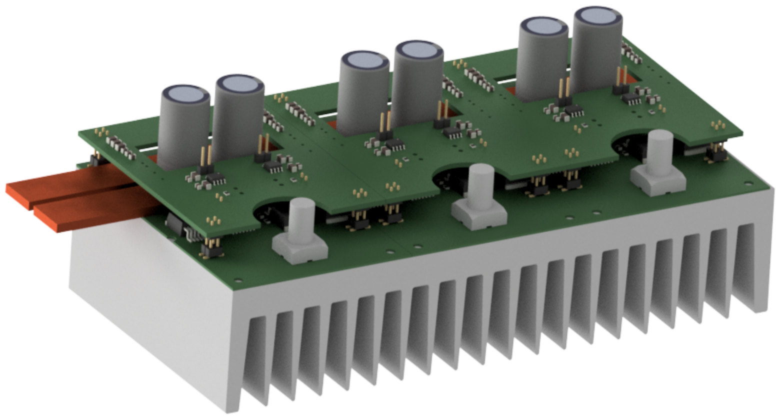
Control of PMAC motor for an electrical kart

University of Southern Denmark

Students:
Casper Busch Melchiorsen, 010395
David Kis-Toth, 060193
Jacob Offersen, 311294
Jonas Holmgren, 260785
Peter Nielsen, 130394

Supervisors:
Karsten Holm Andersen: *kha@mci.sdu.dk*
Kurt Bloch Jessen: *kbj@mci.sdu.dk*

Period:
February 2019 - June 2019 (F19)



Signatures

Approved:	[illegible]	Approved:	[illegible]
28-05-2019	Casper Busch Melchiorsen	28-05-2019	David Kis-Toth
Approved:	[illegible]	Approved:	[illegible]
28-05-2019	Jacob Offersen	28-05-2019	Jonas Holmgren
Approved:	[illegible]		
28-05-2019	Peter Juel Nielsen		

1 Abstract

The project goes through the steps needed to design a three-phase inverter with a build-in motor controller for an electric go kart. The inverter is designed to run the *ME1117 Brushless PMAC motor* at 4.5kW continuous and 14kW peak. The inverter hardware consists of four different kinds of PCB's. A main interface board that was provided as material for the project, an analog interface board that handles the analog signals, a power circuit on an aluminum PCB for each phase handling the four SMD MOSFET's for each leg and lastly the driver boards converting control signals from the embedded system.
The embedded system consists of a Xilinx Zybo Zynq-7000 ARM/FPGA Training Board. The controller measures rotor angle, phase currents and torque pedal position and controls the motor with Field Orientated Control to output the requested torque. At low speeds a more accurate rotor angle than the one out of the encoder is approximated with linear interpolation to make the motor run smoother.

2 Preface

The project is the semester project on the 1st semester on the Masters in Electronics at The University of Southern Denmark and was done in the period from January 2019 to June 2019.
The group would like to thank our supervisors Karsten Holm Andersen and Kurt Bloch Jessen for the help and support throughout the project.

3 Reading guide

The report is expected to be read in chronological order.
It contains extensive use of symbols and abbreviations. A list of both including short descriptions can be found in section 10 and 11 respectively.
The report is written expecting the reader to have knowledge about power electronics, control theory and embedded systems.

The report has the following sections:

Section 4 - Introduction gives a short introduction to the project and outlines a set of system requirements.

Section 5 - Hardware goes through the component selection, circuit and PCB design of the analog board and its handling of the analog signals, the power board's handling of the power components, the driver board's handling the control of the power board depending on signals from the embedded system.

Section 6 - Control contains the modeling of the system, as well as the design of a suitable controller.

Section 7 - Embedded system contains descriptions of all elements implemented in the embedded system. These include the logic on the FPGA as well as functionality implemented on the processing system.

Section 8 - Discussion goes through the interpretation of the test results, some possible problems with the design and explains how the design relates to the requirements found in section 4.1.

Section 9 - Conclusion outlines how the designed system fulfills the requirements.

Section 10 - Symbol list contains a list of all symbols together with a short description.

Section 11 - Abbreviation list contains a list of all abbreviations together with a short description.

Section 12 - References contains a list of the sources used.

Section 13 - Appendix
Appendix A - Code samples contains a selection of the most important and interesting pieces of code.
Appendix B - Component List contains a list of components for each board.
Appendix C - Analog board schematic and layout contains the analog board schematic and the PCB layout.
Appendix D - Driver board schematic and layout contains the driver board schematic and the PCB layout.
Appendix E - Power board schematic and layout contains the power board schematic and the PCB layout.

Contents

4 Introduction

One of the most important and useful motor types in modern electronics is the BLDC (Brushless DC Motor) / PMAC (Permanent Magnet Motor). The motor achieves this status by being highly versatile, robust, have low noise with a high torque. The project focuses on the controlling of a *ME1117 Brushless PMAC* motor for an electrical go kart.

The project designs the hardware shown in the green square on figure 1. The figure show a simplified motor controller consisting of a three-phase inverter controlled by a Zynq-7000 controller.

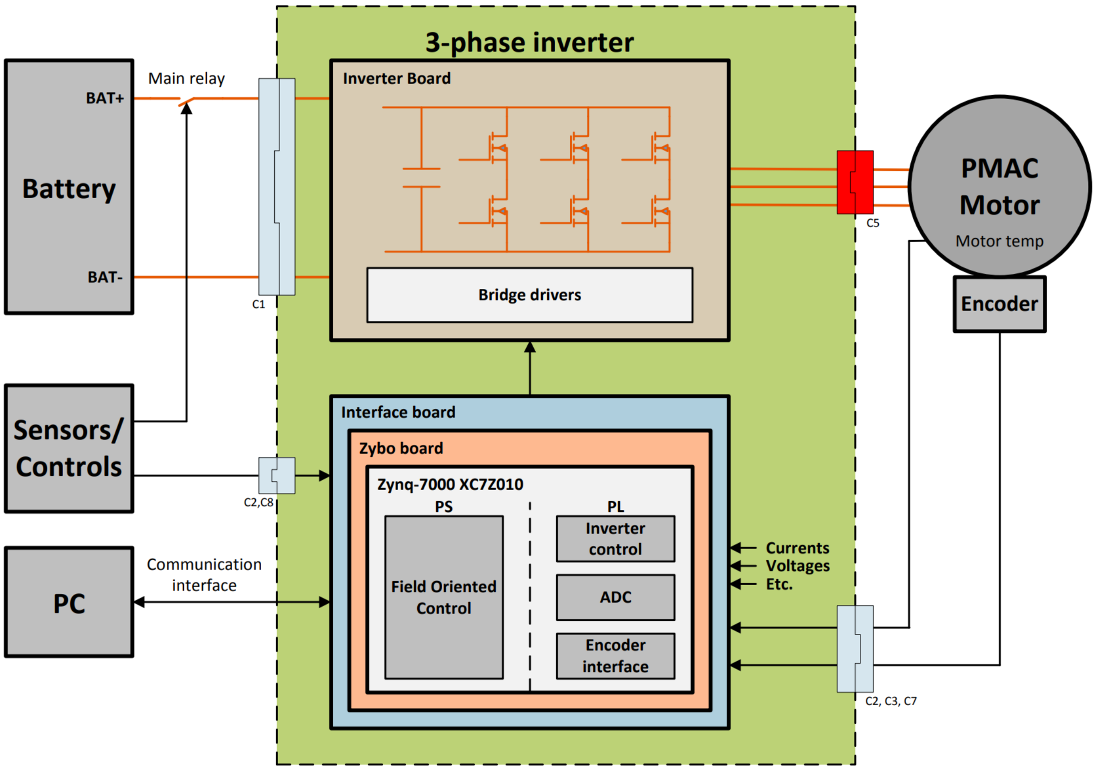


Figure 1: A suggestion to a possible solution given in the project description. [1]

The illustration shows how the main connections between elements should look. On the right side is the motor itself with an encoder attached to measure the rotor angle. The motor is driven by the three-phase inverter developed in this project as can be seen in the middle. Below the inverter in the green square is the Zynq controller on a Zybo training board which controls the inverter. By receiving a range of sensor signals the controller handles the control needed for the inverter to run the motor at the requested torque.

4.1 Requirements

Project requirement
The list below contains the requirements set for the project. Some requirements are part of the project description [1] while others are set by the group.

- The motor controller should only be able to control the PMAC motor in forward direction.
- The solution must use a Zynq FPGA and a self-designed 3-phase inverter to control the PMAC motor.
- There should be a control circuit, with a model of the motor and its controllers.
- It should be possible to set and get relevant parameters through a communication interface on a computer.
- It is not allowed to change any mechanical or electrical parts on the kart, except replacing the Sevcon gen4 controller.
- The cost of components may not exceed 3000 dkr.

5 Hardware

The section goes through the design of the hardware for the motor driver. It includes a walk-through of the provided material for the project which includes an encoder interface. Then the analog board is designed which handles all analog signals. The power board handling high power components and driver board handling driver signals are designed and then a combined section about how they fit together.

5.1 Motor specifications

The motor used in the go kart is a *Motenergy ME1117 PMAC Motor*. Its parameters can be seen in table 1.

Pole pair	$4\ pairs\ (8\ poles)$
Maximum rotational speed	$5000RPM$
Voltage rating	$0-76V$
Continuous current	$100\ A$
Peak Current for 1 min	$300\ A$
Torque constant	$0.13Nm/A$
Phase to Phase Resistance	0.013Ω
Phase to Phase Inductance	0.1 mH
Armature Inertia	$52\ kg/cm^2$

Table 1: Motor parameters specifying the motor *Motenergy ME1117 PMAC* [2].

5.2 Main interface board

Part of the material provided to start the project is the main interface board. The board contains a range of smaller circuits and acts as the interface between the hardware and the embedded controller.
The circuits include a pre-charge circuit, high motor temperature detection circuit, battery voltage measurement circuit, relay control, encoder interface, interface to buttons and switches and some voltage references and supplies.
The section contains a short introduction to the encoder interface, all other modules will be references when they are used.

5.2.1 Encoder

On the motor is an *AM256* magnetic 8 bit encoder mounted. The encoder is supplied from the main interface board and the signals are routed from the encoder to the Zybo.
The decoding and translation of the encoder signal are done in the Zybo which is discussed in section 7.2.2.

5.3 Analog interface board

A smaller part of the hardware design is an analog board with two main functions. Convert the analog signal from the current transducers and the torque sensor into signals between $0V$ and $1V$ for the build in ADC's on the controller.

5.3.1 Current transducer

The current transducer used is the *LF 205-S*. The transducer has the conversion ratio 1 : 2000, which means when the maximum current, $\pm 300A$, passes through, it will give a $\pm 50mA$ signal. To use the ADC's full resolution the signal is converted into a signal going from $0 \rightarrow 1$. First the signal is converted to going from $\pm 0.5V$ with a shunt resistor. The size of the shunt is calculated with equation 1.

$$R = \frac{0.5V}{50mA} = 3.33\Omega \tag{1}$$

The resistance is made by setting three 10Ω in parallel.
The signal is then offset by $+0.5V$, resulting in a signal varying between 0 and $1V$. The offsetting is done using a *AD620A* instrumentation amplifier with an offset input. The two R_g ports are left open, which will give unity gain on the output.

