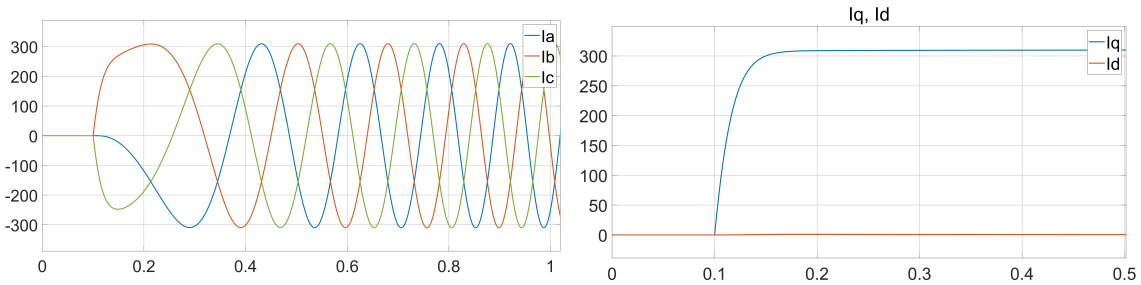


Figure 21: System used for Simulink simulation

15.1 Simulation Results

Imposing an I_q reference step of 311 A results in the three phase currents shown in Figure 22a with a peak value of 311 A and an RMS value of approximately 220 A. Looking at simulation results in Figure 22b reveals that the controller ensures that the current in the q direction settles in 44 ms, which is very close to the settling time desired in Section 14. A simulation with an IP controller was performed, showing almost identical results.

Finally, the results in Figure 23 reveal that even though the torque is delivered almost instantly, the mechanical system clearly limits the acceleration of the motor. It is evident that the motor has a settling time close to 2.5 seconds, this settling time does not reveal much about how the go-kart will behave in reality, as the friction constant and moment of inertia would then have to be estimated specifically for the go-kart.



(a) Phase currents step response

(b) Step response of I_d and I_q

Figure 22: Results from Simulink simulation of step

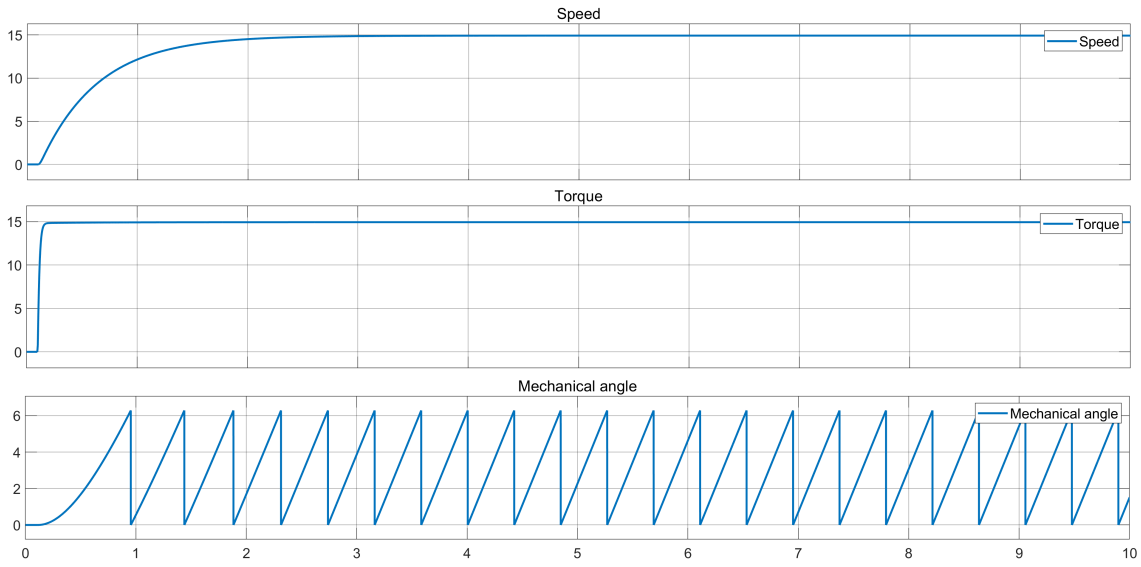


Figure 23: Simulation results showing speed, torque and mechanical angle

15.2 Simulation of Space Vector Modulation

To test the SVM method and verify that the method is viable, the simulation is expanded. A branch is made from the V_d and V_q outputs of the controllers into an inverse Park and inverse Clarke transformation creating three voltages V_a, V_b, and V_c. These are then turned into duty cycles using the method described in Section 12. The implementation of the method can be seen in Figure 24 and the results showing the duty cycle outputs can be seen in Figure 25. The SVM duty cycles look as expected, confirming that the method works.

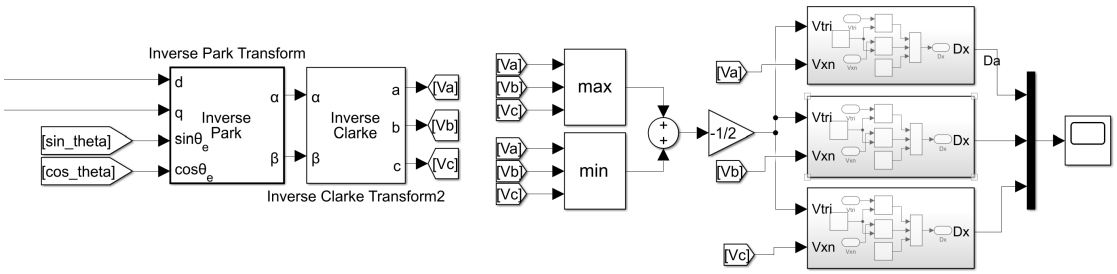


Figure 24: Simulink implementation of space vector modulation

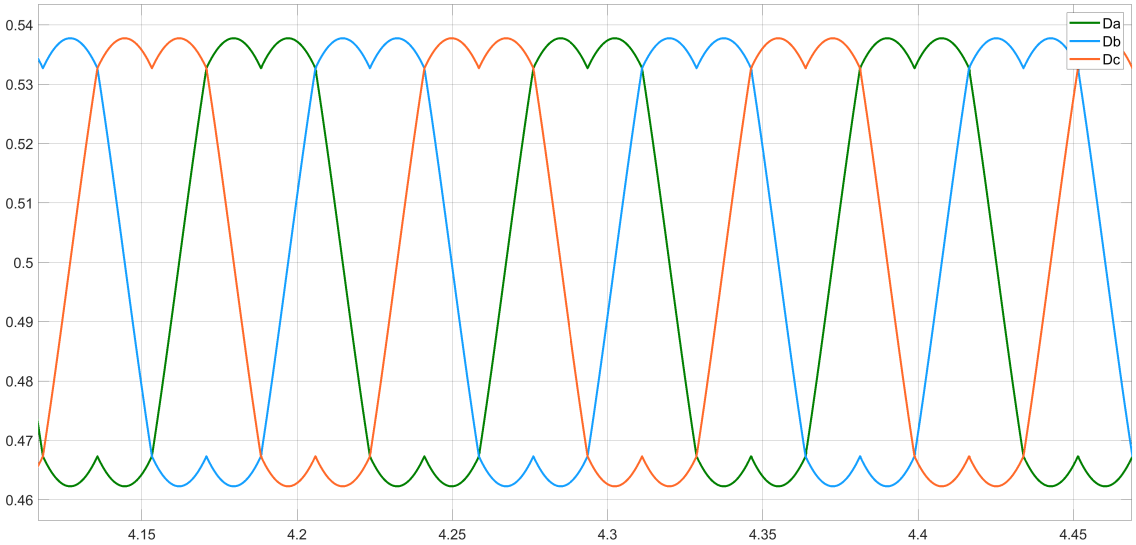


Figure 25: Simulink scope showing SVM results

Subconclusion

In this part, the control system for the motor controller was designed. The chosen control method was field-oriented control using digital PI controllers with $K_p = 0.0029$ and $K_i = 0.4253$. The control was verified using a Simulink simulation based on the motor model, which also included saturation, sampling and calculation delays. Furthermore, a method for duty cycle generation was chosen and similarly verified by simulation.

Part IV
Embedded System

This part will describe the design approaches for both programmable logic and processing system software within the Zynq. An overview of the overall Zynq system architecture is shown in Figure 26, which will be specified in the following sections.

Code implementation for all central parts of the system are found in the Code Snippets chapter at the end of the report.

16 Programmable Logic

The Zynq PL is programmed using Vivado 2018.3 and the hardware description language VHDL. Some functionalities are Intellectual Property cores (IPs) provided by Xilinx, while others are custom designed for this application. The full Vivado block diagram of the programmable logic software is shown in Appendix F.

16.1 Zynq Processing System

In the Vivado block diagram, the ZYNQ7 Processing System IP is used to initialize the Zynq-7010 correctly. This handles clocks, fixed input/output (IO), Random-Access Memory (RAM), and peripherals. The Pmod headers are initially unused, and must thus be initialized by using an AXI GPIO IP in the block diagram, which is associated with a pin in the synthesis stage as listed in Appendix E. The XADC is configured with the XADC Wizard IP using settings specified in Section 11.

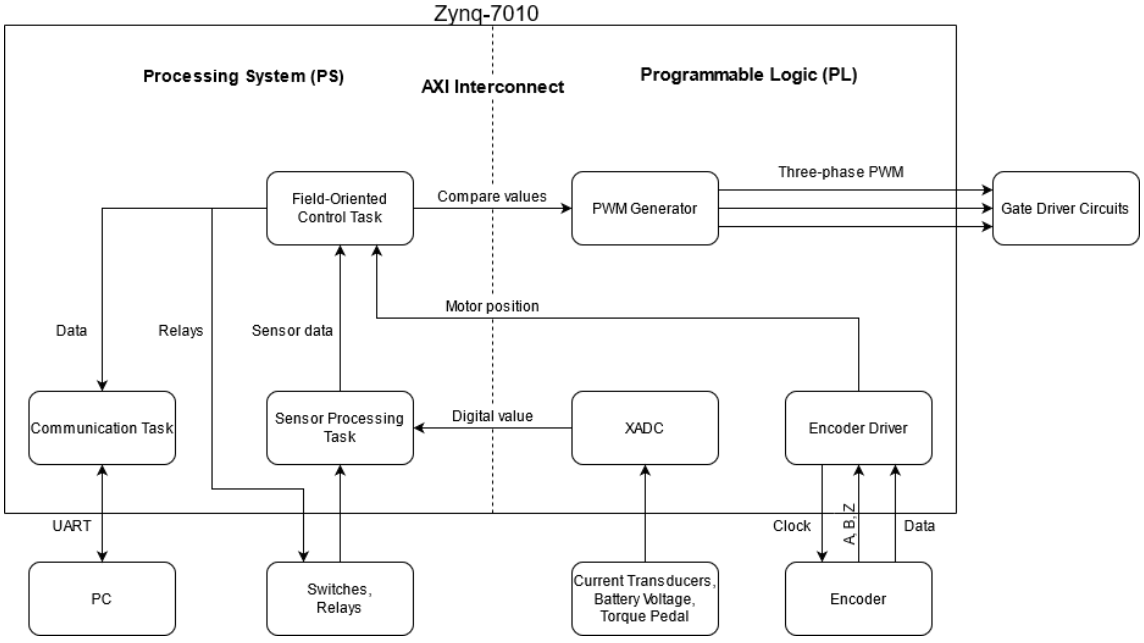


Figure 26: Overview of programmable logic and processing system software

16.2 Pulse-Width Modulation Generator

The PWM generator uses a scaled-down clock to increment a counter. This counter initially starts at 0 and counts toward the maximum value. When it reaches the maximum value, the counter will then count downwards to 0, creating a triangle. A triangular counter is used, as this minimizes harmonic content on the output, as opposed to using a sawtooth counter [19, p. 193]. The counting frequency and maximum value can be changed to attain a specific frequency. The code used to create the VHDL process with a state machine can be seen in Listing 1. Statements used for comparing and generating outputs have been left out in the listing.

```
process (clk_scaled)
begin
    if rising_edge(clk_scaled) then
        if S_AXI_ARESETN = '1' and slv_reg3 = x"00000001" then
            case state is
                when COUNT_UP =>
                    if counter < COUNTER_MAX then
                        counter <= counter + 1;
                    else
                        counter <= counter - 1;
                        state <= COUNT_DOWN;
                    end if;

                when COUNT_DOWN =>
                    if counter > 0 then
                        counter <= counter - 1;
                    else
                        counter <= counter + 1;
                        state <= COUNT_UP;
                    end if;
            end case;
            -- Output statements left out for compactness
        end if;
    end if;
end process;
```

Listing 1: PWM generator process and state machine in VHDL

The PWM generator will receive three compare values through the AXI interface from the processing system, one for each phase, and generate a PWM signal for each of them. The gate driver needs only one PWM input for the half-bridge, and handles the deadtime. The AXI interface is created with the Vivado AXI peripheral IP generator.

The output compare value is calculated in the PS as $OCV = d_x \cdot (count_{max} + 1)$, in which d_x is the duty cycle of one phase and $count_{max}$ the maximum counter value.

The PWM output will be high when the counter value is lower than the output compare value, and vice versa. At the counter's peak, a pulse will initiate an XADC conversion of the phase current measurements. At zero, a second pulse will initiate conversion of the torque sensor and battery voltage measurements. The waveforms of the PWM generator is shown in Figure 27.