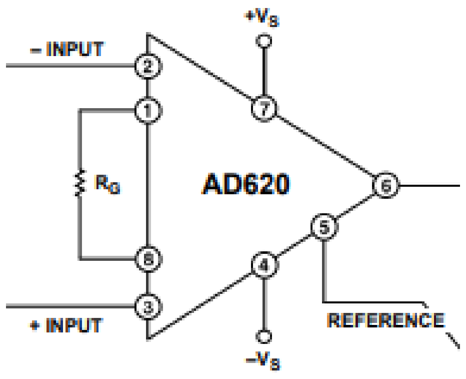


Figure 2: Instrumentation amplifier AD620A

The input reference to the *AD620A* has to be very exact. Small differences can cause great inaccuracies in the current measurment. Therefore a voltage reference is used to keep a stable voltage at $0.5V$. The used voltage reference is an *LM385Z-1.2*. In conjunction with a voltage divider of resistors $R1 = 1k\Omega$, $R2 = 1k\Omega$, $R3 = 470\Omega$, it yields a constant voltage of

$$V = V_{ref} \cdot \frac{R1}{R1 + R2 + R3} = 0.5V \tag{2}$$

where

$$V_{ref} = 1.235V \tag{3}$$

A buffer is added to give a low impedance signal to the *AD620A*.
At the input of the instrumentation amplifier a resistor is placed at both the inverting and the non-inverting input to reduce current. Using the *AD620A* is also

advantageous for its high CMRR (Common-mode Rejection ratio) that helps with cancelling out noise present on both input pins with the same waveform. It can also use the $\pm 15V$ dual supply present on the board.

5.3.2 Torque pedal measurement

The output of the torque pedal connects to the same ADC as one of the current channels, hence it also has to be in the range of $0 - 1V$. The pedal is basically a $7.5k\Omega$ potmeter, so the idea is to parallel it with the lower resistor of a voltage divider to get about $1V$ at full range. Using $V_{ref_3} = 3.3V$ as excitation for the divider and $R1 = R2 = 10K\Omega$, the following equation is true at full throttle:

$$V = V_{ref_3} \cdot \frac{R_{pot} \cdot R15}{R_{pot} \cdot R15 + R14} = 0.99V \tag{4}$$

When the pedal is not pressed at all, the 0Ω of the potmeter shunts the supply to the ground.

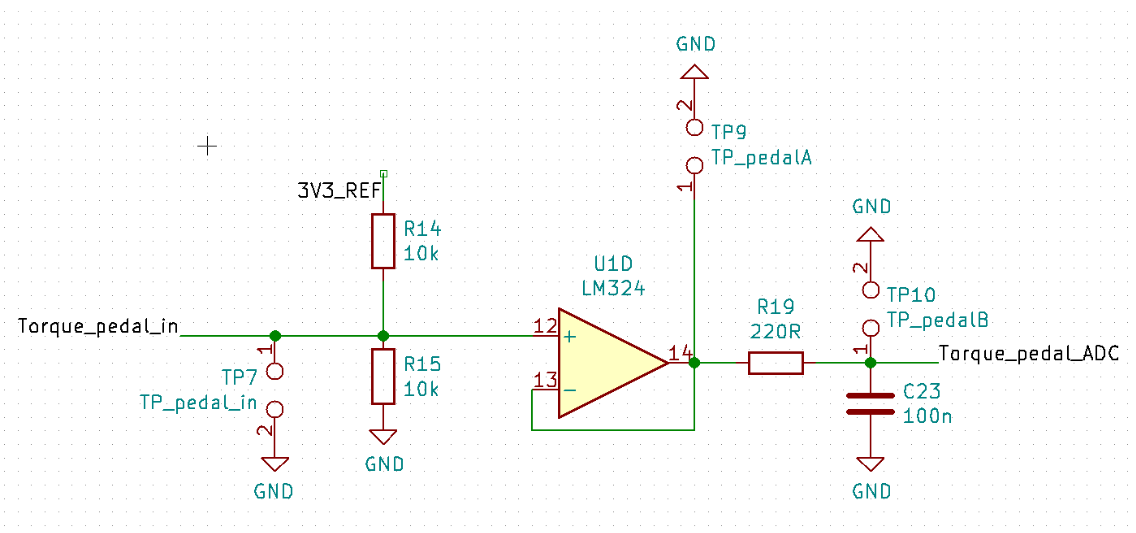


Figure 3: Torque pedal reading circuit

For this channel, an opamp in voltage follower configuration is sufficient instead of an instrument amplifier, since no offsetting is required.

5.4 Power board

The inverter is divided into two PCB's, one for the high power part, the Power board and one for the driver part, the Driver board. The following section describes how the Power board for the inverters are designed and what considerations have been made. It will describe what kind of PCB and which components have been used and why. The layout of the PCBs will also be described and what considerations has been made.

The objective is to design a two level three-phase inverter to drive the Permanent Magnet Synchronous Motor (PMSM). The simple model of the inverter consists of six switches and some capacitance on the DC-link. On figure 4 a simple model of the Power board is shown.

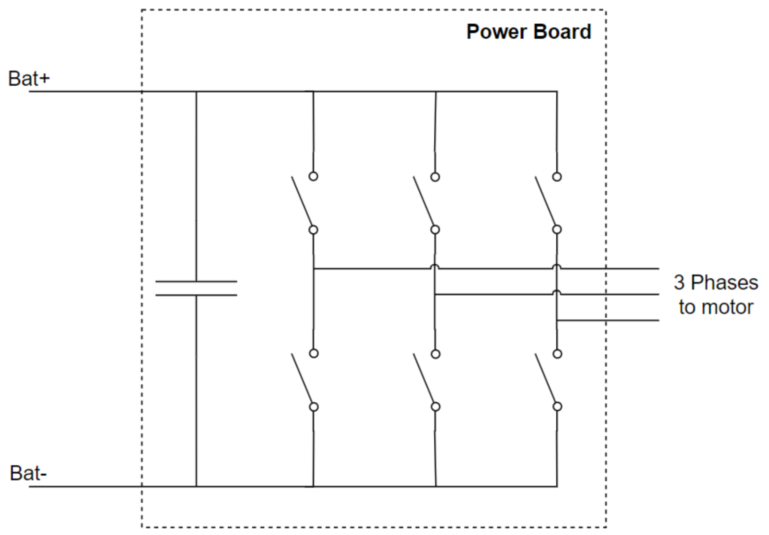


Figure 4: Sketch of the Power Board

It means that a switch component needs to be selected. The switch component has great impact on the inverter design, due to the fact that it is the main source of losses in the inverter. It is therefore important to choose a good transistor for switching. Besides the transistor choice, a good PCB layout and design is needed, to avoid electrical problems and to have good heat transmission from the transistors.

5.4.1 Switching frequency

When deciding a switching frequency for the inverter, the rotation frequency of the motor must be known to ensure a high enough frequency. It is known from the motor parameters in table 1, that the maximum rotation speed is $5000rpm$ and it has 8 poles that equals to 4 pole pairs. From this the maximum frequency of the electrical field be determined.

$$f = \frac{v_{rpm} \cdot P}{60} = \frac{5000 \cdot 4}{60} = 333.33Hz \tag{5}$$

Where v_{rpm} is the rotational speed in rpm and P is the number of pole pairs in the motor. From equation 5 the maximum frequency of the electrical field is $333.3Hz$. It is desired to have between 10 and 20 switch periods per period of the electrical field. 20 switch periods per electrical field period results in a switching frequency of $6.67kHz$. It is not necessary to switch faster than the $6.67kHz$, but it's decided to use a switching frequency of $10kHz$. Choosing $10kHz$ over $6.67kHz$ gives a bit higher switching loss but also results in the need for smaller capacitors.

5.4.2 PCB selection

Two types of PCB's has been considered, an ordinary glass-fiber multi-layer PCB and a single-layer aluminum PCB. Both types have their pros and cons. With the ordinary PCB there is better opportunities for conducting the heat away from through-hole components because the components can be mounted directly to a heat sink. Due to low heat transfer SMD components are not possible with ordinary

PCB's. SMD components can result in a better and more compact layout because of their smaller footprint and are available with multiple source legs, which minimize the common source for the switch components. The use of SMD components will be possible with the aluminum PCB, due to the high heat transfer through the aluminum PCB. With the aluminum PCB it is not possible to use through hole components, which can give some problems when it comes to the mounting of the large capacitors for the DC-link.
The ordinary PCB has the benefit of having multiple layers which makes it easier to route the layout of the board. Due to the price constraints for the project going for more than two layers would not be feasible. It is possible to get the aluminum PCB with two layers, but it results in a too high thermal resistance and price.
Based on these considerations, an aluminum PCB has been chosen, because of the benefits gained by using SMD components.

5.4.3 Transistor selection

Transistor selection is an important aspect in the design of the inverter because the transistors have great affect on the power loss in the system and thus the heat dissipation which set limitations on design. A couple of different transistors characteristics have been the major factors used in the transistor selection. The transistor has to be rated for the minimum drain-source current of $300A$ if no parallel devices are used. It is chosen that the rated drain-source voltage should be around $100V$ due to the battery in the go kart has a maximum voltage of $57.6V$, plus some safety margin for voltage spikes. The maximum battery voltage is found based on the cell voltage of a $LiFePO_4$: $3.6V$, multiplied with the number of cells, which is 16. The transistor has to have a fall time of less than $250ns$ and rise time of less than $50ns$. The transistor also has to have the lowest possible on-resistance, to minimize conduction losses, which are very significant when conducting a current of $300A$.
The MOSFET chosen is an *Infineon "IPB017N10N5"*. Some of the important specifications are listed in the table below.

Parameters	**Value**	**Unit**
V_{DS}	100	V
R_{DS-ON}	1.7	$m\Omega$
I_D	273	A
Q_G $(0-10V)$	168	nC
Q_{GD}	34	nC
Q_{GS}	53	nC
Q_{sw}	51	nC
Rise time	23	ns
Fall time	27	ns

Table 2: Transistor parameters taken from the *IPB017N10N5* datasheet

The transistor with these specifications and the lowest possible drain-source on-resistance, for an affordable price, was a MOSFET. The MOSFET has the benefit of having a low current needed for turning on, the ability to handle high current needed for the motor and fast enough for the switching frequency. Since price is of serious concern when it comes to choice of components, then some sort of compromise will be made.

When looking at the parameters, a couple of things need to be considered. From the top it can be seen that the drain-source voltage is $100V$, so it is able to handle the battery voltage and have a reasonable safety margin. Next the $[I_D]$ is not $300A$ or above, which leads to a design choice of having two or more in parallel. Having more transistors in parallel has the benefit of splitting the current up between them, lowering the conduction power loss. Having lower power loss decreases the amount of heat generated, which leads to lower risk of overheating. The gate charge is the amount of charge that needs to be delivered to the transistor gate from the driver circuit, in order to turn the transistor on. The lower gate charge the better but normally with low drain-source resistance comes higher gate charge. For an inverter it is more important to have low drain-source on-resistance then than low gate charge because speed of the turning on and off of the MOSFET is reduced anyway. Looking at the increase in driver power losses, the loss will not be as significant as the increase in power losses in a MOSFET with higher drain-source on-resistance.

5.4.4 Power and heat dissipation in the transistors

There are two types of power losses in the transistors. One is conduction losses, which is caused by the internal resistance in the MOSFET between the drain and the source, when it is conducting. The other one is switching losses, which appears when the MOSFET is turning on and off. Based on the power loss calculations the expected heat increase of the MOSFET's can be estimated, which will be used to decide, how many MOSFET's in parallel are needed.

5.4.4.1 Conduction power loss

When a MOSFET is conducting there will be a power dissipation due to the internal drain-source on-resistance in the transistor. Thus the conduction power loss in a transistor is calculated in the same way as the power loss in a resistor. Equation 6 describes the conduction power loss for one MOSFET, $P_{con/MOSFET}$.

$$P_{con/MOSFET} = \left(\frac{I}{N \cdot 2}\right)^2 \cdot R_{DS-ON} \tag{6}$$

I is the peak current, N is the number of transistors in parallel per switch, and R_{DS-ON} is the drain-source on-resistance of the MOSFET. The peak current is divided by the number of transistors in parallel, to get the current in one MOSFET. The current is sinusoidal and the average duty cycle of the PWM over a period is 50%, it is therefore divided by two to get the current that one transistor is conducting.
To calculate the conduction power loss for all the transistors, it is then multiplied with the number of parallel MOSFET's per switch, two for taking both the high and low side into account, and the number of legs in the inverter, which is three.

$$P_{con} = \left(\frac{I}{N \cdot \sqrt{2}}\right)^2 \cdot R_{DS-ON} \cdot N \cdot 2 \cdot 3 \tag{7}$$

On figure 5 the relationship between conduction power loss per MOSFET, equation 6, and for all the MOSFET's, equation 7, is plotted as a function of the number of MOSFET's in parallel.

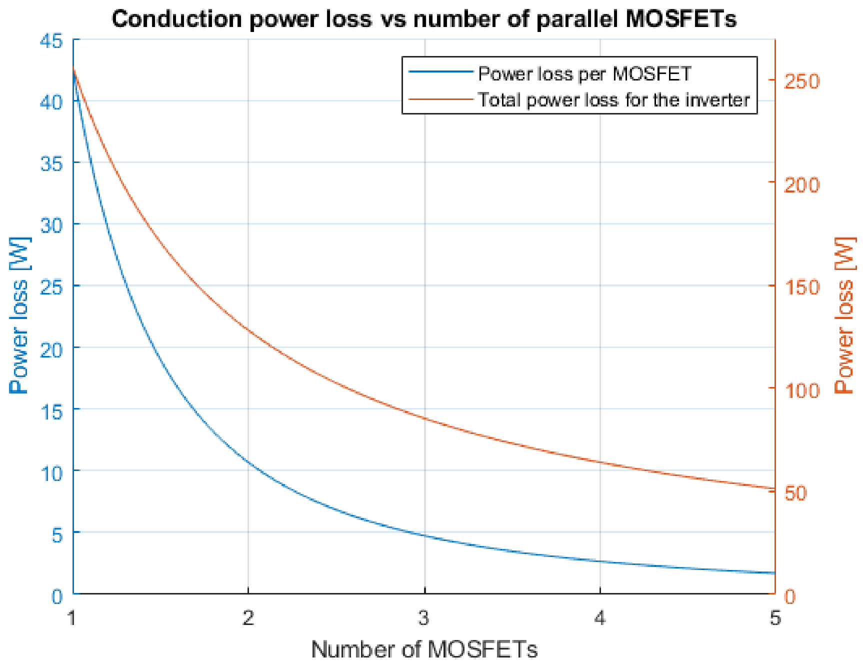


Figure 5: Conduction power loss versus the number of MOSFET's in parallel per switch. The blue curve describes the power loss per MOSFET and the red describes the total power loss

For figure 5 the peak current, I, is set to $300A$, and the drain-source on-resistance, R_{DSon}, is set to $1.9m\Omega$, which is value at $80°C$. [5]
On figure 5 it can be seen that when the number of MOSFET's in parallel per switch increase the conduction power losses reduces exponentially, both for each individual MOSFET and for the whole inverter.

5.4.4.2 Switching power loss

The other cause of power loss in a MOSFET is the switching power loss. The switching power losses is caused by the period, when switching, where both the voltage over and the current through the MOSFET is not zero which is called hard switching. The counterpart to hard switching is soft switching which is when the MOSFET switches on or off without the voltage and current overlapping. Soft switching does not result in any power loss. The switcing types are illustrated on figure 6.

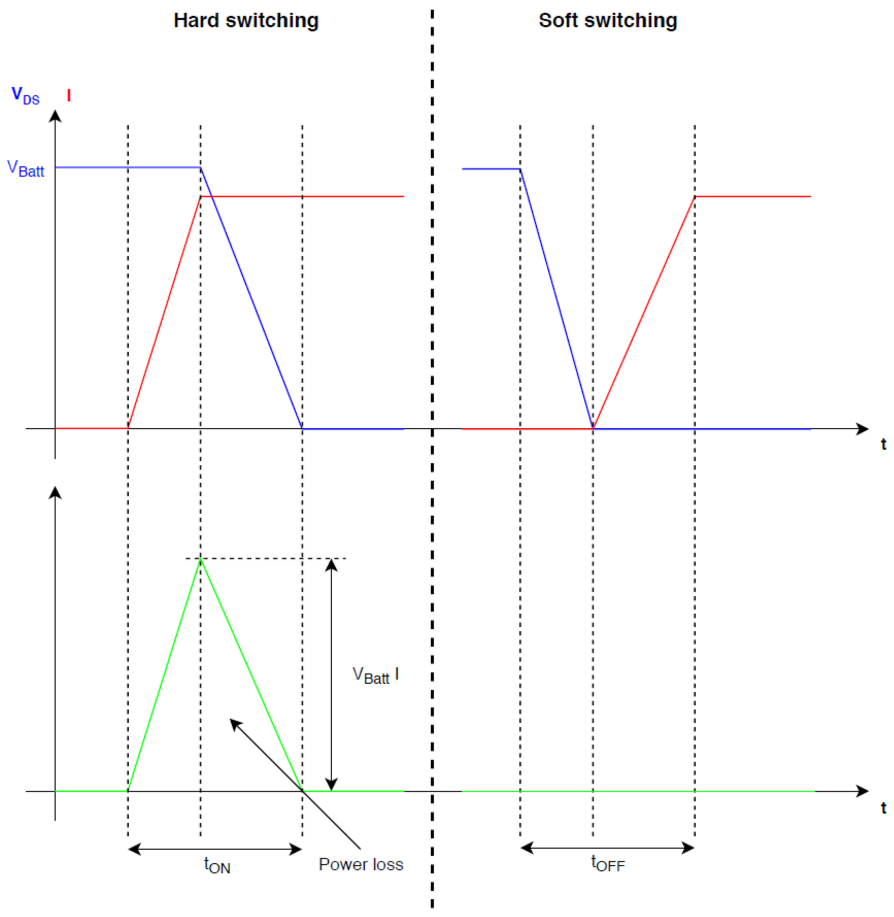


Figure 6: Illustration of hard and soft switching of MOSFET

When running the motor, the motor current has a positive and a negative half cycle. During the positive half cycle the current going into the motor, and in the negative half cycle the current going from the motor into the inverter. In the positive half cycle the low-side switch gets soft switched and the high-side switch gets hard switched. Opposite in the negative half cycle the low-side switch is hard switched and the high-side switch is soft switched. This results in each transistor hard switch 50% of the time.
The switching power is calculated as the area under the green line on figure 6 multiplied by the switching frequency. This is multiplied by a half because the switches are only hard switched 50% of the time, and therefore the power loss is the half.

$$P_{sw/MOSFET} = 0.5 \cdot 0.5 \cdot V_{batt} \cdot \frac{I}{N} \cdot (t_{on} + t_{off}) \cdot f_s \tag{8}$$

Where V_{batt} is the battery voltage, t_{on} is the turn on time, t_{off} is the turn off time and f_s is the switching frequency of the transistors. Equation 8 is the switching power loss per MOSFET. To get the switching power loss for all of the MOSFET's in the inverter, this has to be multiplied with the number of parallel MOSFET's, which makes 2 when taking both the high- and low-side into account, together with the number of phases.

$$P_{sw} = 0.5 \cdot 0.5 \cdot V_{batt} \cdot \frac{I}{N} \cdot (t_{on} + t_{off}) \cdot f_{sw} \cdot N \cdot 2 \cdot 3 \tag{9}$$

From this equation, a curve of the switching power loss in one MOSFET and in all the MOSFET's versus the number of the parallel MOSFET's, is made.

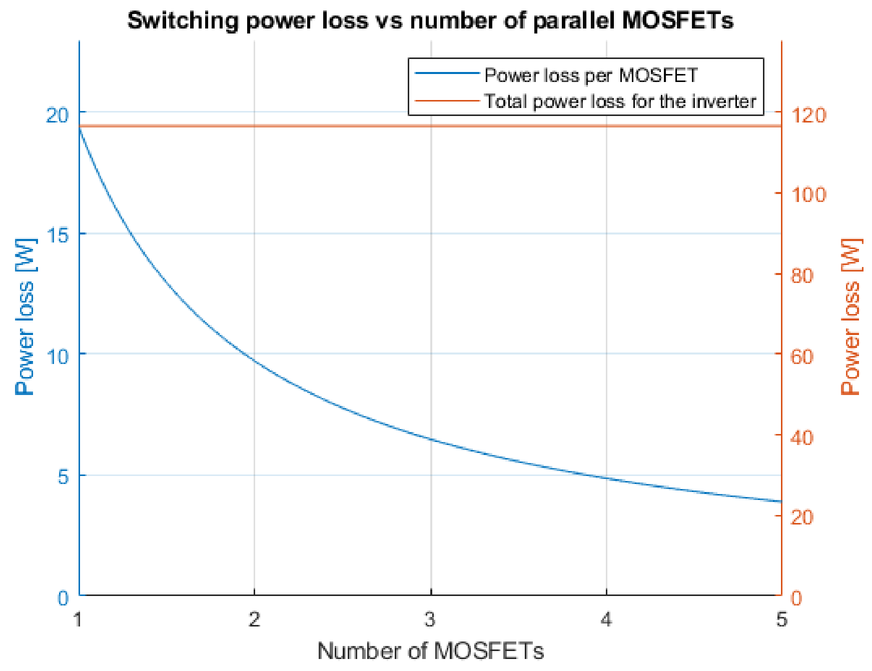


Figure 7: Switching power loss versus the number of parallel MOSFET's in parallel per switch. The blue curve describes power loss per MOSFET and the red describes the total power loss.