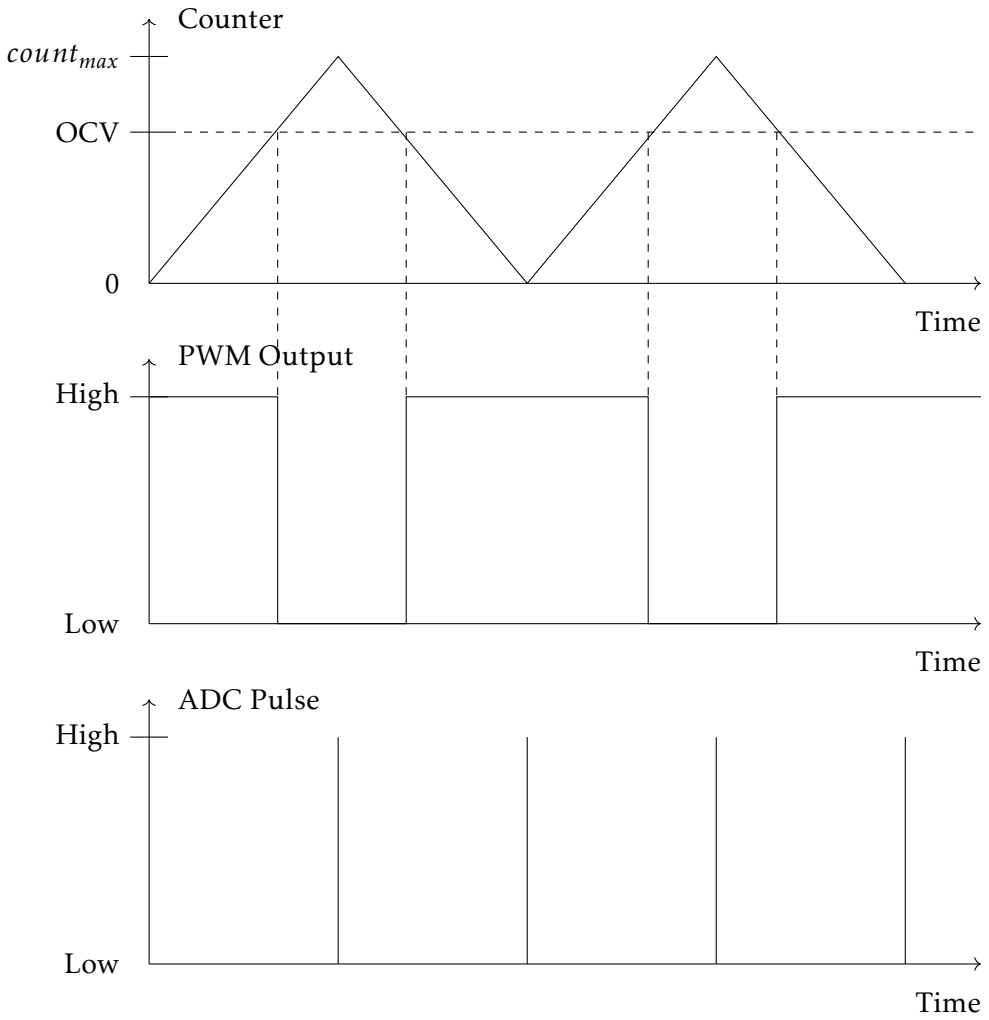


Figure 27: PWM generation using triangular counter, including ADC trigger pulse

16.3 Encoder Driver

The encoder driver is created with an AXI interface in Vivado like the PWM generator. Immediately upon startup, the driver will start a 900 kHz serial clock and log the 8-bit position returned by the encoder. It will then change state depending on the current A and B signals. The 8-bit position is then continuously updated by moving between four states, dependent on the A and B signals. The position is incremented or decremented in accordance with the direction. If the Z signal is asserted, the position is reset to zero. A state diagram for the encoder is shown in Figure 28. The starting state, GET_ABS, is only used immediately after reset and cannot be reached afterwards.

The 8-bit position value can be read by the PS to receive the mechanical angle, and is transformed to the electrical angle in radians with the equation

$$\theta_{\text{el}} = 4 \cdot \frac{2 \cdot \pi}{255} \cdot \text{pos}_{\text{bits}} \tag{16.1}$$

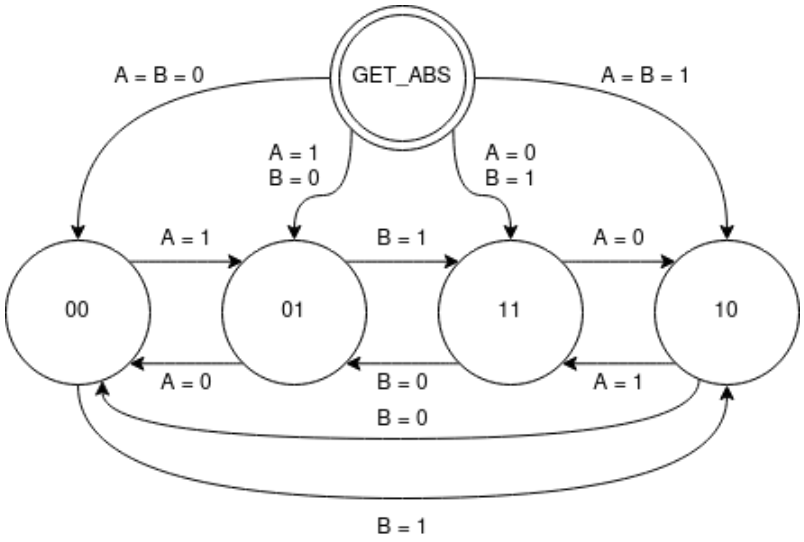


Figure 28: State diagram of the encoder driver IP core

17 Processing System

The code run by the PS is written in C in Xilinx SDK 2018.3 and uses drivers created by Xilinx for using GPIOs, initializing the interrupt, reading the XADC, and communicating through UART.

The processing system consists of three tasks as specified in Figure 30. The Sensor Processing task handles all sensor values and calculates physical values. The task also reads digital I/O pins to confirm switch, relay, and overtemperature status. The data is then used by the Field-Oriented Control task. This contains a state machine that ensures start-up protocol is followed, and any dangerous situation is avoided. It also performs Park-Clarke transformations and incorporates the PI controllers, eventually performing space vector modulation, resulting in a compare value used by the PWM generator. The Communication task continuously checks for UART messages from the PC, which may set or get data values as needed.

The XADC EOS signal creates an interrupt in the processing system and indicates that new values are accessible. As responsiveness is important in the system, the Sensor Processing task and FOC task will be run within the interrupt. The program flow within the Interrupt Service Routine (ISR) is shown in Figure 29.

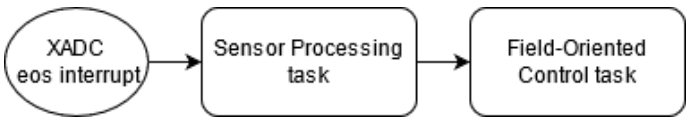


Figure 29: Execution flow within the XADC end-of-sequence interrupt service routine

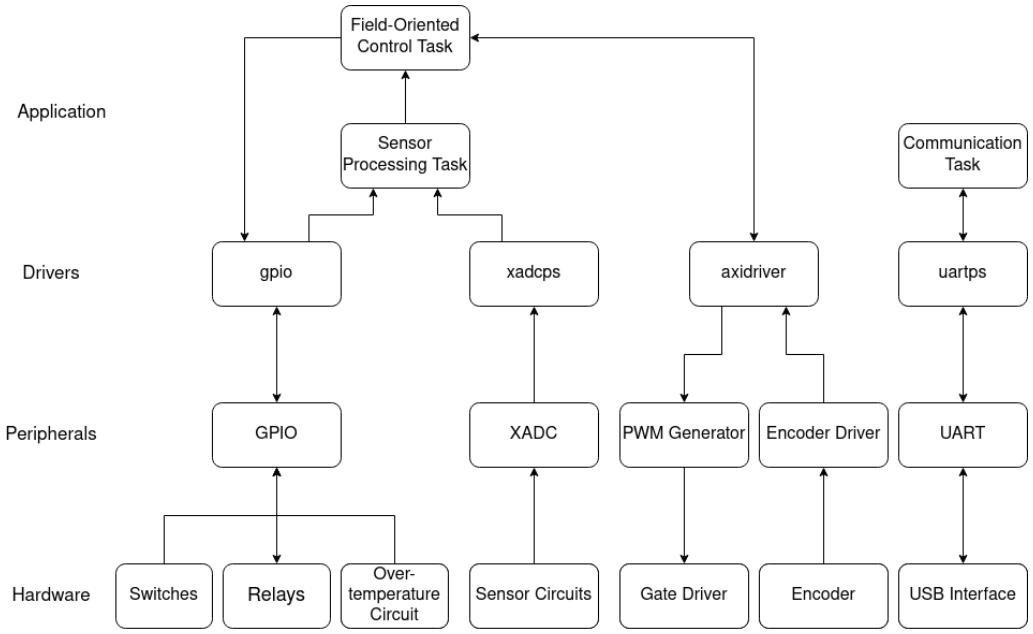


Figure 30: Software architecture of Zynq Processing System and associated hardware

The Communication task timing is less important, and the task is placed in a run-to-completion scheduler in the main program section. This will be interrupted often, but the delay is insignificant, as transmission over UART is slow, and the delay is unlikely to be perceived by the recipient at the PC. The execution flow for power-on and main loop is shown in Figure 31.

In the coming sections, the functions used by the tasks will be described. After this, the tasks themselves will be further specified.

17.1 Hardware Abstraction Layers and Functions

In order to facilitate development of the tasks, Hardware Abstraction Layers (HALs) are created for all peripherals and hardware connections. This includes the XADC, interrupt controller, GPIOs, UART, and PL modules. Additionally, functions are created for handling field-oriented control mathematics and control in the tasks. The sections below will specify the most important HALs and functions used in the program.

17.1.1 AXI Interface Driver

The PL module memory addresses are automatically mapped into the PS by the Vivado bitstream. The auto-generated header file ”platform.h” and the driver files in the hard-

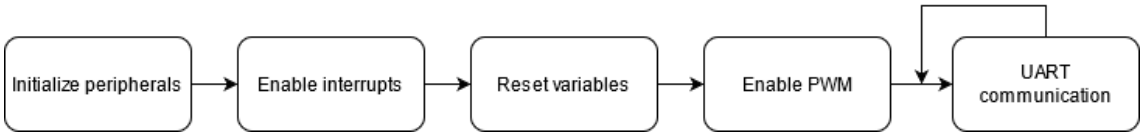


Figure 31: Execution flow of main function containing the Communication task

ware wrapper folder can then be used to communicate with the AXI modules. These are used to create a HAL that allows reading from, writing to, disabling, and enabling the modules. The function used for reading the mechanical motor position as an 8-bit value from the encoder driver is shown in Listing 2.

```
1 void readPosition(u16 *raw_pos) {
2   *raw_pos = ENCODER_DRIVER_mReadReg(ENCODER_DRIVER_BASEADDR,
              ENCODER_DRIVER_S00_AXI_SLV_REG0_OFFSET);
3 }
```

Listing 2: Function used for reading position from encoder driver

17.1.2 Digital PI Controller

The continuous PI controllers are discretized by using bilinear Z-transform, after which the transfer function can be represented by a difference equation. The transfer function for a PI controller in the s-domain is

$$G(s) = K_p \cdot \frac{\tau_i \cdot s + 1}{\tau_i} \tag{17.1}$$

where $\tau_i = \frac{K_p}{K_i}$.

Using the MATLAB function 'c2d()' for transforming the transfer function to the Z-domain results in

$$G(Z) = \frac{a0 \cdot Z - a1}{b0 \cdot Z - b1} \tag{17.2}$$

Dividing the numerator and denominator with the highest power of Z allows the transfer function to be written in terms of delays

$$G(Z) = \frac{Y(Z)}{X(Z)} = \frac{a0 - a1 \cdot Z^{-1}}{b0 - b1 \cdot Z^{-1}} \tag{17.3}$$

where $Y(Z)$ is the output and $X(Z)$ is the input. Z^{-m} represents a time delay, which means that $X(Z) \cdot Z^{-m}$ represents the input at $t = (n-m) \cdot T$. Assuming the T is implicit, this input will be written as $x(n-m)$. Using this and rearranging Equation 17.3 results in

$$b0 \cdot y(n) = b1 \cdot y(n-1) + a0 \cdot x(n) - a1 \cdot x(n-1) \tag{17.4}$$

Thus, the transfer function is transformed to a difference equation, which is implemented in software, as it is faster than a differential function for a microprocessor to process.

Integrator anti-windup is implemented by setting a upper and lower limit for the output of the PI controller. An if-statement is checking whether the output is within the limits, and clamps it to said limits if this is not the case.

The software implementation of the PI controller can be seen in Listing 3.

```
void newSample( controllerPI_t *cont, f32 new_sample,
                f32 *output ) {
  f32 temp_output;

  temp_output = cont->b_1*cont->prev_output + cont->a_0*
                new_sample - cont->a_1*cont->prev_sample;

  if( temp_output > cont->limit ) {
    temp_output = cont->limit;
  }
  else if( temp_output < (-cont->limit) ) {
    temp_output = -cont->limit;
  }

  cont->prev_output = temp_output;
  cont->prev_sample = new_sample;

  *output = temp_output;
}
```

Listing 3: Implementation of the digital PI controller when new data is sampled

17.1.3 Taylor Approximation of Sine and Cosine

The sine and cosine functions are necessary for Park and inverse Park transformations as shown in Section 12. In order to increase the processing speed of the calculations, the sine and cosine are implemented by the use of a 9th order Taylor approximation. The approximation of the sine function is [20, p. 43]

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \tag{17.5}$$

$$= x\left(1 + x^2\left(-\frac{1}{3!} + x^2\left(\frac{1}{5!} + x^2\left(-\frac{1}{7!} + x^2 \cdot \frac{1}{9!}\right)\right)\right)\right) \tag{17.6}$$

In order to increase the processing speed, the factorials and divisions are calculated in advance and the resulting constants are used in the software implementation.

The absolute error of the Taylor approximation from -π to π is shown in Figure 32. This shows that the approximation error rises near -π and π, and will continue outside these limits. Thus, if the input is out of this range, it will traverse the unit circle by either adding or subtracting 2π until in range for increased accuracy.

The cosine function is a displacement of the sine function

$$\cos(x) = \sin\left(\frac{\pi}{2} + |x|\right) \tag{17.7}$$

After displacing the input, the cosine function will make use of the previously explained sine function.