Where I is the peak current that is set to $300A$. The battery voltage, V_{batt}, is set to $57.6V$ and the switching frequency is set to the chosen frequency: $10kHz$. The turn on time, t_{on}, and the turn off time, t_{off}, are calculated with the following equations.

$$t_{on} = \frac{Q_{sw}}{I_{GSource}} \tag{10}$$

$$t_{off} = \frac{Q_{sw}}{I_{GSink}} \tag{11}$$

Where Q_{sw} is the switching charge of the MOSFET, which is stated in the data sheet.[5] $I_{GSource}$ and I_{GSink} are respectively the current the driver deliverers to the gate of the MOSFET when turning it on and off. These are calculated in section 5.5.2. The turn on and off end up being $75ns$ and $375ns$, which could be faster, but is chosen to be limited to this. Due to reducing high-frequency ringing when switching the MOSFET's. The result in a higher switching power loss, but because the switching loss does not have that big of an impact on the total power loss, due to the low switching frequency, it is not considered a problem.
On figure 7 it can be seen that the switching power loss of each MOSFET is reduced with number of parallel MOSFET's, but the over all power loss of the inverter is the same independent of the number of parallel MOSFET's.

5.4.4.3 Heat dissipation

When the conduction power loss and switching power loss is determined compared to the number of transistors, the total power loss is known. From the total power loss in the transistors, the heat dissipation can be determined by the number used. When determining the heat dissipation of the inverter, all the thermal resistances between the junction of the MOSFET and the ambient has to be specified. First of all there is the internal junction to case resistance for the MOSFET. When using the aluminum PCB with the SMD components, the heat has to be transferred through the PCB. Which adds an extra thermal resistance to the system. The aluminum PCB is divided into three parts, the copper layer, a dielectric layer for isolating the copper from the aluminum, and the aluminum layer. Then there is a thin layer of thermal paste to connect the PCB to the heat sink. Lastly resistance before the ambient is the heat sink. It is all illustrated on figure 8.

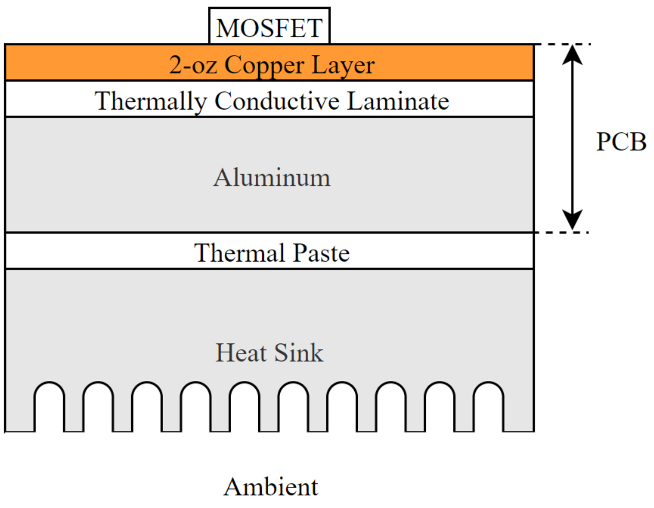


Figure 8: Overview of the thermal resistances between the junction of the MOSFET and ambient

The thermal resistance from the junction to the case of the MOSFET is stated to be $0.4K/W$ in the data sheet for the MOSFET.[5] The thermal resistance of the copper layer, is calculated from equation 12

$$R_{thermal} = \frac{t_{layer}}{A_{MOSFET} \cdot k_{material}} \tag{12}$$

Where t_{layer} is the thickness of the layer in m, A_{MOSFET} is the surface area of the MOSFET in m^2, and $k_{material}$ is the thermal conduction constant for the layer material in $\frac{W}{m \cdot K}$.
The thickness of the copper layer is known to be $1oz/foot^2$ from *pcbway.com*, which is converted to $34 \cdot 10^{-6}m$ from the density of copper. The surface area of the MOSFET is found in the data sheet to be $63 \cdot 10^{-6}m^2$.[5] The area of the MOSFET is used all the way down to the heat sink, because it is assumed that the heat will go straight through the PCB and the paste. Which is most likely not whole truth but be the worst case. The thermal conduction constant of copper is $401\frac{W}{m \cdot K}$.[11]
The thermal resistance of the dielectric layer is calculated based on the same equation as 12.

Where the thickness of the dielectric layer is $100\mu m$, which is stated by the manufacture, *pcbway.com*. The thermal conduction constant is chosen to be $1\frac{W}{m \cdot K}$. The manufacture also offers the aluminum PCB with a dielectric layer with a thermal conduction constant of $2\frac{W}{m \cdot K}$, but due to price this was not chosen.
Equation 12 is used again for calculating the thermal resistance of the aluminum layer of the PCB. Where the thickness of the aluminum layer is the chosen PCB thickness of $0.8mm$ minus the thickness of the two other layers. The thickness of $0.8mm$ is chosen because that is the thinnest option without getting the PCB to increase in price. The thermal conduction constant for aluminum is $236\frac{W}{m \cdot K}$.[11]
The thermal resistance for the total PCB is then:

$$R_{PCB} = R_{copper} + R_{del} + R_{alu} \tag{13}$$

Where R_{copper} is the thermal resistance of the copper layer, R_{del} is the thermal resistance of the dielectric layer, and R_{alu} is the thermal resistance of the aluminum layer.
To calculate the thermal resistance of the thermal paste, equation 12 is used as well. The thickness of the paste is estimated to be $0.1mm$ and the thermal conduction constant of the paste is set to $3\frac{W}{m \cdot K}$. Based on the values for different thermal paste types, and a value close to the average of them was chosen.
The results for the total temperature difference from the thermal paste to the MOSFET junction can be calculated with equation 14.

$$T_{JP} = P_{loss/MOSFET} \cdot (R_{JC} + R_{PCB}) \tag{14}$$

Where R_{JC} is the internal junction to case thermal resistance and P_{loss} is the total power loss for one MOSFET.
To calculate the temperature difference over the heat sink, the average power loss is used instead of the maximum power loss. This is done because the heat sink has a higher thermal capacitance, and therefore is less reactive to changes in temperature. The average power loss is calculated in the same way as the maximum power loss, but instead of using the maximum current an average current is estimated. The average current is estimated to $75A$. The temperature difference over the heat sink can then be calculated with equation 15.

$$T_{HA} = P_{Avg} \cdot R_{HA} \tag{15}$$

The P_{Avg} is the average power loss for all the MOSFET's and R_{HA} is the thermal resistance of the heat sink.
The final temperature difference from ambient to the junction of one MOSFET can be calculated with equation 16.

$$T_J = T_A + T_{HA} + T_{JP} \tag{16}$$

Where the T_A is the ambient temperature. It should be noted that the temperature difference over the heat sink depends on the total power loss. The temperature difference from the paste to the junction of a MOSFET depends on the power loss for one MOSFET, due to the thermal resistance of the heat sink that is common for all MOSFET's. There is also a parallel thermal resistance for each MOSFET from the paste to the junction.

On figure 9 is the relationship between the junction temperature of the MOSFET's and the number of MOSFET's in parallel showed.

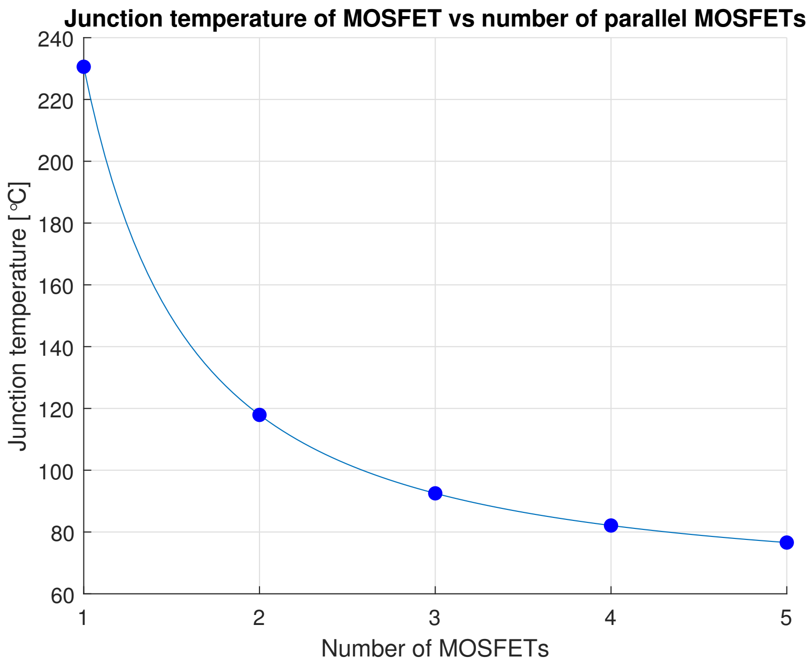


Figure 9: Junction temperature versus the number of transistors in parallel per switch

Based on figure 9 it is decided to use two MOSFET's in parallel per switch. Because the Power board only got one layer, it is preferred to have as few MOSFET's as possible, to avoid making the layout of the Power board too complicated. From figure 9 it can be seen that the junction temperature of the MOSFET's are going to be $118°C$ which is a bit high, but considering it being the worst case, this is determined acceptable.

5.5 Driver board

The section describes the theoretical background, the design considerations taken and the implementation of the driver circuit and the PCB.

5.5.1 Analysis

The gate driver selected for this design is the *SI8261BAC-C-IS*[7] from Silicon Labs. It is driven by a $3.3V$ output of the Zybo while providing an output of $12V$, constrained by the attached $12V$ floating supply *Cosel MGS1R54812*[8]. A floating supply is required for the high-side, in order to take on a potential independent of the power ground, when its drain and source potential is constrained by the load and the battery voltage. The gate driver's ground floats along with the source potential of the MOSFET, so it's always able to provide a voltage of $12V$ between the gate and the source. The advantage of having an isolated supply for the low-side as well, is better noise tolerance - the MOSFET's reference level is made somewhat independent of the ripple current on the power ground from all the loads. The high-side floating supply also has the advantage over a bootstrap capacitor, of not requiring part of an output cycle to be used for charging the capacitor. This way more power can be used to propel the go kart.

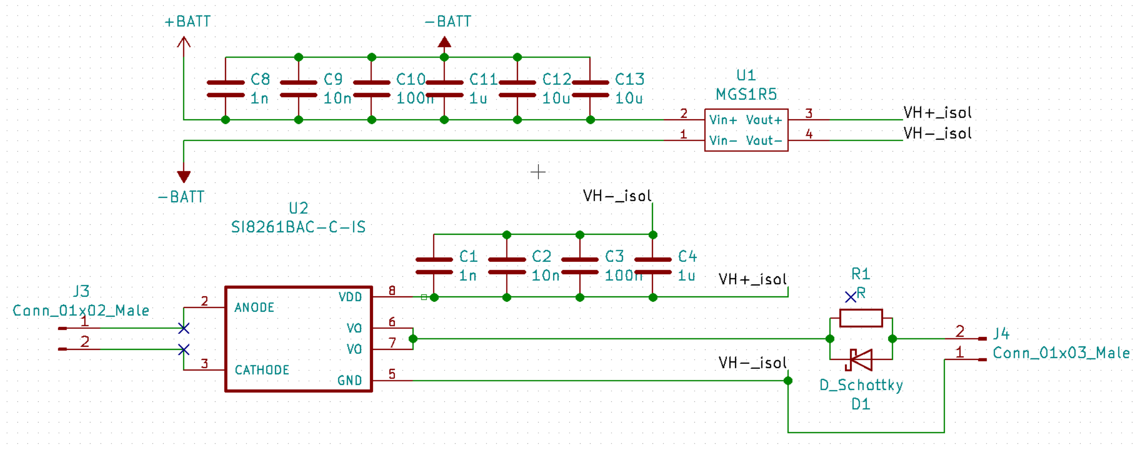


Figure 10: One leg of one phase of the driver circuit

Figure 10 shows the driver circuit of one leg of one phase. Separate drivers were chosen for the separate legs of each phase to reduce the thermal stress of the individual ICs. The low-side and the high-side driver circuits are identical.

A series of capacitors are attached both to the input and output of the floating supply. The smaller ones are meant to serve as high-frequency decouplers while the $10\mu F$ ones are the bulk capacitors to prevent droop on the $12V_{DC}$ rail. It helps with countering the noise-emitting effects of the long wires and quick transients from the battery when the load changes rapidly.

The turn-on and turn-off times were chosen to be around the slower end of the available range to make the circuit less prone to overshoot and ringing caused by the parasitic inductances and capacitances inherently present in the layout. The analysis of deciding the switching frequency is detailed in section 5.4.1.

In Figure 11 and Figure 12 taken from *TIDA-00364*[4] reference design by Texas Instruments, the use of split gate resistors can be seen. These consist of R2, which is common to all the parallel MOSFETs, and R20 through 24, which are individual to each MOSFET. R2 helps to easily change the turn on gate current of all the parallel MOSFETs by changing a single resistor instead of changing each of the individual resistors. R20 through R24 help in suppressing the circulating current between the gate of the MOSFETs, which may cause gate voltage ringing. To describe the mechanism, consider only two MOSFETs in parallel as shown in Figure 11 . Due to the layout, the parasitic common source inductance (CSI) of all the MOSFETs will not be equal. If both the MOSFETs are turned on the same instant and if the di/dt of the drain currents are equal, the voltage across CSI of both MOSFETs (VLs1 and VLs2) will not be equal. This drives a circulating current as shown in Figure 12. Another possibility is if there is a skew in the V_{DS} rise and fall time of the parallel MOSFETs, circulating current can flow through the C_{GD} as shown in Figure 12 and cause unwanted behavior. Using individual gate resistors help to suppress the circulating current and help damp unwanted ringing effects. Due to a misunderstanding, we failed to place split resistors, only one is placed at the output of the driver IC, parallel to a diode meant to provide quicker turn-off times. In the upcoming sections the case of the split resistors will be further examined.

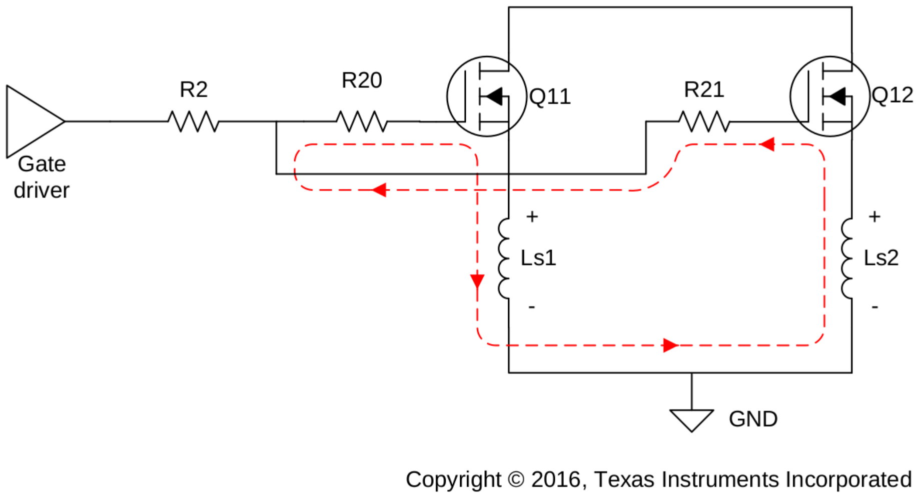


Figure 11: Gate circulating current due to CSI.

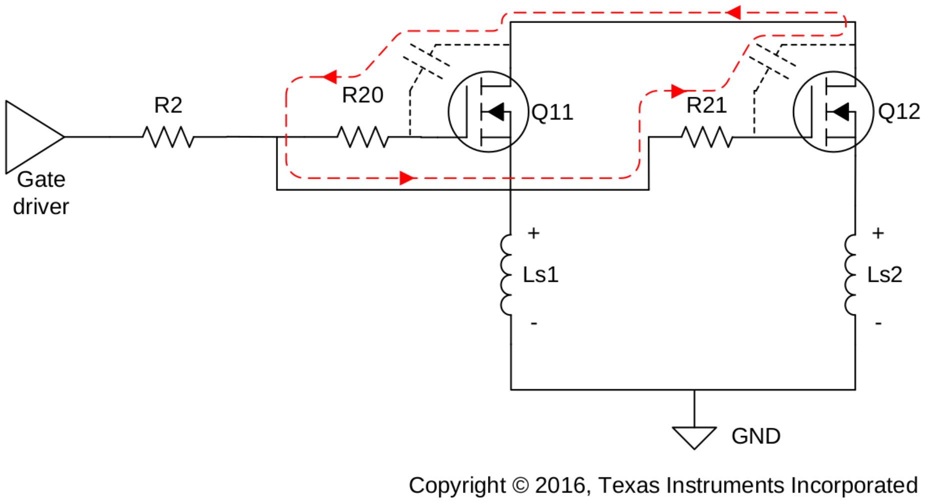


Figure 12: Gate circulating current due to skew.

5.5.2 Design

The V_{DS} fall time (the MOSFET switch-on time) is slowed down to $\approx 250ns$ and the rise time (switch-off) is $\approx 50ns$. The required resistor values can be calculated using Equation 17 through Equation 25.

To get the resistor values, first, the sink current must be calculated to draw the gate charge in the required switch-off time for the individual MOSFETs.

$$i_{Gsink/FET} = \left(\frac{Q_{GD}}{t_{sw\_off}}\right) \tag{17}$$

where

- $i_{Gsink/FET}$ is the required switch-off current
- Q_{GD} is the gate-drain charge of the MOSFET
- t_{sw_off} is the desired switch-off time

$$i_{Gsink/FET} = \left(\frac{V_{Miller}}{R_{G\_FET} + R_{G\_FET\_int}}\right) \tag{18}$$

Rearranging Equation 19 yields the following:

$$R_{G\_FET} = \left(\frac{V_{Miller}}{i_{Gsink/FET}} - R_{G\_FET\_int}\right) \tag{19}$$

where

- R_{G_FET} is the resistor for the individual MOSFETs
- V_{Miller} is the Miller-plateau of the MOSFET
- $R_{G\_FET\_int}$ is the internal gate-source resistance of the MOSFET

Now the switch-on current and common resistance can be calculated:
Taking after Equation 17:

$$i_{Gsource/FET} = \left(\frac{Q_{GD}}{t_{sw\_on}}\right) \tag{20}$$

$$i_{Gsource/FET} = \frac{\frac{V_G - V_{Miller}}{R_{pullup} + R_G}}{N} \tag{21}$$

and

$$R_G = R_{G\_common} + \frac{R_{G\_FET} + R_{G\_FET\_int}}{N} \tag{22}$$

Substituting Equation 22 into Equation 21 and rearranging them yields the following:

$$R_{G\_common} = \frac{i_{Gsource/FET} \cdot N}{V_G - V_{Miller}} - R_{pullup} - \frac{R_{G\_FET} + R_{G\_FET\_int}}{N} \tag{23}$$

It is easy to see that when $N = 2$, the total current to be supplied and drawn by the driver is double of the individual MOSFETs.

$$i_{Gsource\_total} = i_{Gsource/FET} \cdot 2 \tag{24}$$

$$i_{Gsink\_total} = i_{Gsink/FET} \cdot 2 \tag{25}$$

The following variables and their values, obtained from their respective datasheets or calculated, were used:

- $i_{Gsource/FET} = 136mA$ is the required switch-on current per MOSFET
- $i_{Gsource\_total} = 272mA$ is the required total switch-on current
- $i_{Gsink/FET} = 680mA$ is the required switch-off current per MOSFET
- $i_{Gsink\_total} = 1.36A$ is the required total switch-off current
- $Q_{GD} = 34nC$ is the gate-drain charge of the MOSFET
- $t_{sw\_on} = 250ns$ is the desired switch-on time
- $V_G = 12V$ is the gate voltage applied
- $V_{Miller} = 4.4V$ is the Miller-plateau of the MOSFET
- $R_{pullup} = 2.6\Omega$ is the gate driver IC internal pullup resistance
- $R_G = 25.34\Omega$ is the equivalent resistance seen by the gate driver
- $R_{G\_common} = 22.1\Omega$ is the common resistance value
- $R_{G\_FET} = 5.17$ is the resistor for the individual MOSFETs
- $R_{G\_FET\_int} = 1.7\Omega$ is the internal gate-source resistance of the MOSFET
- $N = 2$ is the number of parallel MOSFETs

The diode parallel to the common resistor introduces a forward voltage drop of up to $900mV$ when draining the gate of charge. It should not cause problems, since as Figure 13 shows, the drain current of the MOSFET is insignificant in that region.

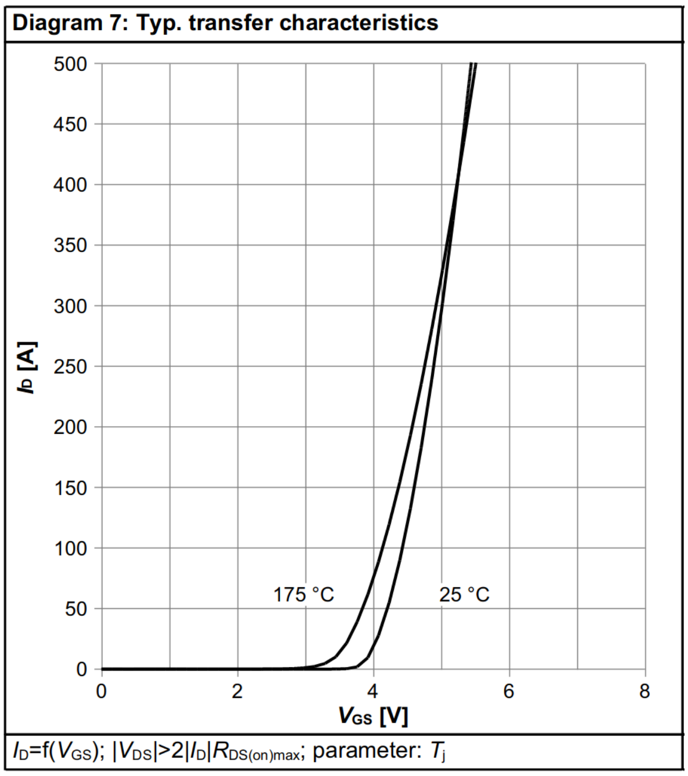


Figure 13: Typical transfer characteristics of a MOSFET

5.6 Power and Driver board layout

This section describes the design and layout considerations for both boards.