17.1.4 Clarke and Park Transformations

The Clarke, Park, and the inverse of these transformations are shown in Equation 12.1 through Equation 12.4. The digital implementation of the transformations follow these

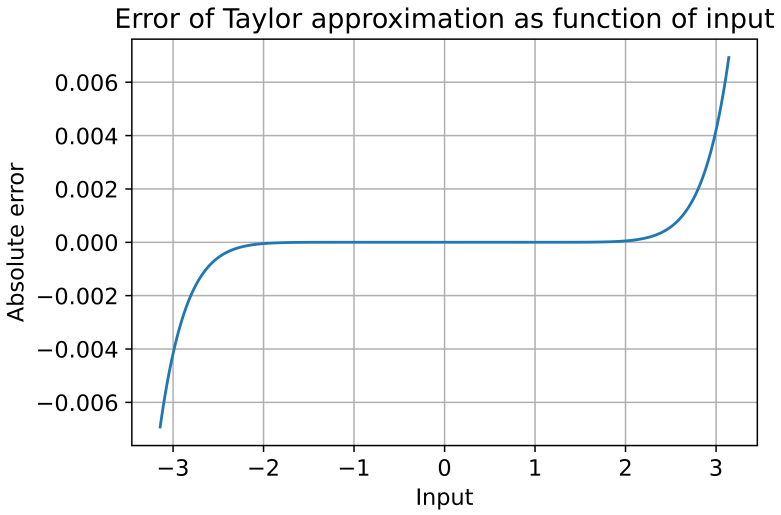


Figure 32: Absolute error of 9th order Taylor approximation of sine

equations, which is evident from the Park transformation shown in Listing 4.

```
void parkTransform( two_phased_t *input, f32 input_angle,
                    two_phased_t *output ) {
  output.arg1 = input->arg1*cos( input_angle )
              + input->arg2*sin( input_angle );
  output.arg2 = input->arg1*cos( input_angle )
              - input->arg2*sin( input_angle );
}
```

Listing 4: The implementation of the Park transform

The transformations make use of the previously defined sine and cosine as well as structs containing the currents or voltages forming the inputs and outputs of the transformations. This way, correlating values are kept together. One such struct, containing two values, is shown in Listing 5.

```
typedef struct two_phased_t {
  f32 arg1;
  f32 arg2;
} two_phased_t;
```

Listing 5: The inputs and outputs of the transformations are contained in structs similar to the two-phased struct shown here

17.1.5 Space Vector Modulation

Following the implementation from Section 12, an SVM function is written that takes the phase voltages and battery voltage as inputs and outputs the PWM compare values. This can be seen in Listing 6, with the code for comparing phase voltages left out.

```
void spaceVectorModulation(three_phased_t* vabc, f32 v_bat,
                           ocvvalues_t* ocvs) {
    f32 max_phase, min_phase;

    // Code for finding maximum and minimum phase voltages left
   out for compactness

    f32 v_tri  = -0.5 * (max_phase + min_phase);
    ocvs->ocv1 = (0.5 + v_tri/v_bat + vabc->arg1/v_bat)
                 * OCR_MAX;
    ocvs->ocv2 = (0.5 + v_tri/v_bat + vabc->arg2/v_bat)
                 * OCR_MAX;
    ocvs->ocv3 = (0.5 + v_tri/v_bat + vabc->arg3/v_bat)
                 * OCR_MAX;
}
```

Listing 6: Implementation of space vector modulation for calculating PWM compare values

17.2 Sensor Processing Task

The task checks the digital signals from the enable and foot switch, the overcurrent activation switch, the overtemperature warning, and the main relay status. It then retrieves the raw digital values for each of the four analog measurement values from the XADC. These are placed in their own structs, which also contains the measurement offset, the filtered measurement, and the physical value equating to it. The filtered values are obtained by subtracting the offset from the raw value.

The task then checks for errors by looking for values out of their expected ranges. This may be battery overvoltage or undervoltage, torque sensor overvoltage implying a disconnect, phase overcurrent, or motor overtemperature. All errors are indicated by one bit, either 0 or 1, in the errors_t struct, where 1 corresponds to a fault.

The physical values of voltage and current are then calculated with Equation 11.1, Equation 11.7, and Equation 11.9. If the overcurrent switch has been pressed within ten seconds, overcurrent is still active. The third phase current is calculated from the other two. The task ends with reading the motor position as specified by the encoder driver. The raw 8-bit position, as well as mechanical and electrical positions in radians, are stored in the encoder_t struct. The task's flow is described in Figure 33.

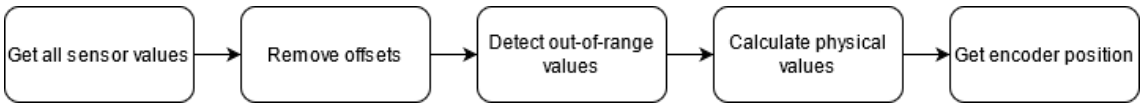


Figure 33: Execution flow within the Sensor Processing task

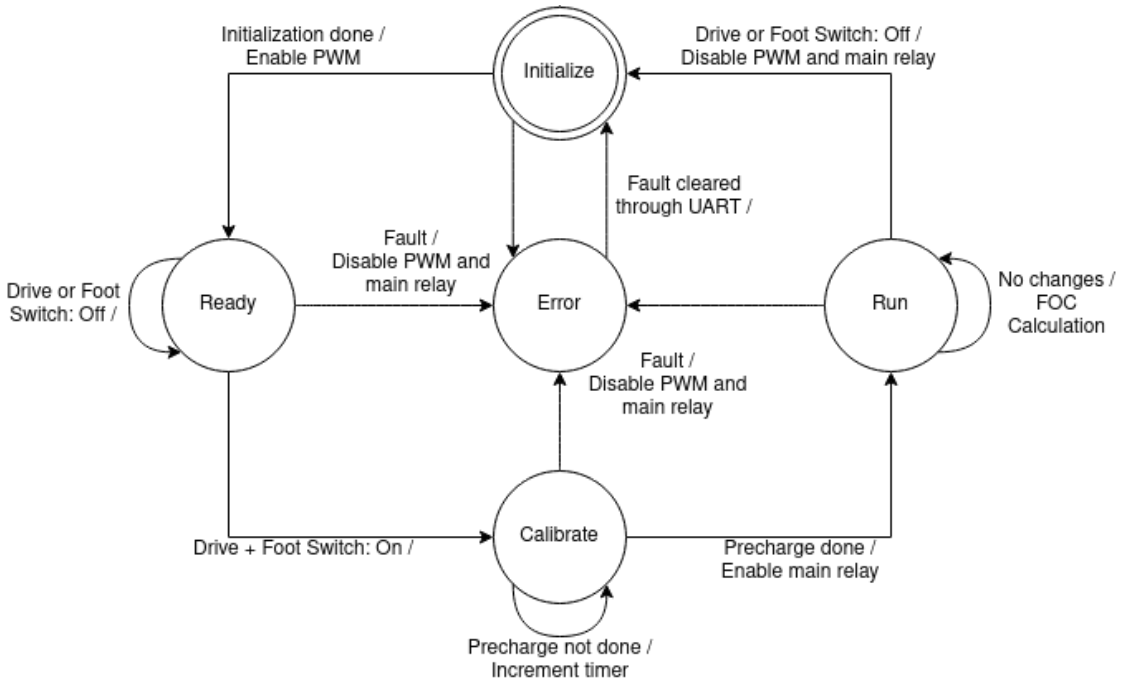


Figure 34: State machine used by processing system software for controlling the go-kart

17.3 Field-Oriented Control Task

The FOC task starts by checking for any fault by checking the contents of the errors_t struct. If any faults have occurred at any point, the state is instantly set to Error, disabling the main relay and PWM. The error state can only be left if a ”Clear Faults” message is received through UART, or the system is reset.

Afterwards, the state machine is run. The state diagram is shown in Figure 34.

The Initialize state is a one-pass that disables all relays, clears errors and variable values, resets PI controllers, and enables the PWM with a duty cycle of 0. Once left, it can only be re-entered after clearing a fault or ending the Run state. The state will always lead to the Ready state once finished.

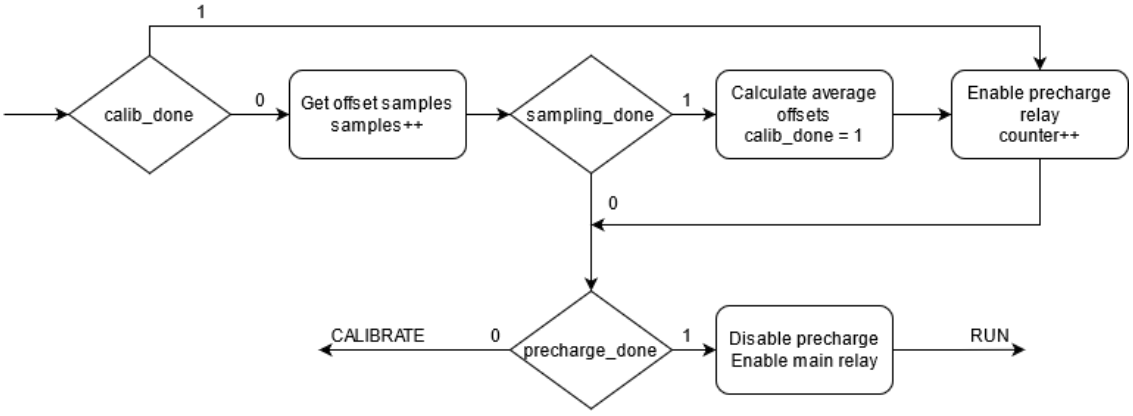


Figure 35: Execution flow in the Calibrate state of the field-oriented control task

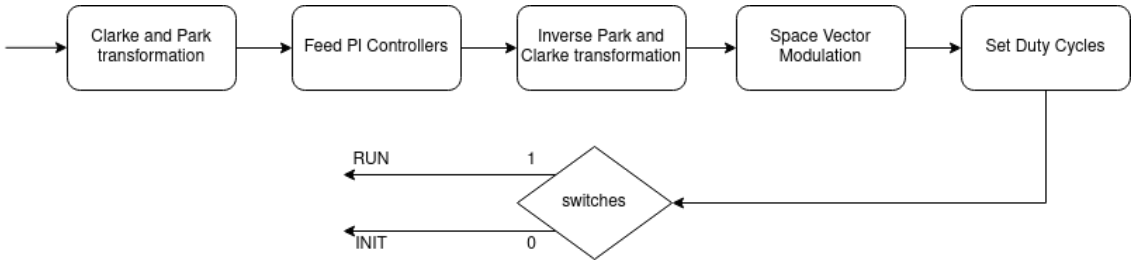


Figure 36: Execution flow in the Run state of the field-oriented control task

In the Ready state, the enable switch and foot switch are checked. If both are enabled, the state is changed to Calibrate, while it remains in the Ready state if not.

When in the Calibrate state, all duty cycles are set to zero to avoid any fluctuation in currents and voltage. All analog values are sampled across multiple interrupts, and the average value is used as offset by saving them in the measurement_t struct. When the calibration finishes, the pre-charge relay is enabled to charge the DC-link capacitors as specified in Section 10. Afterwards, the main relay is enabled, and the pre-charge relay disabled. The state is then changed to Run. The program flow in the Calibrate state is shown in Figure 35.

The Run state performs FOC using Clarke and Park transformations and the PI controller as specified in Section 12. The torque sensor's physical measurement is used as target for the PI controller, while controller outputs are limited to the battery voltage measurement. The PI controller outputs are then inverse transformed and space vector modulation is used to calculate duty cycles. The state flow is shown in Figure 36, while the code used is shown in Listing 7.

```
// Transform the measured currents to the dq plane
clarkeTransform(&currentsABC, &currentsAlphBe);
parkTransform(&currentsAlphBe, position.th_el, &currentsDQ);

// Limit PI controller to battery voltage and calculate output
qController.limit = battery_voltage.phys;
dController.limit = battery_voltage.phys;

err_d = -currentsDQ.arg1;
err_q = torque.phys - currentsDQ.arg2;
newSample(&dController, err_d, &voltagesDQ.arg1);
newSample(&qController, err_q, &voltagesDQ.arg2);

// Transform the calculated dq voltages to three phase voltages
invParkTransform(&voltagesDQ, position.th_el, &voltagesAlphBe);
invClarkeTransform(&voltagesAlphBe, &voltagesABC);

// Perform Space Vector Modulation and set OCV values
spaceVectorModulation(&voltagesABC, battery_voltage.phys,
                      &ocvvalues);
setDutyCycles(&ocvvalues);
```

Listing 7: Code used in Run state for implementation of field-oriented control

17.4 Communication Task

The Communication task is run in the main loop. The task provides a User Interface (UI) through the UART, allowing the PC user to access a variety of data at once. The terminal main menu is shown in Figure 37. It allows the user to see state machine data, motor errors, PI controller values, calculated physical sensor values, and current duty cycles. At high speed, many of these values will change much faster than the UART can transmit, and it is thus more useful for debugging if the motor is stuck.

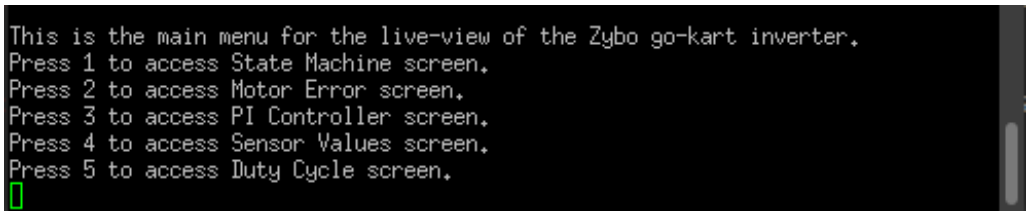


Figure 37: Main menu of UART Communication task shown in PC terminal

The PI controller screen allows changing the various controller values on-the-fly. The values are specified by five digits from 00000 to 99999. The value is then divided by 1000, giving a range of 0.000 to 99.999. The PI controller menu is shown in Figure 38, in which variable $a0$ and $a1$ for the q-direction controller have been changed by sending values '12345' and '56789' respectively.

Figure 38: PI controller menu of UART Communication task shown in PC terminal

Subconclusion

In this part, a PWM generator and an encoder driver were designed as AXI modules in PL. HALs and helper functions have been written in C for the PS, and are used by the Sensor Processing task and Field-Oriented Control task to calculate duty cycles. A state machine is used to follow start-up protocol and ensure safety.

Variables can be retrieved and adjusted from a PC through UART communication.

Part V
Testing

The designed hardware and software must be tested to ensure that it can be used for the go-kart motor under the circumstances designed for. A list of instruments used for testing is shown in Table 7.

Instrument	Name	Serial Number
Power Supply	TTi EL302RD	437661
Power Supply	TTi EL302RD	436442
Power Supply	EA-PS 9080-100	1017620003
Multimeter	Keysight 34410A	MY5301888
Oscilloscope	Teledyne Lecroy Wavesurfer 3054	LCRY3705N16458

Table 7: Instruments used during testing of the motor controller

18 Inverter

In order to test the inverter, the PCBs are manufactured, and the inverter is assembled with DC-link and heat sink as shown in Figure 39. The bill of materials for all PCBs are shown in Appendix A, with a total price of 2307.97 DKK.