5.6.1 Overview

The proper PCB layout of the driver board is just as essential as that of the power board. To keep the price of the PCB's down, both the power and the driver board has to be kept under $100mm \times 100mm$. The layer number of the boards is also a pricey concern. Read more on PCB selection in Section 5.4.2. The driver board was designed on 2-layer FR4 1 oz. material.

The power board was first laid out to get a precise and optimal placement of the critical components, power rails, outputs and interboard connectors. After that came defining the contour of the driver board, precise placement of interboard connectors and crude placement of components. It is vital for all signals to have their firm ground plane under them to act as reference. That is why the bottom side of the driver board was partitioned according to the components on the top side to have their respective ground planes. The surface of the copper fills were reduced as much as possible to counter capacitive noise propagation. Supply-ground decoupling capacitors were placed as close to their supported components as possible. 0805 and 1206 size code footprints were placed to have more possibilities for modification or rework if need be. A cut-out over the power rails provides space for the electrolyte capacitors attached to the bus bar. For approximate mechanical construction, see Figure 14 and Figure 15.

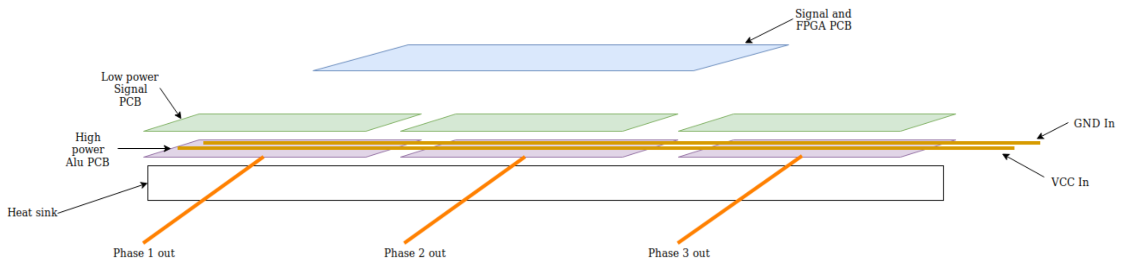


Figure 14: Perspective view of mechanical construction

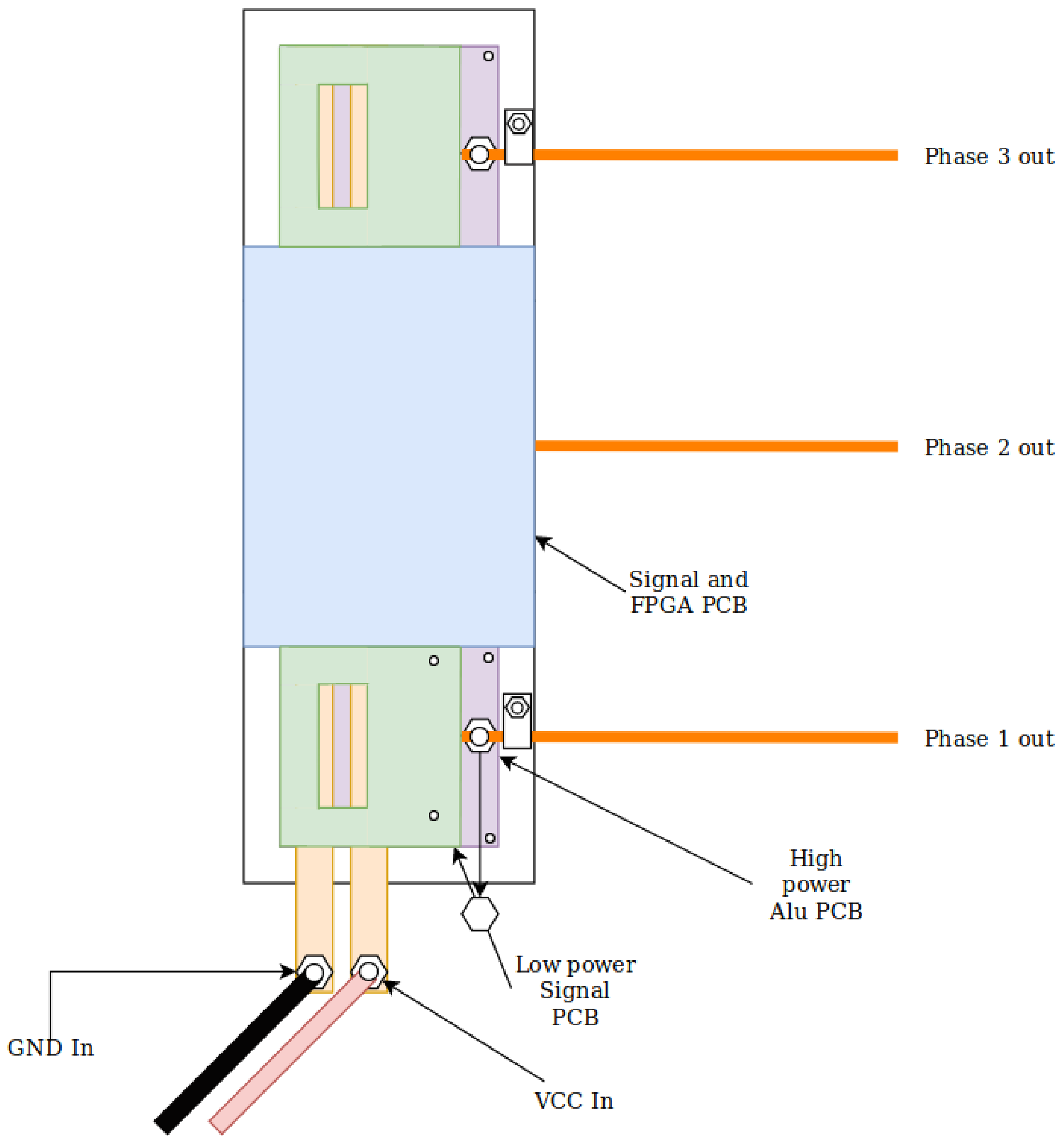


Figure 15: Top view of mechanical construction

5.6.2 Power board

Individual PCB's of $100mm \times 65mm$ were chosen for each phase which makes it possible to keep the board within the general size constraints and fit the available $100mm \times 200mm$ heat sink. The three boards are interconnected with the power rail busbars. Screw holes in each provide fixture points to help with assembly. Only the top side of the aluminum PCB has a copper layer which is divided into 3 regions. The first is the battery positive side, it is connected to the drain of the high-side transistors. The common region is the output, it gives place to the high-current screw terminal. The source of the low-side transistors is connected to the battery negative. To provide the best supply-ground capacitive decoupling and keep the circuit highlighted in Figure 16 as short as possible, capacitors are placed along the border of the VCC and GND planes. Electrolyte capacitors are attached to the busbars. It is important to have as high capacitance as possible while keeping the ESR (Equivalent Series Resistance) and stray inductance to a minimum. *UVZ2A471*[12] was therefore chosen. Its small form factor provides all the aforementioned advantages. The MOSFETs are as close to each other as possible. The space is limited by the presence of the interboard connector conducting the gate current. Having the shortest possible traces for source and gate helps with reducing the stray capacitance and inductance of the line. It is important in order to have uniform turn-on and turn-off times. Screw holes are also present to help press the power board against the heat sink for better thermal coupling.

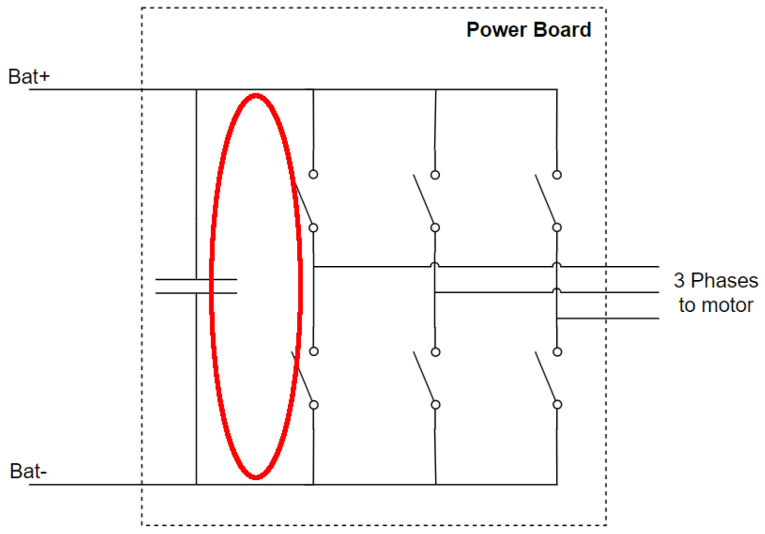


Figure 16: Route of current through stabilizer capacitors and MOSFETs

Subconclusion

An analog interface that scales the current transducer and torque sensors signals has been made and a PCB produced. A power board for each phase is designed with four SMD MOSFET's used per phase which is made possible by the PCB's being of aluminum. A driver board is then designed to sit on top of the power board and switch the transistors. The board has cut-outs for the DC-link capacitors.
Component lists for all boards, for the heat sink and for additional material can be found in appendix B.

6 Control

The section will go through which control type has been chosen, how it works and how it is used. Then the motor model is setup and explained and lastly the motor controller is designed.

6.1 Field Orientated Control

Multiple ways exist to control motors but in this project Field Orientated Control (FOC) has been chosen for its simplicity and ease of implementation. The idea behind FOC is to transform the three phase currents into a single vector in a rotating frame of reference that rotates with the same speed as the rotor. Perform the control in the rotating frame and then transform the controller signals back out into three phase signals which results in a control loop as can be seen on figure 17.

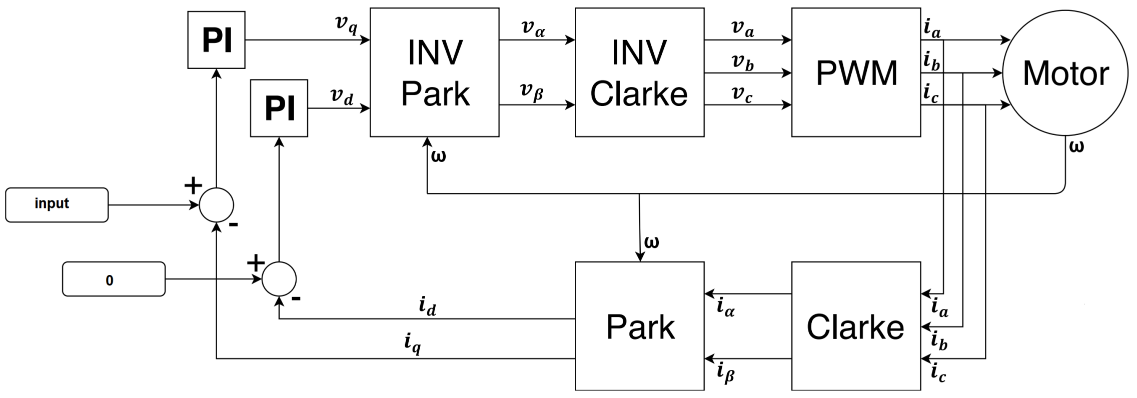


Figure 17: Motor model used for the control as an overview.