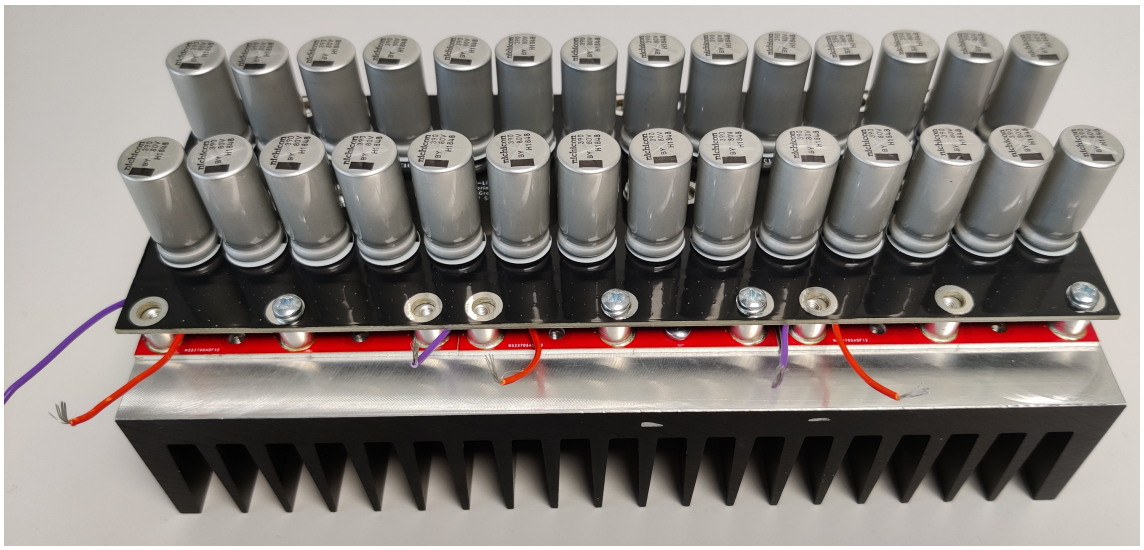
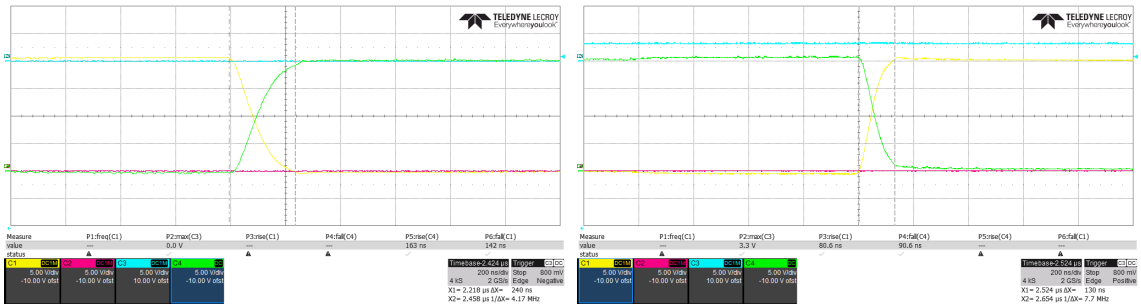


Figure 39: Heat sink, inverter boards, and DC-link as assembled for testing

18.1 Inverter Switching Characteristics

In this section the switching performance of the inverter will undergo several tests in order to verify the design and look for adjustments of the gate driver circuitry.

In Subsection 9.2, the gate resistors were dimensioned for a low-side MOSFET turn-on time of $t_{turn\_on} = 237\,\text{ns}$ and a turn-off time of $t_{turn\_off} = 50.05\,\text{ns}$. Looking at the transi-



(a) V_{DS} fall time measured to 240 ns (b) V_{DS} rise time measured to 130 ns

Figure 40: V_{DS} rise and fall times of the MOSFETs in the inverter. Yellow: low-side MOSFET, green: high-side MOSFET, blue: PWM

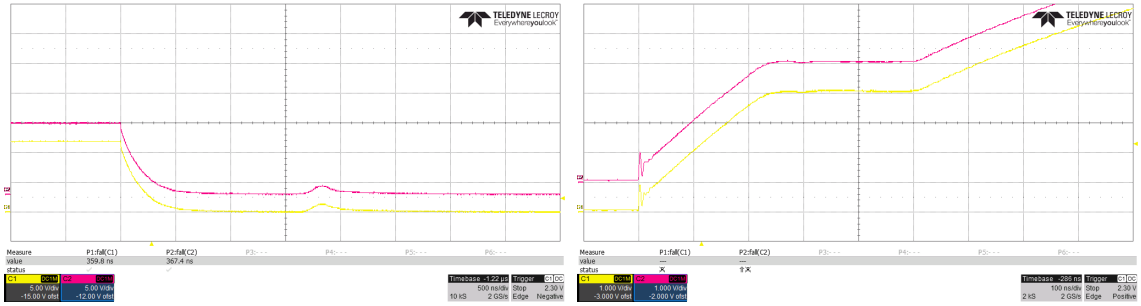
tions of V_{DS} of the low-side MOSFET, the switching times can be measured as shown in Figure 40.

The V_{DS} fall time, measured to 240 ns, shows the turn-on time of the MOSFET and is consistent with the designed 237 ns. However, the measured rise time of 130 ns is significantly slower than the designed turn-off time of 50 ns. This can be due to parasitic effects in the gate path and the diode in the turn-off path, which have not been taken into consideration during the dimensioning of the resistors.

The next measurements will examine if the parallel MOSFETs are switching simultaneously, which is important to avoid large currents through a single MOSFET as mentioned in Subsection 9.3. The gate signals are measured during turn-on and -off and compared in Figure 41.

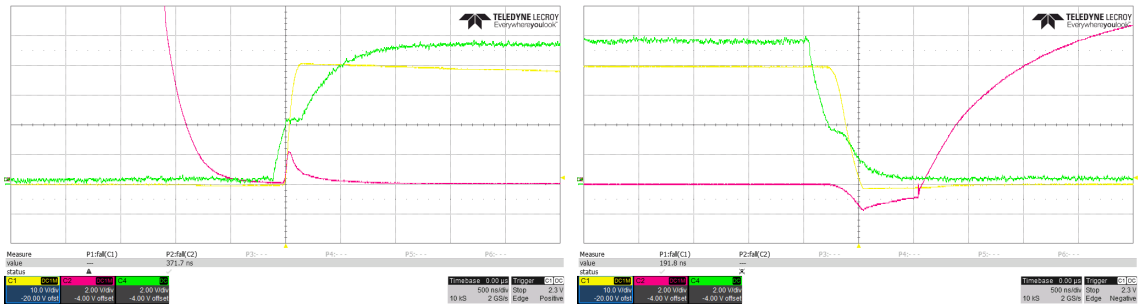
It is apparent from the figure that the MOSFETs are switching simultaneously.

Focusing on the switching characteristics of the inverter in Figure 42, it can be seen that the low-side MOSFET's gate-source signal increases when the high-side turns on. When operating the inverter with a load current of 3.1 A and an input voltage of 40 V,



(a) (b)

Figure 41: Gate-source signals of both the low-side MOSFETs. To be able to distinguish the signals, the red signal is shifted upwards. Yellow: FET closest to the gate driver, Red: FET furthest away from the gate driver



(a) (b)

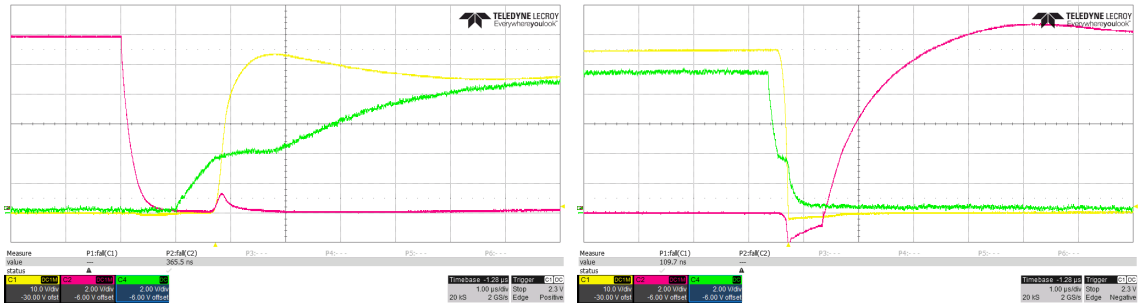
Figure 42: Switching signals of the inverter with input voltage of 40 V and load of 3.1 A. Yellow: Phase-GND voltage, red: low-side gate-source, green: high-side gate-source

the increase reaches 2.4 V. Considering the low load and that the threshold voltage for the MOSFET is 2.2 V - 3.8 V, the increase is deemed to large.

One solution is to increase the turn-on time of the high-side FETs as it will reduce the responsible $\frac{dV}{dt}$. Increasing R4 from 22 Ω to 68 Ω results in Figure 43. It is now possible to increase the input voltage to 57.6 V, which corresponds to the maximum battery voltage for endurance. The gate-source signals are measured with a load current of 8.2 A and the increase reaches 1.4 V. This is more promising as only the load current will be increased further. However, the maximum load current is significantly larger than the current during tests. It was not possible to test the inverter at greater currents.

The consequences of increasing the turn-on time for the high-side FETs can be seen on the phase output voltage of Figure 43. The phase output is measured between the phase and ground and has not yet settled after 6 µs. Therefore, the large increase in gate resistor is not the optimal solution.

It is also evident from the figures that the deadtime can be reduced to improve efficiency. However, this step should be done lastly to avoid crossconduction when adjusting the gate circuitry.



(a) (b)

Figure 43: Switching signals of the inverter with input voltage of 57.6 V and load of 8.2 A. Yellow: Phase-GND voltage, red: low-side gate-source, green: high-side gate-source

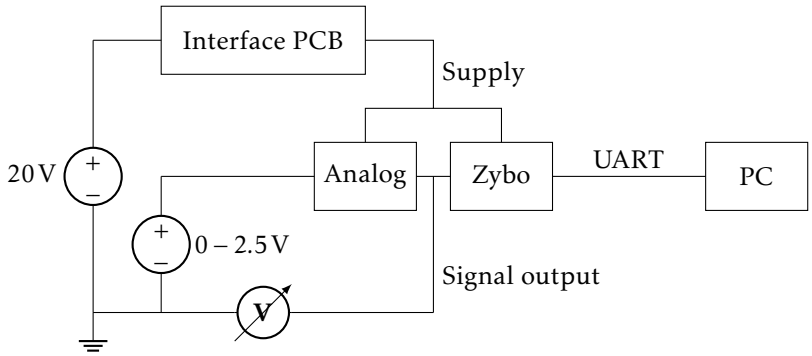


Figure 44: Test setup used for testing the analog interface PCB

19 Analog Interface Board

For testing the analog board, the general interface board must be powered, which in turn powers both the Zybo and analog interface board. Additionally, a power supply must be used for the various signal inputs to the board. The outputs are measured with a multimeter. The test setup can be seen in Figure 44.

19.1 Phase Current Measurement

The accuracy of the phase current measurements are of interest, and the circuits must thus be tested by applying known voltages and logging the converted values through UART. In the initial test, no voltage is applied, and the output is thus expected to be the offset of 0.5 V. The results for the phase current circuits is shown in Table 8. The individual samples can be seen in Appendix G. The samples show that the offset is actually 0.509 V and 0.505 V respectively due to component tolerances.

Phase A, N = 5	**Voltage [V]**	**Converted**	**Expected**	**Deviation**
Mean (μ)	0.509	2065	2048	17
Std. deviation (σ)	0	0.63	0	-
Phase B, N = 5	**Voltage [V]**	**Converted**	**Expected**	**Deviation**
Mean (μ)	0.505	2051.2	2048	3.2
Std. deviation (σ)	0	0.75	0	-

Table 8: Measured voltages and converted values for phases with no applied voltage

The deviation between the converted and expected values with no voltage applied is saved as an offset error and subtracted from future samples in the PS as described in Subsection 17.2. For this reason, it will also be subtracted in the coming tests.

A gain error also exists within the circuits, which cannot be dynamically removed. To check it, a voltage of 2.44 V and -2.44 V is applied to the circuits, with the output voltage and conversions shown for the current phases Table 9.

Both circuits show a larger deviation when the positive voltage is applied, with the calculated current being up to -3.2 A incorrect.

Phase A, N = 5	**Voltage [V]**	**Converted**	**Expected**	**Deviation**
Mean (μ)	0.752	3040.2	3062	-20.2
Std. deviation (σ)	0	1.72	0	-
Mean (μ)	0.264	1064.8	1064	0.8
Std. deviation (σ)	0	0.75	0	-
Phase B, N = 5	**Voltage [V]**	**Converted**	**Expected**	**Deviation**
Mean (μ)	0.749	3044.8	3063.8	-19
Std. deviation (σ)	0	0.75	0	-
Mean (μ)	0.261	1056.8	1065.8	-9
Std. deviation (σ)	0	0.75	0	-

Table 9: Measured voltages and converted values for phases with applied voltages

19.2 Torque Measurement

Following the same procedure as for the current measurement, the torque circuit is tested. Initially, the input is shunted to ground, simulating an untouched pedal. This is consistently converted to 0, and no offset is thus present in the active filter. When leaving the pin unconnected, the voltage rises to 1.52 V, which registers as 4095 in the XADC. Two additional voltage levels are applied to check the accuracy and gain error, shown in Table 10. They show a large negative deviation, which may impact the target I_q by up to -2.95 A. This means that the torque pedal must be pressed further for an identical target, potentially reducing maximum torque.

The individual samples used for the measurement can be found in Appendix G.

0.88 V applied, N = 5	**Voltage [V]**	**Converted**	**Expected**	**Deviation**
Mean (μ)	0.88	3579.8	3604	-24.2
Std. deviation (σ)	0	0.75	0	-
0.48 V applied, N = 5	**Voltage [V]**	**Converted**	**Expected**	**Deviation**
Mean (μ)	0.48	1927.2	1966	-38.8
Std. deviation (σ)	0	0.75	0	-

Table 10: Measured voltages and converted values for torque circuit

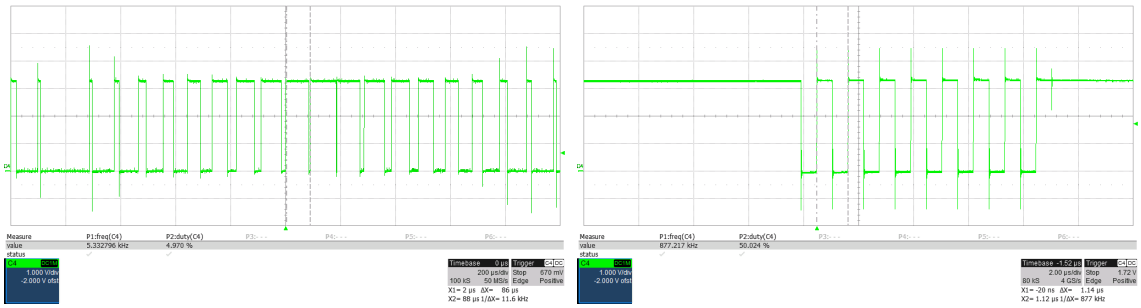
20 Embedded System

The code written in VHDL and C must be tested to ensure correct functionality. The outputs from the Zybo board are measured with an oscilloscope.

20.1 AXI Modules

The PWM generator module is tested by writing a C application increasing or decreasing the duty cycle by 10% with consistent intervals. This uses the AXI interface driver for communicating between PS and PL. The resulting measurement on Pmod pin JD2 is shown in Figure 45a. The figure shows a duty cycle ranging from 0 to 100%, with the switching frequency being 10.6 kHz as expected.

As the encoder was not accessible, the entire driver functionality was not possible to test. The serial clock output, as well as state transitions, were tested. A measurement of



(a) Oscillogram of one PWM output (b) Oscillogram of encoder serial clock

Figure 45: Oscillograms from tests of AXI modules

the serial clock on Pmod pin JB3 is shown in Figure 45b, which shows a frequency of 877 kHz, close to the 899 kHz expected.

By applying either 0 V or 3.3 V to A, B, and Z on Pmod pin JB2, JB1, and JB7 respectively, the remaining state transitions can be tested. The position is continuously updated by the encoder driver and read by the PS through the AXI interface. The test confirms that a Z pulse resets the position, while the position is incremented or decremented as expected when A and B change.

20.2 Execution Timing

As the XADC sampling time must be consistent for the FOC to function correctly, this must be tested. The execution time for the Sensor Processing and Field-Oriented Control tasks in the Run state are also important, as they must be completed within one sampling period. The measurements are acquired by enabling a GPIO at the beginning of the XADC ISR, and disabling it at the end. The resulting measurement is shown in Figure 46. The interrupt frequency is 10.6 kHz, while the entire ISR is executed in 14 µs when performing field-oriented control with space vector modulation. The frequency is as expected, and the tasks are executed much faster than required.

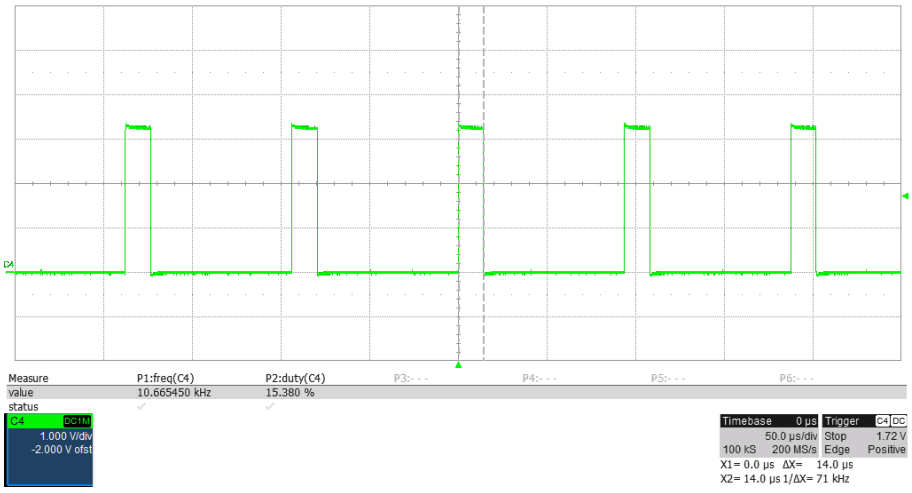


Figure 46: Oscillogram showing timing of XADC ISR

Part VI
Discussion

In this section, a summarizing discussion of the project will be made. The discussions will build on the findings throughout the project and on the test results.

Inverter

The inverter utilises two MOSFETs in parallel for each switch to reduce conduction loss compared to a single MOSFET. As could be seen in Subsection 9.1, these parameters could be reduced further by increasing the number of FETs in parallel. This would increase performance slightly, but it would come at a cost of increased price and greater requirements for the gate driver and gate driver circuitry. The number of MOSFETs in parallel can be revised if testing shows that too high temperatures occur. Due to the limited laboratory time these tests has not been performed.

Section 18 revealed an issue with the low-side gate-source signal that potentially can cause crossconduction. Slowing down the turn-on time of the high-side FET helped, but created another issue on the phase voltage. Due to time constraints, the adjustments were not finalised. A solution could be to add a transistor to create a shorter path with less resistance between the gate and ground. The increased performance might outweigh the cost of adding additional components.

Changing the gate driver to one with greater sinking capabilities can also help on this matter. Having a dedicated driver for each side might increase the currents that charges and discharges the gate capacitances. However, this will increase price.

A bootstrap capacitor is used as the floating supply for the high-side MOSFET. As the capacitor has to be charged, switching must occur and the duty cycle cannot be in the extremes. The on-resistance of the gate driver's internal bootstrap diode limits the current significantly, which might affect the duty cycle limits. Different duty cycles have not been tested, so the scope of the problem has not been revealed. To resolve this potential issue and better utilise the full duty cycle spectrum, an external diode can be added with a small resistor to limit the charging current or the gate driver can be changed to one that works with a supply for the floating high-side and with greater sink capabilities.

DC-Link

The DC-link consists of 27 electrolytic capacitors. The total number of capacitors can be brought down by changing to a hybrid capacitor DC-link combining film capacitors and electrolytic capacitors. Theoretically, the DC-link could consist of six electrolytic capacitors and a small number of film capacitors. The electrolytic capacitors generally have a larger capacitance and keep the voltage ripple low, while the film capacitors can handle the current stress. However, to determine how the current will be distributed, the

ESRs of the capacitors must be known. This requires that the ESR of the film capacitor is measured, which was not possible during the design process. Therefore, the hybrid capacitor DC-link was not implemented.

Analog Interface Board

Tests have been performed in Section 19, where a value measured by the multimeter was compared to the XADC measurements. The measurements are accurate enough for the application, but a large negative deviation from the expected value is noticed. This issue should be further explored to find the cause, and could be resistance in the connections, or sampling noise on the Zybo board itself due to lack of capacitance.

The standard deviations are consistently low for all measurements, indicating that atmospheric noise does not impact the measurements. Measurements have not been taken when switching large currents in the inverter, and the impact of switching noise on the signal stability is thus unknown. It may be necessary to implement digital filtering, which can be done both with digital signal processors in the programmable logic, or in the processing system code.

The accuracy of the calculated currents, as well as the inaccuracy's impact on system performance, when using the actual current transducers should be tested.

Control

The PI controller used to control the torque of the motor was designed using the pole placement method, where a settling time for the q-direction current of 50 ms was chosen. As no real world testing has been performed, it is difficult to know whether this settling time is appropriate. The simulations in Section 15 revealed that the speed of the car is mostly dependent on the much slower mechanical system. However, since the inertia and friction coefficient of the go-kart was not known, the simulation does not reveal how the vehicle actually behaves, and the settling time may need to be revised. Furthermore, there is a quantization of the measured angle due to the encoder resolution, which could impact the angle of the magnetic field, possibly reducing torque. The scope of which is not known, as it was not implemented in the simulation.

Embedded System

Some parts of the embedded system were tested in Section 20. The measurements show signals that are as expected, with some noise. Both the PWM and encoder clock signals are digital, meaning noise is unlikely to impact performance. The PWM signals are also filtered lightly before the gate driver, further reducing impact of noise.

The field-oriented control is performed at the expected frequency of 10.66 kHz. The control is executed in 14 µs, which is much faster than required with a switching frequency of 10.66 kHz, equating to a period of 94 µs. This shows that the switching frequency can be increased significantly without overburdening the processor.

Part VII
Conclusion

The project revolved around designing a motor controller for an electric go-kart. As part of this controller, a three-phased inverter with two parallel MOSFETs was designed and implemented on an aluminium PCB. The inverter was designed to be able to drive the motor with up to 220 A RMS of phase current and from a battery with maximum voltage of 57.6 V. The inverter was tested with low-current resistive loads to verify the design. Furthermore, the power loss was verified by a PLECS simulation, and calculations showed that the junction temperature of the MOSFETs would not increase above approximately 74 °C during full load operation. The total cost of the project was 2307.97 DKK, well within the budget.

A DC-link was designed to handle current stress due to the inverter and keep the ripple of the supply voltage for the inverter below 1 %.

A control system was designed utilising field-oriented control and a proportional-integral controller, ensuring that the motor could deliver controlled torque without steady-state error and with a settling time of approximately 44 ms.

An analog interface board was designed for handling the phase current and torque pedal measurements. The accuracy of these measurements was tested with the Zynq's ADC, showing a not-insignificant deviation of up to 21 LSB for the currents and 39 LSB for the torque, requiring further investigation.

Modules were designed for the Zynq's programmable logic in VHDL for generating the inverter PWM signals and for interfacing with the encoder. The PWM generator was designed for a switching frequency of 10.66 kHz, which tests confirmed to be accurate. The encoder driver could not be fully tested, but initial testing showed that all interactions function as expected.

A program was written for the Zynq's processing system in C code, which was responsible for a state machine for start-up of the go-kart. This included field-oriented control using space vector modulation to calculate the PWM duty cycles. The processing system also handled UART communication with a PC, providing an in-terminal UI for getting and setting important variable values within the program.

Further testing is required to ensure that the inverter and control works as intended during various load scenarios and when inserted into the go-kart.

List of Abbreviations

Abbreviation	Description
ADC	Analog-to-Digital Converter
AUX	Auxiliary ADC Channel
AXI	Advanced eXtensible Interface
CMRR	Common-Mode Rejection Ratio
EOS	End-of-Sequence
FET	Field-Effect Transistor
FOC	Field-Oriented Control
FPGA	Field-Programmable Gate Array
FPU	Floating-Point Unit
HAL	Hardware Abstraction Layer
LSB	Least Significant Bit
IO	Input/Output
IP	Intellectual Property
ISR	Interrupt Service Routine
MOSFET	Metal-Oxide-Semiconductor FET
MPU	Microprocessor
PCB	Printed Circuit Board
PL	Programmable Logic
PMAC	Permanent Magnet AC
PS	Processing System
PWM	Pulse-Width Modulation
RAM	Random-Access Memory
SVM	Space-Vector Modulation
UI	User Interface
XADC	Mixed-signal ADC
ZOH	Zero-Order Hold

Table 11: List of Abbreviations

List of Symbols

Symbol	Unit	Description
A, B, Z	-	The outputs of the encoder
aX, bX	-	Constants with indicator X
$bits_x$	-	Binary value with indicator x
C_x	F	Capacitance or capacitor with indicator x
$cos(\phi)$	-	Power factor
D_x, d_x	-	Duty cycle with indicator x
f_{sw}	Hz	Switching frequency
$G_x(s)$	-	Transfer function with indicator x in the s-domain
h	m	Height or thickness
I_x, i_x	A	Current with indicator x
J	Kg*m^2	Moment of inertia
K_x	-	Gain with indicator x
k	$\frac{\mathrm{W}}{\mathrm{m \cdot K}}$	Thermal conductivity
L_x	H	Inductance or inductor with indicator x
l	m	Length
M	-	Modulation index
N	-	Amount
n	-	Order
p_x	%	Percentage with indicator x
pos_x	rad/s	Rotational position with indicator x
Q_x	C	Charge with indicator x
R_x	Ω	Resistance, thermal resistance, or resistor with indicator x
t_x	s	Time period with indicator x
T_x	°C	Temperature with indicator x
V_x, v_x	V	Voltage with indicator x
w	m	Width
Θ_x	rad/s	Angle with indicator x
λ_{MAG}	Vs/rad	Flux linkage
τ_x	s or Nm	Time constant or torque with indicator x
φ	°	Phase shift
ω_x	rad/s	Rotational speed with indicator x

Table 12: List of symbols used

Bibliography

[1] Motenergy. Me1117 permanent magnet ac motor, April 2012. http://www.motenergy.com/me1117.html, last accessed 30/03-2021.

[2] RLS. Rmb28 / rmf44 angular magnetic encoder modules, May 2020. https://www.rls.si/eng/fileuploader/download/download/?d=1&file=custom%2Fupload%2FRMB28D01_16_EN_data_sheet.pdf, last accessed 30/03-2021.

[3] Digilent. Zybo fpga board reference manual, February 2017. https://reference.digilentinc.com/_media/reference/programmable-logic/zybo/zybo_rm.pdf, last accessed 30/03-2021.

[4] Internation Rectifier. Igbt or mosfet: Choose wisely. https://www.infineon.com/dgdl/Infineon-IGBT_or_MOSFET_Choose_Wisely-Article-v01_00-EN.pdf?fileId=5546d462533600a40153574048b73edc, last accessed 26/04-2021.

[5] Infineon. OptiMOS™-5 Power-Transistor, May 2020. https://www.infineon.com/dgdl/Infineon-IAUT300N08S5N012-DS-v01_00-EN.pdf?fileId=5546d46258fc0bc10158fdabe0f90580, last accessed 04/05-2021.

[6] Texas Instruments. 48-vdc battery powered inverter power stage reference design for 5-kw forklift ac traction motor, Oct. 2016. https://www.ti.com/lit/ug/tiducb6/tiducb6.pdf, last accessed 26/04-2021.

[7] PCBway. Aluminum pcb, 2014. https://www.pcbway.com/blog/Engineering_Technical/Aluminum_PCB.html, last accessed 20/05-2021.

[8] Engineering Toolbox. Thermal conductivity of common metals, metallic elements and alloys, 2005. https://www.engineeringtoolbox.com/thermal-conductivity-metals-d_858.html, last accessed 17/05-2021.

[9] Fischer Elecktronik. Data sheet product sk 47, 2021. https://www.fischerelektronik.de/web_fischer/en_GB/PR/SK47_/datasheet.xhtml, last accessed 20/05-2021.

[10] Engineering Toolbox. Specific heat of some metals, 2005. https://www.engineeringtoolbox.com/specific-heat-metals-d_152.html, last accessed 18/05-2021.

[11] ST. High voltage high and low-side 2 A gate driver, Nov. 2017. https://www.st.com/resource/en/datasheet/l6494.pdf, last accessed 20/04-2021.