On the right the motor can be seen. The phase current out of the inverter is transformed into a rotating frame of reference by first performing a Clarke Transformation and then a Park Transformation. Out comes two signals i_d and i_q which goes into two individual controllers. FOC usually use PI controllers because only a proportional and a integrator part is needed. The motor torque is biggest when the electric field is perpendicular to the rotor position. The i_d signal should always be 0 because it resembles electric field not perpendicular to the rotor. Therefore it is subtracted with 0 which makes i_d the error and the signal is out into the PI controller. The i_q resembles the electric field perpendicular to the rotor which is what produces the torque. Therefore this signal is compared with the input which is the target torque that comes from the torque pedal. After the PI controllers the two control signals in the rotating frame of reference are transformed back out into three phases with an Inverse Park Transformation and an Inverse Clarke Transformation. The signals can then be used to control the inverter.

6.1.1 Clarke and Park Transformations

With Clarke and Park transformations it is possible to convert three rotating phase current into a static vector in a rotating frame. When controlling the motor, it makes it simpler to control a single vector in a rotating frame, rather that three rotating vectors.

6.1.1.1 Clarke Transformation

The Clarke transformation converts three rotating phase currents to one rotating vector in the alpha-beta frame. The Clarke transformation is performed by using the two equations 26.

$$i_\alpha = \frac{2}{3} \cdot i_a - \frac{1}{3} \cdot i_b - \frac{1}{3} \cdot i_c, \qquad i_\beta = 0 \cdot i_a + \frac{1}{\sqrt{3}} \cdot i_b - \frac{1}{\sqrt{3}} \cdot i_c \tag{26}$$

i_a, i_b and i_c are the three rotating phase currents, and i_α and i_β are the two components of the rotating vector in the alpha-beta frame. It can also be written in matrix form as in equation 27.

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} \frac{3}{2} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \tag{27}$$

6.1.1.2 Park Transform

To convert the one rotating vector in the alpha-beta frame to a static vector in the rotating dp-frame, the Park transformation is used. The park transformation is performed with equation 28.

$$i_d = cos(\omega t) \cdot i_\alpha + sin(\omega t) \cdot i_\beta, \qquad i_q = -sin(\omega t) \cdot i_\alpha + cos(\omega t) \cdot i_\beta \tag{28}$$

i_d and i_q are the components of the static vector in the rotating dq-frame. ωt is the angle of the electrical field, based on the angle of the rotor. The Park transformation can be seen on matrix form in equation 29.

$$\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} cos(\omega t) & sin(\omega t) \\ -sin(\omega t) & cos(\omega t) \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \tag{29}$$

6.1.1.3 Inverse Park Transform

With the inverse Park transformation a static vector in a rotating dq-frame can be converted back into a rotating vector in a alpha-beta frame and can be done with equation 30.

$$i_\alpha = cos(\varphi) \cdot i_d - sin(\varphi) \cdot i_q, \qquad i_\beta = sin(\varphi) \cdot i_d + cos(\varphi) \cdot i_q \tag{30}$$

Equation 31 is the inverse Park transformation in matrix form.

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} cos(\omega t) & -sin(\omega t) \\ sin(\omega t) & cos(\omega t) \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} \tag{31}$$

6.1.1.4 Inverse Clarke Transform

The inverse Clarke transformation converts a rotating vector in the alpha-beta frame into three phase signals. The equation for this transformation is equation 32.

$$i_a = i_\alpha, \qquad i_b = -\frac{1}{2} \cdot i_\alpha + \frac{\sqrt{3}}{2} \cdot i_\beta, \qquad i_c = -\frac{1}{2} \cdot i_\alpha - \frac{\sqrt{3}}{2} \cdot i_\beta \tag{32}$$

The matrix form of the inverse Clarke transformation can be seen in equation 33.

$$\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \tag{33}$$

6.2 Motor model

A model of the motor needs to be made for the system model. The motor is a PMSM, which means it is driven by three phase currents. To simplify the model, the equivalent circuit for the d- and q-directions are drawn, and a model is made, based on these. In that way the PMSM can be controlled like an DC-machine.

6.2.1 Motor model in the d-direction

The equivalent circuit for the d-direction voltage v_d of the PMSM is shown on figure 18.

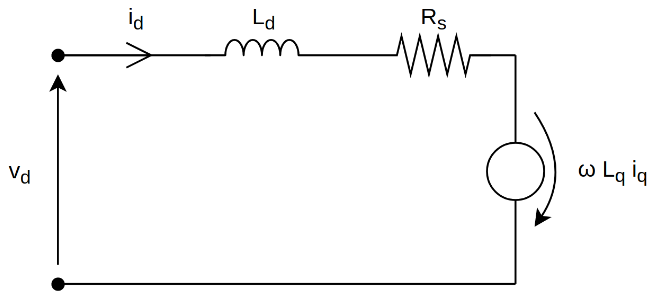


Figure 18: D-dierection equivalent circuit for the PMSM.

From the equivalent diagram the d-direction voltage can be described as in equation 34.

$$v_d = L_d \frac{di_d}{dt} + R_s i_d - \omega L_q i_q \tag{34}$$

L_d and L_q are respectively the d- and q-direction equivalent stator inductances for the motor, i_d and i_q are respectively the d- an q-direction currents, R_s is the stator resistance for the motor, and ω is the rotational speed of the rotor.
If i_d from the $L_d \frac{di_d}{dt}$ part is isolated and Laplace transformed it result in equation 35.

$$I_d = \frac{1}{S} \frac{1}{L_d} (V_d - I_d R_d + \omega L_q I_q) \tag{35}$$

From equation 35 the model of the d-direction of the motor is determined. That can be seen on figure 19.

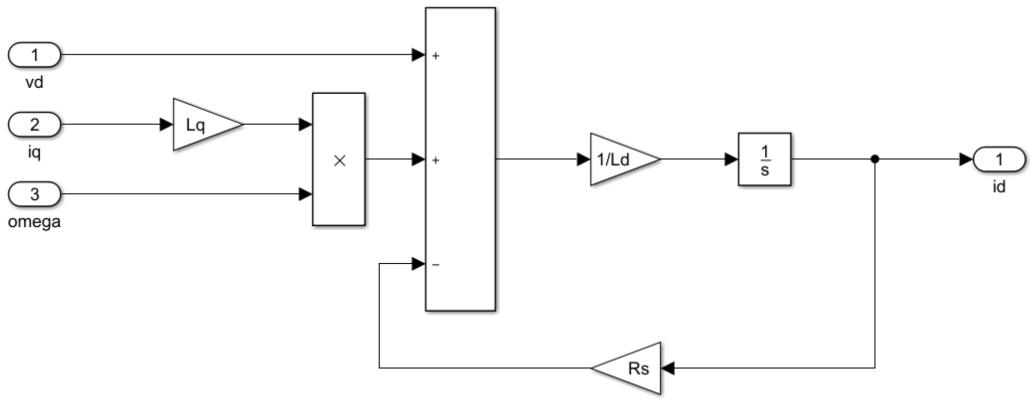


Figure 19: Simulink model of the d-direction of the motor model.

6.2.2 Motor model in the q-direction

The equivalent circuit for the q-direction voltage v_q is shown on figure 20.

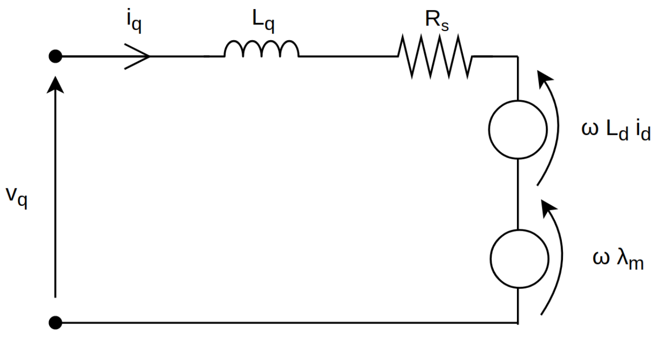


Figure 20: Q-direction equivalent circuit for the PMSM.

The voltage in the q-direction can be described from equation 36.

$$v_q = L_q \frac{di_q}{dt} + R_s i_q + \omega L_d i_d + \omega \lambda_m \tag{36}$$

λ_m is the flux linkage.
The i_q from the $L_q \frac{di_q}{dt}$ part is isolated and Laplace transformed which results in equation 37.

$$I_q = \frac{1}{S} \frac{1}{L_q} (V_q - I_q R_q - \omega L_d I_d - \omega \lambda_m) \tag{37}$$

From equation 37 the q-direction model can be produced. The model can be seen on figure 21.

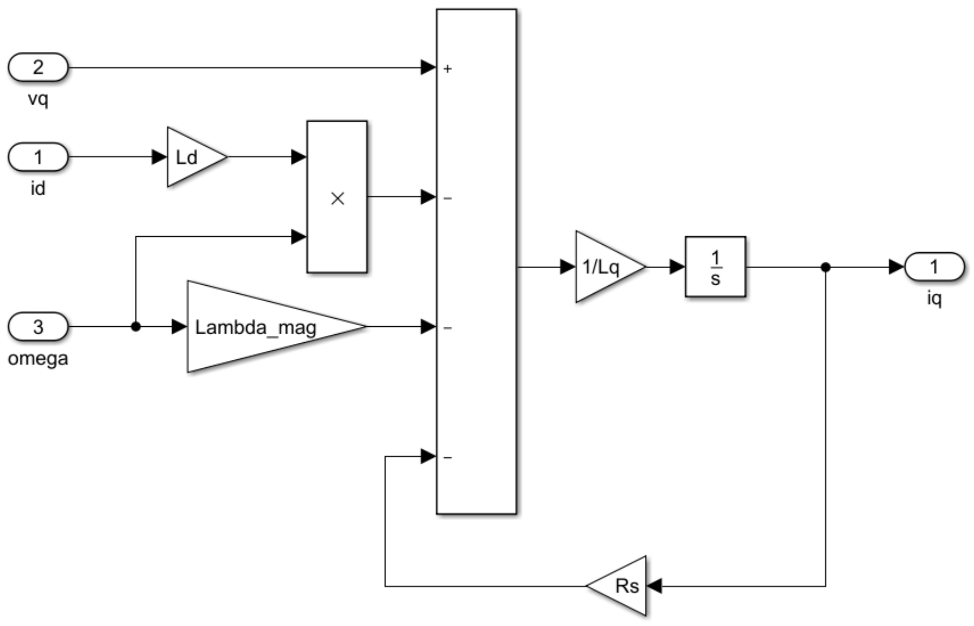


Figure 21: Simulink model of the q-direction of the motor model.