[12] PCBWay. General introduction of Aluminum PCB. https://www.pcbway.com/pcb_prototype/General_introduction_of_Aluminum_PCB.html, last accessed 22/04-2021.

[13] Thomas M. Wolbank Johann W. Kolar and Manfred Schrödl. Analytical Calculation of the RMS Current Stress on the DC Link Capacitor of Voltage DC Link PWM Converter Systems. *Ninth International Conference on Electrical Machines and Drives*, 468:81–89, 1999.

[14] Nichicon. Aluminum Electrolytic Capacitors. https://www.mouser.dk/datasheet/2/293/UBY_e-1280407.pdf, last accessed 04/05-2021.

[15] LEM. Current transducer lf 205-s, July 2020. https://www.lem.com/sites/default/files/products_datasheets/lf_205-s.pdf, last accessed 04/04-2021.

[16] Analog Devices. Wide Supply Range, Micropower, Rail-to-Rail Instrumentation Amplifier, Jan. 2015. https://www.mouser.dk/datasheet/2/609/AD8420-1502327.pdf, last accessed 14/03-2021.

[17] Texas Instruments. Analysis of the Sallen-Key Architecture, September 2002. https://www.ti.com/lit/an/sloa024b/sloa024b.pdf, last accessed 03/01-2021.

[18] Microsemi. Park, Inverse Park and Clarke, Inverse Clarke Transformations MSS Software Implementation User Guide, 2013. https://www.microsemi.com/document-portal/doc_view/132799-park-inverse-park-and-clarke-inverse-clarke-transformations-mss-software-implementation-user-guide, last accessed 17/05-2021.

[19] Ned Mohan. *Power Electronics: A First Course*. John Wiley & Sons, 2012.

[20] NXP. GFLIB User's Guide, Dec 2020. https://www.nxp.com/docs/en/user-guide/CM4GFLIBUG.pdf, last accessed 16/05-2021.

Part VIII
Appendices

A Bill of Materials

The components for the three PCBs are listed in this appendix, with a total price for all phases listed at the bottom of each table. The total price for all components amounts to 2307.97 DKK.

Analog Interface Board

Designation	Description	Value	Unit Price [DKK]
R2, R8	1% tol., SMD 1210	54.9 Ω	1.46
R4, R10	1% tol., SMD 1210	56 Ω	0.95
R5, R11	1% tol., SMD 0603	3.09 Ω	0.64
R1, R3, R7, R9	1% tol., SMD 0603	20 kΩ	0.64
C2, C4, C7, C9	5% tol., 50 V, SMD 0603	100 pF	0.26
C3, C8	5% tol., 50 V, SMD 0603	1 nF	0.57
C1, C5, C6, C10, C15	X7R, 50 V, SMD 0603	100 nF	0.26
U1, U2	AD8420ARMZ, In Amp, high CMRR, SMD	-	18.00
C11, C13, C14	X5R, 25 V, SMD 0805	10 µF	1.39
R13	1% tol., SMD 0603	5.1 kΩ	0.64
R14	1% tol., SMD 0603	1.43 kΩ	0.64
R20	1% tol., SMD 0603	1 kΩ	0.64
D4	LM4041CFTA, reference, SMD SOT-23-3	1.225 V	4.24
J301, J302, J304, J305	Standard 2x5 pin header	-	4.3
R21, R22	1% tol., SMD 0603	270 Ω	0.64
R24, R19	1% tol., SMD 0603	10 kΩ	0.64
D5, D6	LED, SMD 0603	-	0.95
SW1	PTS645SK50SMTR92LFS, SPST button	-	0.95
R15	1% tol., SMD 0603	11.5 kΩ	0.64
R16, R17	1% tol., SMD 0603	7.5 kΩ	0.64
R18	1% tol., SMD 0603	147 Ω	0.64
U3	MCP6L01T-E/OT, Op Amp, SMD SOT-23-5	-	1.58
C16, C17	X7R, 16 V, SMD 0603	33 nF	0.64
C18	X7R, 25 V, SMD 0603	56 nF	0.76
PCB	Two-layer, 1 oz., 5 pcs.	-	30.55
Total			166.88

Table A.1: Components used for the analog interface board

DC-Link Board

Designation	Description	Value	Unit Price [DKK]
C1-C26	Aluminium, 80 V, 20%, 1640 mA ripple	390 µF	7.84
PCB	Two-layer, 1 oz., 5 pcs.	-	201.65
Total			431.70

Table A.2: Components used for the DC-link board

Inverter board

Designation	Description	Value	Unit Price [DKK]
-	7466203R, SMT M3 Thread	-	15.46
U1, U2, U3, U4	IAUT300N08S5N012ATMA2, 80 V, 300 A MOSFET	-	40.16
IC1	L6494 gate driver	-	9.00
-	SK 47/100 SA heat sink	-	240.85
D1, D2	2 A Schottky, SMD 0603	-	2.38
R1	SMD 0603	1 kΩ	0.64
R2	SMD 0603	10 kΩ	0.64
R3	SMD 0603	2 kΩ	0.64
R4, R6	SMD 0603	22 Ω	0.64
R5, R7, R12, R13	SMD 0603	1 Ω	0.64
R8, R9, R10, R11	SMD 0603	3.3 Ω	0.64
C1, C9, C15	SMD 1206	1 nF	0.95
C2, C4, C13	SMD 1206	100 nF	0.64
C3, C6, C12	SMD 1206	1 µF	0.95
C5	SMD 0603	100 pF	0.64
C7, C8	SMD 0603	33 nF	0.64
C10, C11, C16, C17, C18, C19	SMD 1206	4.7 µF	2.09
C14	SMD 1206	10 nF	1.02
PCB	1 layer, 1 W/(mK) alu., 10 pcs.	-	158.87
Total			1709.39

Table A.3: Components used for the inverter board

B Wiring Diagram for Current System

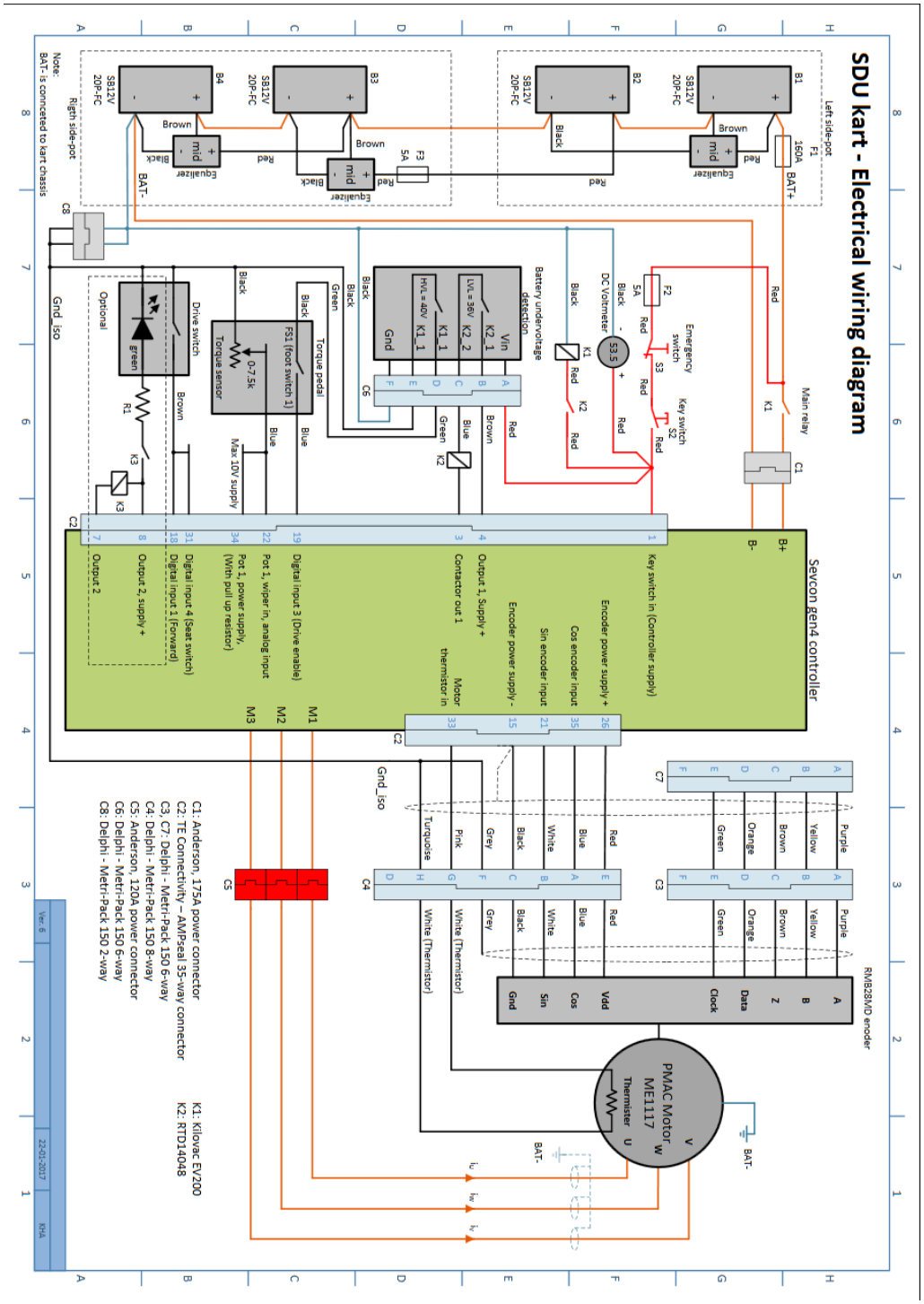


Figure B.1: Wiring diagram for the SDU go-kart with Sevcon Gen4 controller. From project description

C Simulink Implementation of Motor Model

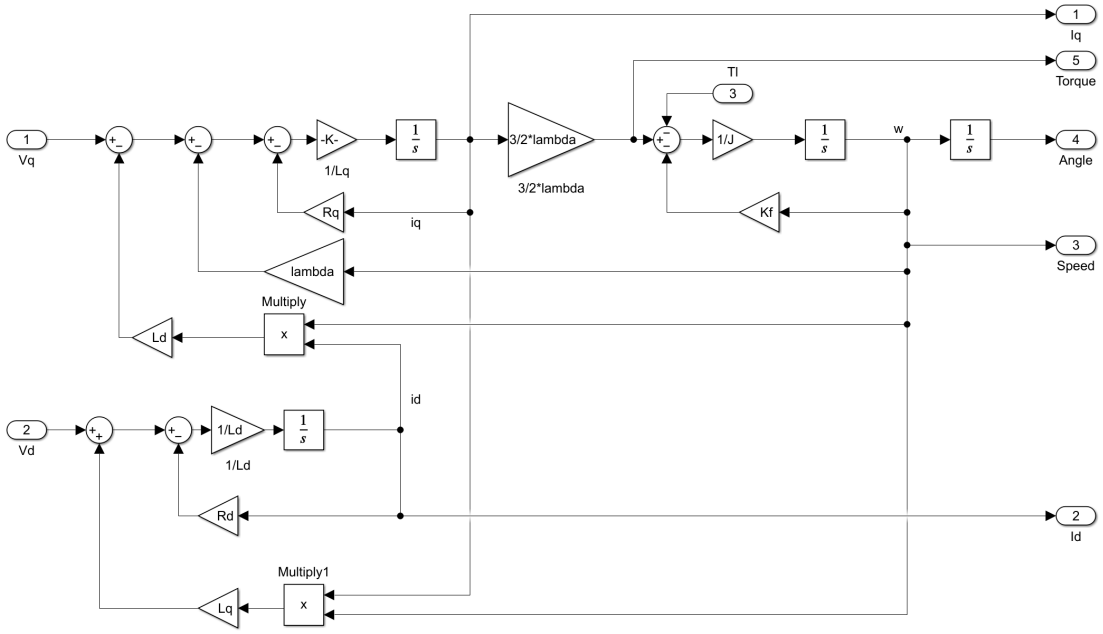


Figure C.1: Simulink implementation of dq motor model

D Simulink Implementation of Digital PI Controller

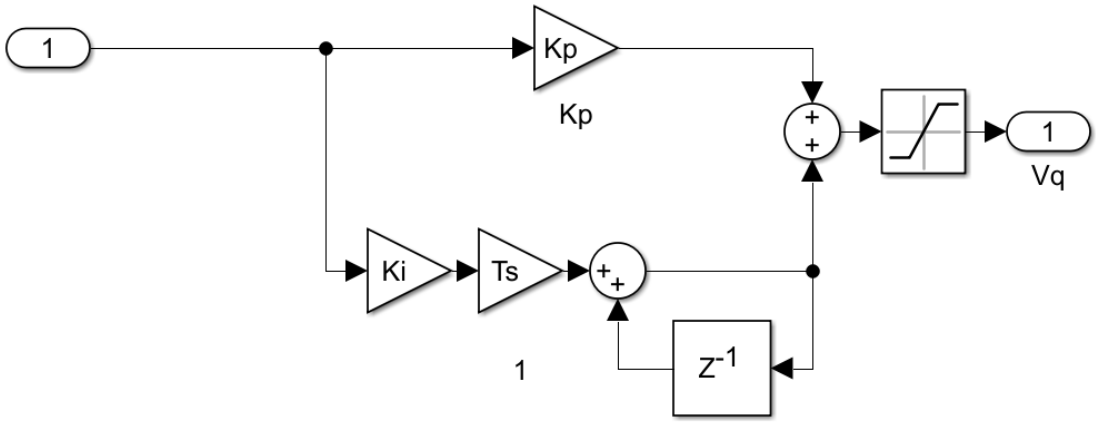


Figure D.1: Simulink implementation of digital PI controller

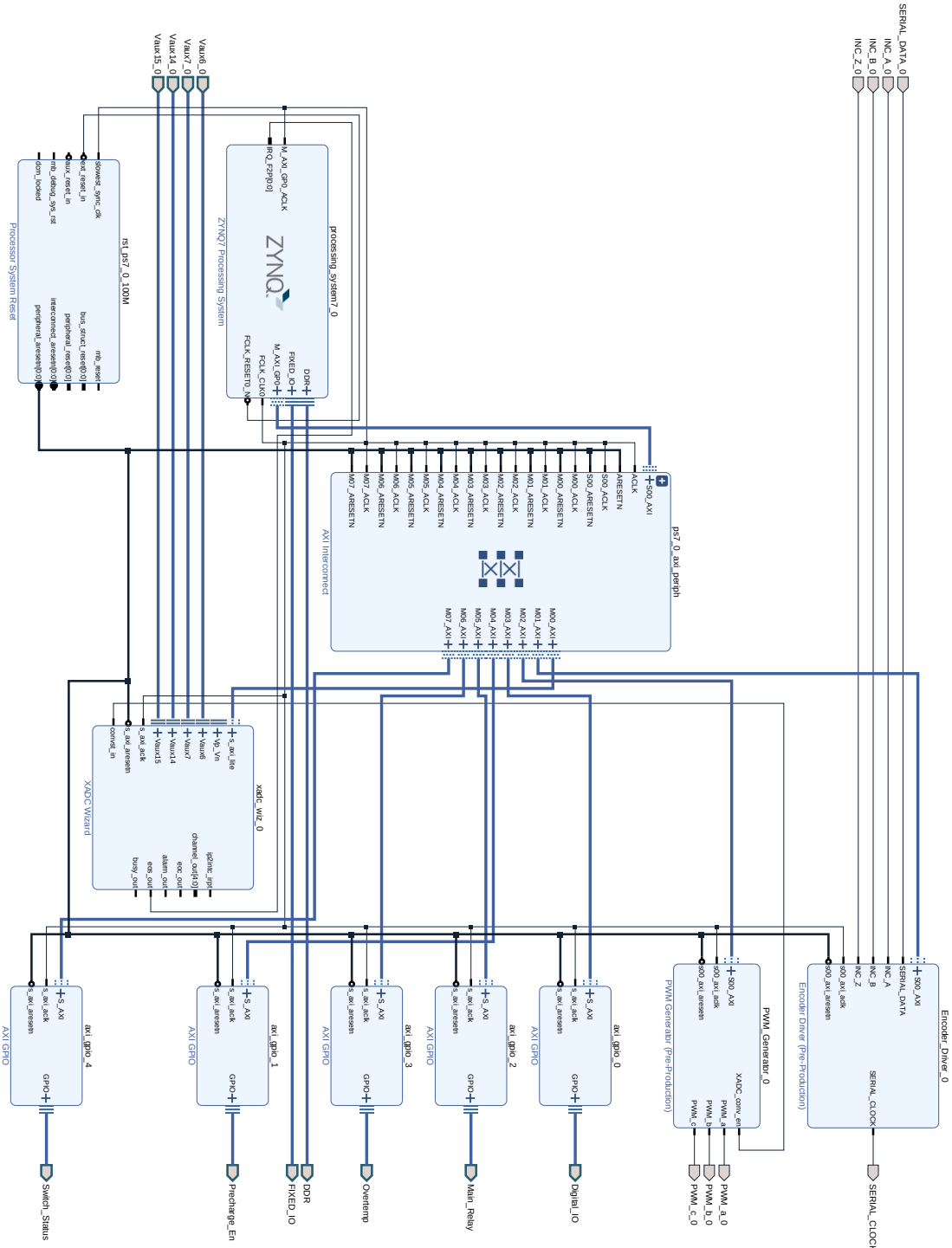
E Connections for Zybo board

Pmod Pin	Zynq Pin	Name	Description
JA1	N15 (AUX 14)	ADC 2	Battery voltage measurement
JA2	L14 (AUX 7)	ADC 1	Phase A current measurement
JA3	K16 (AUX 15)	ADC 4	Phase B current measurement
JA4	K14 (AUX 6)	ADC 3	Torque pedal measurement
JA5	-	GND	Ground
JA6	-	3.3 V	Supply
JA7	N16	GND	NC
JA8	L15	GND	NC
JA9	J16	GND	NC
JA10	J14	GND	NC
JA11	-	GND	Ground
JA12	-	3.3 V	Supply
JB1	T20	Enc B	Encoder B value
JB2	U20	Enc A	Encoder A value
JB3	V20	Enc Clk	Encoder clock
JB4	W20	Enc Data	Encoder data
JB5	-	GND	Ground
JB6	-	3.3 V	Supply
JB7	Y18	Enc Z	Encoder Z value
JB8	Y19	NC	NC
JB9	W18	NC	NC
JB10	W19	NC	NC
JB11	-	GND	Ground
JB12	-	3.3 V	Supply
JC1	V15	Dig 1	Digital I/O pin 1
JC2	W15	Dig 2	Digital I/O pin 2
JC3	T11	Dig 3	Digital I/O pin 3
JC4	T10	Dig 4	Digital I/O pin 4
JC5	-	GND	Ground
JC6	-	3.3 V	Supply
JC7	W14	Dig 5	Digital I/O pin 5
JC8	Y14	NC	NC
JC9	T12	NC	NC
JC10	U12	NC	NC
JC11	-	GND	Ground
JC12	-	3.3 V	Supply
JD1	T14	PWM 4	Transistor PWM 4

Pmod Pin	Zynq Pin	Name	Description
JD2	T15	PWM 3	Transistor PWM 3
JD3	P14	PWM 2	Transistor PWM 2
JD4	R14	PWM 1	Transistor PWM 1
JD5	-	GND	Ground
JD6	-	3.3 V	Supply
JD7	U14	PWM 5	Transistor PWM 5
JD8	U15	PWM 6	Transistor PWM 6
JD9	V17	NC	NC
JD10	V18	NC	NC
JD11	-	GND	Ground
JD12	-	3.3 V	Supply
JE1	V12	Pre-Charge	Pre-charge circuit activation
JE2	W16	Relay Control	Control of main relay
JE3	J15	Relay On	Status of main relay
JE4	H15	Overtemp	Motor overtemperature status
JE5	-	GND	Ground
JE6	-	3.3 V	Supply
JE7	V13	NC	NC
JE8	U17	Drive Enable	Status of enable switch
JE9	T17	Foot Switch	Status of foot switch
JE10	Y17	NC	NC
JE11	-	GND	Ground
JE12	-	3.3 V	Supply
JF1	E8	NC	NC
JF2	E9	NC	NC
JF3	C6	NC	NC
JF4	D9	NC	NC
JF5	-	GND	Ground
JF6	-	3.3 V	Supply
JF7	E6	NC	NC
JF8	B5	NC	NC
JF9	C5	NC	NC
JF10	C8	NC	NC
JF11	-	GND	Ground
JF12	-	3.3 V	Supply

Table E.1: Connections for Zybo board on interface board

F Vivado Block Diagram of Programmable Logic IP Cores



G Data Samples for Analog Board Measurements

Phase A	Voltage [V]	Converted
No voltage applied		
#1	0.509	2065
#2	0.509	2064
#3	0.509	2065
#4	0.509	2066
#5	0.509	2065
2.44 V applied		
#1	0.752	3043
#2	0.752	3039
#3	0.752	3038
#4	0.752	3041
#5	0.752	3040
-2.44 V applied		
#1	0.264	1065
#2	0.264	1065
#3	0.264	1064
#4	0.264	1066
#5	0.264	1064

Table G.1: Full list of data samples used for phase A circuit testing

Phase B	Voltage [V]	Converted
No voltage applied		
#1	0.505	2052
#2	0.505	2050
#3	0.505	2052
#4	0.505	2051
#5	0.505	2051
2.44 V applied		
#1	0.749	3045
#2	0.749	3044
#3	0.749	3044
#4	0.749	3046
#5	0.749	3045
-2.44 V applied		
#1	0.261	1057
#2	0.261	1056
#3	0.261	1058
#4	0.261	1056
#5	0.261	1057

Table G.2: Full list of data samples used for phase B circuit testing

Torque	Voltage [V]	Converted
0.48 V applied		
#1	0.48	1928
#2	0.48	1927
#3	0.48	1927
#4	0.48	1928
#5	0.48	1926
0.88 V applied		
#1	0.88	3580
#2	0.88	3579
#3	0.88	3581
#4	0.88	3579
#5	0.88	3580

Table G.3: Full list of data samples used for torque measurement circuit testing

Code Snippets

Encoder Driver

```
process (clk_scaled)
begin
    if rising_edge(clk_scaled) then
        if S_AXI_ARESETN = '1' and slv_reg1 = x"00000001" then
            case state is
                when GET_ABS =>
                    if (index = -1) then
                        position <= data_temp(DATA_LENGTH-1
                                              downto 0);

                        inc_AB := INC_A & INC_B;
                        case inc_AB is
                            when "00" => state <= ZERO;
                            when "01" => state <= ONE;
                            when "10" => state <= TWO;
                            when "11" => state <= THREE;
                            when others => state <= ZERO;
                        end case;
                    else
                        data_temp(index) <= SERIAL_DATA;
                        index <= index - 1;
                    end if;

                when ZERO =>
                    if INC_A = '1' then
                        position <= position - 1;
                        state <= TWO;
                    elsif INC_B = '1' then
                            position <= position + 1;
                            state <= ONE;
                    end if;

                when ONE =>
                    if INC_A = '1' then
                        position <= position + 1;
                        state <= THREE;
                    elsif INC_B = '0' then
                        position <= position - 1;
                        state <= ZERO;
                    end if;

                when TWO =>
                    if INC_A = '0' then
                        position <= position + 1;
                        state <= ZERO;
                    elsif INC_B = '1' then
                        position <= position - 1;
                        state <= THREE;
                    end if;

                when THREE =>
                    if INC_A = '0' then
                        position <= position - 1;
                        state <= ONE;
                    elsif INC_B = '0' then
```

```
                        position <= position + 1;
                        state <= TWO;
                    end if;
            end case;

            if INC_Z = '1' then
                position <= (others => '0');
            end if;

        else
            state <= GET_ABS;
            position <= (others => '0');
            index <= DATA_LENGTH;
        end if;
    end if;
end process;

slv_reg0 <= x"000000" & std_logic_vector(position);
```

Structs used for Motor Control

```
typedef struct errors_t {
  u8 motor_overtemp;
  u8 overvoltage;
  u8 undervoltage;
  u8 torque_disc;
  u8 phaseA_overcurr;
  u8 phaseB_overcurr;
} errors_t;

typedef struct measurement_t {
  u16 raw;
  u16 flt;
  f32 phys;
  s32 offset;
} measurement_t;

typedef struct statemachine_t {
  u8 state;
  u8 relay_status;
  u8 switches;
  u8 clear_fault;
} statemachine_t;

typedef struct ocvvalues_t {
  u16 ocv1;
  u16 ocv2;
  u16 ocv3;
} ocvvalues_t;

typedef struct encoder_t {
  u16 pos_raw;
  f32 th_el;
  f32 th_mech;
} encoder_t;

typedef struct three_phased_t {
  f32 arg1;
  f32 arg2;
```

```
  f32 arg3;
} three_phased_t;

typedef struct two_phased_t {
  f32 arg1;
  f32 arg2;
} two_phased_t;

typedef struct controllerPI_t {
  f32 a_0;
  f32 a_1;
  f32 b_1;
  f32 limit;
  f32 prev_sample;
  f32 prev_output;
} controllerPI_t;

errors_t motor_errors;
measurement_t battery_voltage;
measurement_t torque;
measurement_t phaseA;
measurement_t phaseB;
measurement_t phaseC;
statemachine_t statemachine;
ocvvalues_t ocvvalues;
encoder_t position;
three_phased_t currentsABC;
three_phased_t voltagesABC;
two_phased_t currentsAlphBe;
two_phased_t voltagesAlphBe;
two_phased_t currentsDQ;
two_phased_t voltagesDQ;
controllerPI_t dController;
controllerPI_t qController;
```

AXI Interface HAL

```
#define ENCODER_DRIVER_BASEADDR
        XPAR_ENCODER_DRIVER_0_S00_AXI_BASEADDR
#define PWM_GENERATOR_BASEADDR
        XPAR_PWM_GENERATOR_0_S00_AXI_BASEADDR

void readPosition(u16 *raw_pos) {
    *raw_pos = ENCODER_DRIVER_mReadReg(ENCODER_DRIVER_BASEADDR,
               ENCODER_DRIVER_S00_AXI_SLV_REG0_OFFSET);
}

void setDutyCycles(ocvvalues_t *ocvs) {
    PWM_GENERATOR_mWriteReg(PWM_GENERATOR_BASEADDR,
    PWM_GENERATOR_S00_AXI_SLV_REG0_OFFSET, ocvs->ocv1);
    PWM_GENERATOR_mWriteReg(PWM_GENERATOR_BASEADDR,
    PWM_GENERATOR_S00_AXI_SLV_REG1_OFFSET, ocvs->ocv2);
    PWM_GENERATOR_mWriteReg(PWM_GENERATOR_BASEADDR,
    PWM_GENERATOR_S00_AXI_SLV_REG2_OFFSET, ocvs->ocv3);
}
```

XADC HAL

```
#define TORQUE_CHANNEL 6
#define PHASE_A_CHANNEL 7
#define BATTERY_CHANNEL 14
#define PHASE_B_CHANNEL 15
#define LOWER_TWELVE_SHIFT 4

XAdcPs XAdc;

void getBatteryRaw(u16 *raw) {
  *raw = (XAdcPs_GetAdcData(&XAdc, XADCPS_CH_AUX_MIN +
          BATTERY_CHANNEL) >> LOWER_TWELVE_SHIFT);
}

void getTorqueRaw(u16 *raw) {
  *raw = (XAdcPs_GetAdcData(&XAdc, XADCPS_CH_AUX_MIN +
          TORQUE_CHANNEL) >> LOWER_TWELVE_SHIFT);
}

void getPhaseARaw(u16 *raw) {
  *raw = (XAdcPs_GetAdcData(&XAdc, XADCPS_CH_AUX_MIN +
          PHASE_A_CHANNEL) >> LOWER_TWELVE_SHIFT);
}

void getPhaseBRaw(u16 *raw) {
  *raw = (XAdcPs_GetAdcData(&XAdc, XADCPS_CH_AUX_MIN +
          PHASE_B_CHANNEL) >> LOWER_TWELVE_SHIFT);
}
```

Clarke and Park Transformation

```
#define RECI_SQRT_3 (f32)0.5774
#define SQRT_3_OVER_2 (f32)0.8660
#define ONE_THIRD (f32)0.33333333333333333333333333333
#define TWO_THIRDS (f32)0.66666666666666666666666666667

void clarkeTransform( three_phased_t *input,
                      two_phased_t *output ) {
  output->arg1 = TWO_THIRDS*input->arg1 - ONE_THIRD *
                 (input->arg2 - input->arg3);
  output->arg2 = RECI_SQRT_3*(input->arg2 - input->arg3);
}


void parkTransform( two_phased_t *input, f32 input_angle,
                    two_phased_t *output ) {
  output->arg1 = input->arg1 * taylor_cos(input_angle) +
                 input->arg2 * taylor_sin(input_angle);
  output->arg2 = input->arg2 * taylor_cos(input_angle) -
                 input->arg1 * taylor_sin(input_angle);
}


void invClarkeTransform( two_phased_t *input,
                         three_phased_t *output ) {
  output->arg1 = input->arg1;
  output->arg2 = -0.5*input->arg1 + SQRT_3_OVER_2*input->arg2;
  output->arg3 = -0.5*input->arg1 - SQRT_3_OVER_2*input->arg2;

```

```
}


void invParkTransform( two_phased_t *input, f32 input_angle,
                       two_phased_t *output ) {
  output->arg1 = input->arg1 * taylor_cos(input_angle) -
                 input->arg2 * taylor_sin(input_angle);
  output->arg2 = input->arg2 * taylor_cos(input_angle) +
                 input->arg1 * taylor_sin(input_angle);
}
```

Sensor Processing Task

```
#define TEN_SECONDS 100000
#define OVERVOLT_THRESHOLD 4025
#define UNDERVOLT_THRESHOLD 2300
#define DISCONNECT_THRESHOLD 3800
#define OVERCURRENT_THRESHOLD 3950
#define NEG_OVERCURRENT_THRESHOLD 4095-OVERCURRENT_THRESHOLD
#define BATTERY_CONVERSION
        (f32)0.015140415140415140415140415140415
#define TORQUE_CONVERSION
        (f32)0.030698684032017364842204187880 27
#define OVERCURR_CONVERSION
        (f32)0.084420024420024420024420024420024
#define CURRENT_CONVERSION
        (f32)0.158730158730158730158730158730 16
#define CURRENT_OFFSET (f32)325.0

u16 overcurrentswitch, relay, overtemp, enableswitch, footswitch
u32 overcurrent_timer;

void sensorProcessing() {
  // Read all digital sensor values
  checkOvercurrentSwitch(&overcurrentswitch);
  checkMainRelay(&relay);
  checkOvertemp(&overtemp);
  checkEnableSwitch(&enableswitch);
  checkFootSwitch(&footswitch);

  // If button is pressed, start overcurrent timer
  if (overcurrentswitch) {
    overcurrent_timer = TEN_SECONDS;
  }

  // Write overtemperature error
  motor_errors.motor_overtemp = overtemp;

  // Write values for state machine to check
  statemachine.relay_status = relay;
  statemachine.switches = (enableswitch & footswitch);

  // Read analog values from conversions
  getBatteryRaw(&battery_voltage.raw);
  getTorqueRaw(&torque.raw);
  getPhaseARaw(&phaseA.raw);
  getPhaseBRaw(&phaseB.raw);

  // Remove offset from all analog values
```

```
47    battery_voltage.flt = battery_voltage.raw -
48                          battery_voltage.offset;
49    torque.flt = torque.raw - torque.offset;
50    phaseA.flt = phaseA.raw - phaseA.offset;
51    phaseB.flt = phaseB.raw - phaseB.offset;
52
53    // Check for errors and calculate all physical values from
       analog values
54    motor_errors.overvoltage = (battery_voltage.flt >
       OVERVOLT_THRESHOLD - battery_voltage.offset) ? 1 : 0;
55    motor_errors.undervoltage = (battery_voltage.flt <
       UNDERVOLT_THRESHOLD - battery_voltage.offset) ? 1 : 0;
56    battery_voltage.phys = battery_voltage.flt *
57                           BATTERY_CONVERSION;
58
59    motor_errors.torque_disc = (torque.flt > DISCONNECT_THRESHOLD
       - torque.offset) ? 1 : 0;
60
61    if (overcurrent_timer) {
62      torque.phys = torque.flt * OVERCURR_CONVERSION;
63      overcurrent_timer--;
64    } else {
65      torque.phys = torque.flt * TORQUE_CONVERSION;
66    }
67
68    motor_errors.phaseA_overcurr =
69    ((phaseA.flt > OVERCURRENT_THRESHOLD - phaseA.offset) |
70    (phaseA.flt < NEG_OVERCURRENT_THRESHOLD - phaseA.offset))
71    ? 1 : 0;
72    motor_errors.phaseB_overcurr =
73    ((phaseB.flt > OVERCURRENT_THRESHOLD - phaseB.offset) |
74    (phaseA.flt < NEG_OVERCURRENT_THRESHOLD - phaseB.offset))
75    ? 1 : 0;
76    phaseA.phys = phaseA.flt*CURRENT_CONVERSION - CURRENT_OFFSET;
77    phaseB.phys = phaseB.flt*CURRENT_CONVERSION - CURRENT_OFFSET;
78    phaseC.phys = -phaseA.phys - phaseB.phys;
79
80    currentsABC.arg1 = phaseA.phys;
81    currentsABC.arg2 = phaseB.phys;
82    currentsABC.arg3 = phaseC.phys;
83
84
85    // Read the motor's position from encoder driver
86    readPosition(&position.pos_raw);
87    rawToRadsMech(position.pos_raw, &position.th_mech);
88    rawToRadsEl(position.pos_raw, &position.th_el);
89  }
```

State Machine

```
#define AVG_SAMPLES 16
#define TWELVE_BIT_MAX 4095
#define SIX_TENTHS_SECOND 6000

#define STATE_INIT 0
#define STATE_READY 1
#define STATE_CALIB 2
#define STATE_RUN 3
#define STATE_FAULT 4

u16 calib_counter;
u8 calib_done;
s32 voltage_samples[AVG_SAMPLES];
s32 torque_samples[AVG_SAMPLES];
s32 phaseA_samples[AVG_SAMPLES];
s32 phaseB_samples[AVG_SAMPLES];
f32 err_q, err_d;

void fieldOrientedControl() {
  // Fault detection
  if (motor_errors.motor_overtemp|motor_errors.overvoltage|
      motor_errors.undervoltage|motor_errors.torque_disc|
      motor_errors.phaseA_overcurr|motor_errors.phaseB_overcurr)
  {
    statemachine.state = STATE_FAULT;
  }

    switch (statemachine.state) {
        case STATE_INIT:
            initVariables();
            statemachine.state = STATE_READY;
            break;

        case STATE_READY:
            if (statemachine.switches) {
                statemachine.state = STATE_CALIB;
            }
            break;

        case STATE_CALIB:
            // Get the sensor offsets
          if (calib_counter < AVG_SAMPLES) {
            voltage_samples[calib_counter] =
                battery_voltage.raw;
            phaseA_samples[calib_counter] =
                phaseA.raw - (TWELVE_BIT_MAX >> 1);
            phaseB_samples[calib_counter] =
                phaseB.raw - (TWELVE_BIT_MAX >> 1);
            calib_counter++;
          } else {
            s32 acc_v = 0, acc_A = 0, acc_B = 0;
            for (u16 i = 0; i < calib_counter; i++) {
              acc_v += voltage_samples[i];
              acc_A += phaseA_samples[i];
              acc_B += phaseB_samples[i];
            }
            battery_voltage.offset = acc_v/AVG_SAMPLES;
            phaseA.offset = acc_A/AVG_SAMPLES;
```

```
            phaseB.offset = acc_B/AVG_SAMPLES;

            calib_done = 1;
          }

          // Now precharge the DC-link capacitors for 600 ms.
          if (calib_done) {
            enablePrecharge();
            if (precharge_counter++ > SIX_TENTHS_SECOND) {
              enableMainRelay();
              disablePrecharge();
              precharge_counter = 0;
              statemachine.state = STATE_RUN;
            }
          }

          if (!statemachine.switches) {
            statemachine.state = STATE_INIT;
          }
          break;

        case STATE_RUN:
          // Transform the measured currents to the dq plane.
          clarkeTransform(&currentsABC, &currentsAlphBe);
          parkTransform(&currentsAlphBe, position.th_el,
                        &currentsDQ);

          // Limit PI controller and calculate output
          qController.limit = battery_voltage.phys;
          dController.limit = battery_voltage.phys;

          err_d = -currentsDQ.arg1;
          err_q = torque.phys - currentsDQ.arg2;
          newSample(&dController, err_d, &voltagesDQ.arg1);
          newSample(&qController, err_q, &voltagesDQ.arg2);

          // Transform the calculated dq voltages
          invParkTransform(&voltagesDQ, position.th_el,
                           &voltagesAlphBe);
          invClarkeTransform(&voltagesAlphBe, &voltagesABC);

          // Perform Space Vector Modulation
          spaceVectorModulation(&voltagesABC,
                                battery_voltage.phys,
                                &ocvvalues);
          setDutyCycles(&ocvvalues);


          if (!statemachine.switches) {
            statemachine.state = STATE_INIT;
          }
          break;

        case STATE_FAULT: // If any fault is detected, stop car.
          disableMainRelay();
          ocvvalues.ocv1 = 0;
          ocvvalues.ocv2 = 0;
          ocvvalues.ocv3 = 0;
          setDutyCycles(&ocvvalues);
          if (statemachine.clear_fault) {
```

```
            motor_errors.motor_overtemp = 0;
            motor_errors.overvoltage = 0;
            motor_errors.undervoltage = 0;
            motor_errors.torque_disc = 0;
            motor_errors.phaseA_overcurr = 0;
            motor_errors.phaseB_overcurr = 0;
            statemachine.state = STATE_INIT;
            statemachine.clear_fault = 0;
          }
          break;
    }
}
```

UART Communication

```
#define UART_MAIN 0
#define UART_SMACH 1
#define UART_ERROR 2
#define UART_PI 3
#define UART_SENS 4
#define UART_DUTY 5
#define UART_SET_D 6
#define UART_SET_Q 7

const char* states_uart[] = {"INIT\0", "READY\0", "CALIB\0",
                             "RUN\0", "FAULT\0"};
u8 uart_state = UART_MAIN;
u16 whole, comma;

u8 set_value[] = {0, 0, 0, 0, 0};
u8 read_bytes = 0;
u8 set_variable = 0;

void communicationTask() {

  u8 uart_cmd;
  uart_cmd = uartReceiveByte();
  if ((uart_cmd > 47) && (uart_cmd < 58)) {
    uart_cmd -= 48;
  }

  switch (uart_state) {
  case UART_MAIN:
    xil_printf("%c[2J", 27); // Clear screen
    xil_printf("This is the main menu for the live-view of the
                Zybo go-kart inverter.\r\n");
    xil_printf("Press 1 to access State Machine screen.\r\n");
    xil_printf("Press 2 to access Motor Error screen.\r\n");
    xil_printf("Press 3 to access PI Controller screen.\r\n");
    xil_printf("Press 4 to access Sensor Values screen.\r\n");
    xil_printf("Press 5 to access Duty Cycle screen.\r\n");

    if ((uart_cmd > 0) && (uart_cmd < 6)) {
      uart_state = uart_cmd;
    }
    break;

  case UART_ERROR:
    xil_printf("%c[2J", 27);
```

```
45      xil_printf("Motor Errors\r\n\r\n");
46      xil_printf("Motor overtemperature: %d\r\n",
47                  motor_errors.motor_overtemp);
48      xil_printf("Battery overvoltage: %d\r\n",
49                  motor_errors.overvoltage);
50      xil_printf("Battery undervoltage: %d\r\n",
51                  motor_errors.undervoltage);
52      xil_printf("Torque disconnected: %d\r\n",
53                  motor_errors.torque_disc);
54      xil_printf("Phase A overcurrent: %d\r\n",
55                  motor_errors.phaseA_overcurr);
56      xil_printf("Phase B overcurrent: %d\r\n\r\n",
57                  motor_errors.phaseB_overcurr);
58
59      xil_printf("Press 1 to clear all errors.\r\n");
60      xil_printf("Press ESC to return to main menu.\r\n");
61      if (uart_cmd == 27) {
62        uart_state = UART_MAIN;
63      } else if (uart_cmd == 1) {
64        statemachine.clear_fault = 1;
65      }
66      break;
67
68    case UART_SET_D:
69      if ((!read_bytes) && (!set_variable)) {
70        xil_printf("%c[2J", 27); // Clear screen
71        xil_printf("Set d-direction PI Controller Values\r\n\r");
72        xil_printf("1: a0\r\n");
73        xil_printf("2: a1\r\n");
74        xil_printf("3: b1\r\n");
75        xil_printf("4: Limit\r\n");
76
77        xil_printf("Press ESC to return to previous screen.\r\n");
78
79        if (uart_cmd == 27) {
80          uart_state = UART_PI;
81        } else if ((uart_cmd > 0) && (uart_cmd < 5)) {
82          set_variable = uart_cmd;
83        }
84      } else {
85        if (!(uart_cmd == 27)) {
86          set_value[read_bytes] = uart_cmd;
87          read_bytes++;
88          xil_printf("%d", uart_cmd);
89          if (read_bytes >= 5) {
90            switch (set_variable) {
91            case 1:
92              dController.a_0 = set_value[0]*10.0 + set_value[1] +
93                                set_value[2]/10.0 +
94                                set_value[3]/100.0 +
95                                set_value[4]/1000.0;
96              break;
97            case 2:
98              dController.a_1 = set_value[0]*10.0 + set_value[1] +
99                                set_value[2]/10.0 +
100                               set_value[3]/100.0 +
101                               set_value[4]/1000.0;
102             break;
103           case 3:
104             dController.b_1 = set_value[0]*10.0 + set_value[1] +
```

```
                                set_value[2]/10.0 +
                                set_value[3]/100.0 +
                                set_value[4]/1000.0;
            break;
          case 4:
            dController.limit = set_value[0]*10.0 + set_value[1]
                                + set_value[2]/10.0 +
                                set_value[3]/100.0 +
                                set_value[4]/1000.0;
          }
          set_variable = 0;
          read_bytes = 0;
          uart_state = UART_PI;
        }
      } else {
        set_variable = 0;
        read_bytes = 0;
        uart_state = UART_PI;
      }
    }
    break;

  // Further states left out, as they operate by same principle.
  }
}
```

Main

```
// XADC interrupt handler
static void XAdcInterruptHandler(XAdcPs *XAdc);
static void XAdcInterruptHandler(XAdcPs *XAdc) {
  sensorProcessing();
  fieldOrientedControl();
}

int main()
{
    init_platform();

    initGpios();
    initUart();
    initXAdc();
    initGicXAdc((Xil_ExceptionHandler) XAdcInterruptHandler);
    initVariables();

    while(1) {
        communicationTask();
        usleep(250000);
    }

    cleanup_platform();
    return 0;
}
```