6.2.3 Mechanical model

For a multiple pole synchronous motor, the torque produced from the electrical field can be found from equation 38.

$$T_e = \frac{3P}{2}\big(\lambda_m i_q + (L_d - L_q)i_q i_d\big) \tag{38}$$

Where P is the number of pole pairs.
Because L_q is higher than L_d, it can be seen from equation 38 the current running in the d-direction produces negative torque and is therefore reducing the output torque.
The total torque at the shaft of the motor, will be the sum of the torque that the electrical field produces, the torque produced by the friction in the motor, and maybe an external load torque. The total torque can be set equal to the inertia of the system multiplied with the acceleration. As seen in equation 39.

$$J\frac{d\omega}{dt} = T_e - B\omega - T_{load} \tag{39}$$

J is the inertia, B is the friction coefficient, and T_{load} is the torque from the external load. Equation 39 is rewritten and combined with equation 38 to a equation for the rotational speed. The resulting equation is Laplace transformed, and the result is equation 40.

$$\omega = \frac{1}{S}\frac{1}{J}\left(\frac{3P}{2}\big(\lambda_m I_q + (L_d - L_q)I_q I_d\big) - B\omega - T_{load}\right) \tag{40}$$

From equation 40 a model can be created with the torque, rotational speed, the angle of the motor as output, and the d- and q-current as inputs. The model can be seen on figure 22.

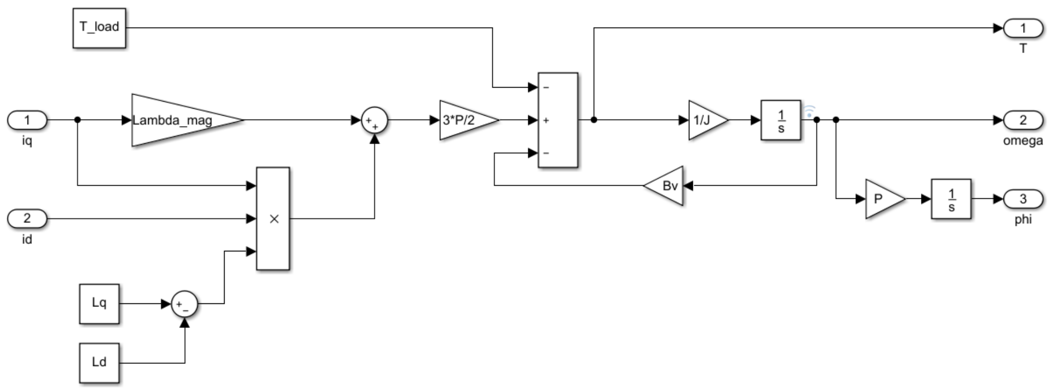


Figure 22: Simulink model of the motor with torque, rotational speed and the position of the rotor as output and the the d- and q-current as input.

6.2.4 Complete motor model

The three models, for the d-direction, the q-direction and the mechanical part, are put into subsystems and connected as can be seen on figure 23.

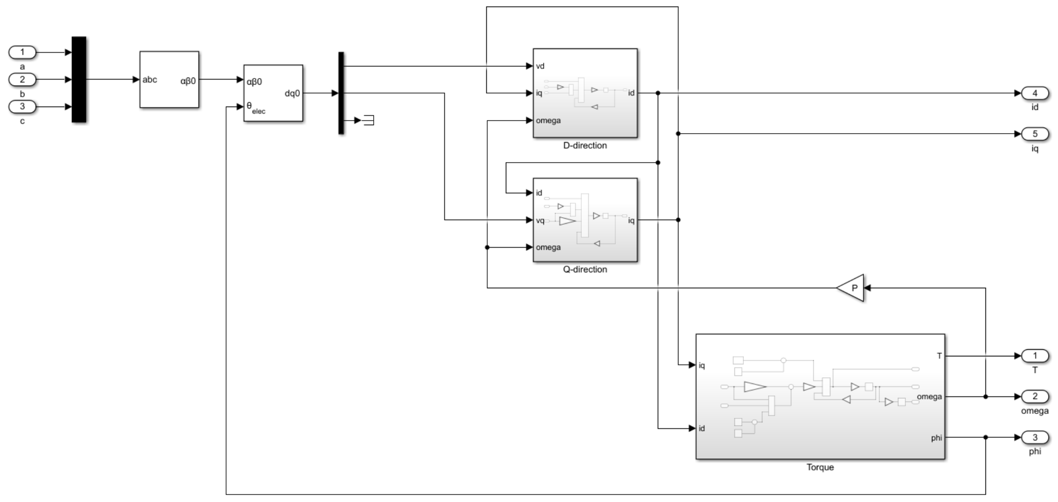


Figure 23: Simulink model of the full motor model.

The motor model is in the dq-frame which means the three rotational phases voltages are converted to the dq-frame in the beginning of the model. The model outputs the d- and q-current, the torque, speed and angle of the motor. The rotational speed out of the motor model is the mechanical speed, and is therefore converted to the electrical speed before it is put into to the d- and q-models.

6.3 Control system

For controlling the motor, the output d- and q-current of the motor are feedback to a PI-controller for each of them. The d-current is desired to be zero, and is therefore compared with zero before the PI-controller. The q-current is desired to be set to

the reference point set by the input from the torque pedal. The torque reference input is therefore converted to a current, which can be compared with the feedback q-current from the motor model.
After the PI-controller the d- and q-voltages are converted back into three phases with an Inverse Clarke and an Inverse Park transformation. The three phase voltages are then put into the motor model.
The full model of the system can be seen on figure 24.

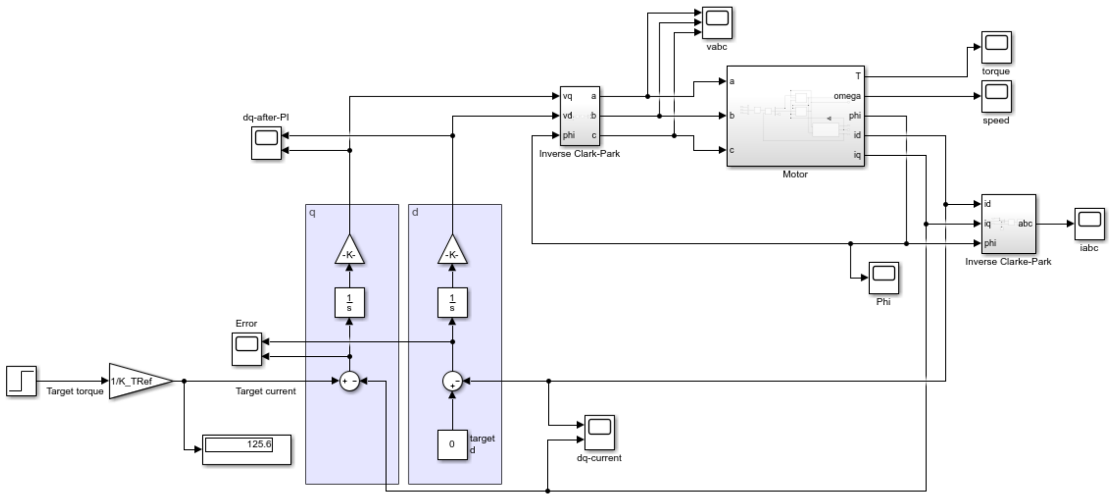


Figure 24: The entire control model, with PI controller and motor model.

6.4 PI-controller

To set the PI-controller the transfer functions for the d- and q-direction is determined, with the voltages as input and the currents as output. This is done based on the equations for the voltages in the d- and q-direction, equation 34 and 36. A transfer function can only have one input and one output and therefore the two equations, 34 and 36, are shortened. The new equations can be seen in equation 41 and 42.

$$v_d = L_d \frac{di_d}{dt} + R_s i_d \tag{41}$$

$$v_q = L_q \frac{di_q}{dt} + R_s i_q \tag{42}$$

As seen in the two equations, 41 and 42, the parts in the equations depending on other variables than the current is neglected. The equations are Laplace transformed and converted to the transfer functions for the two systems.

$$G_d = \frac{I_d(s)}{V_d(s)} = \frac{\frac{1}{\frac{L_d}{R_s}}}{R_s\left(\frac{1}{\frac{L_d}{R_s}} + s\right)} \tag{43}$$

$$G_q = \frac{I_q(s)}{V_q(s)} = \frac{\frac{1}{\frac{L_q}{R_s}}}{R_s\left(\frac{1}{\frac{L_q}{R_s}} + s\right)} \tag{44}$$

Equation 43 and 44 are the transfer functions for the motor.
To set the PI-controller the cutoff frequency is determined. The cutoff frequency, ω_{cutoff}, is found from the time constant of the motor.

$$\omega_{cutoff} = \frac{1}{\tau} \tag{45}$$

Where τ is the time constant of motor. The time constant of the motor in the d- and q-direction is described in equation 46.

$$\tau_d = \frac{L_d}{R_s}, \qquad \tau_q = \frac{L_q}{R_s} \tag{46}$$

From the cut off frequency the crossover frequency is determined.

$$\omega_c = k_1 \cdot \omega_{cutoff} \tag{47}$$

Where k_1 is a constant which can be changed when tuning the controller. As the next step the gain margin for the two systems, G_d and G_q, multiplied with the transfer function for a PI controller, without the gain. The transfer function for the PI-controller without the gain can be seen in equation 48.

$$G_i = \frac{\tau s + 1}{\tau s} \tag{48}$$

The proportional part of the PI-controller is then found from the gain margin, GM.

$$K_p = \frac{1}{GM} \tag{49}$$

The k_1 is tuned, so that the system gets a phase margin around 70°. This amount of phase margin is desired because it results in the fastest response without given any overshoot. From this requirement the K_p is found to be $0.78 \cdot 10^{-3}$ for the d-direction and $0.83 \cdot 10^{-3}$ for the q-direction. The phase margin is 74.1 for the d-direction and 72.3 for the q-direction. A step response for the PI-controller on the motor can be seen figure 25.

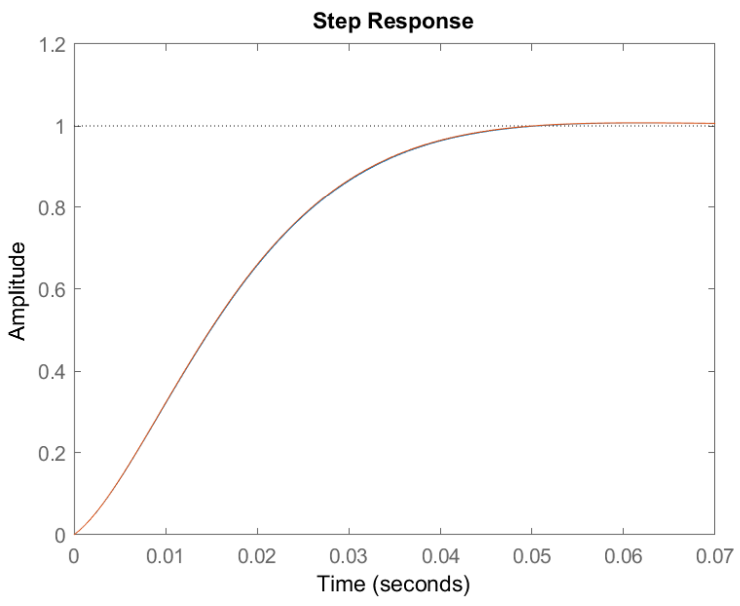


Figure 25: Step response for the controlled motor. The curve for d- and q-direction is placed in top of each other

6.5 Simulation

The model is tested with a step from 0 to $47.8Nm$, which correspond to a target q-current of $300A$. At the start the load torque is set to $20Nm$, and after $0.5s$ it is increased to $40Nm$.
The three phase currents produced can be seen on figure 26.

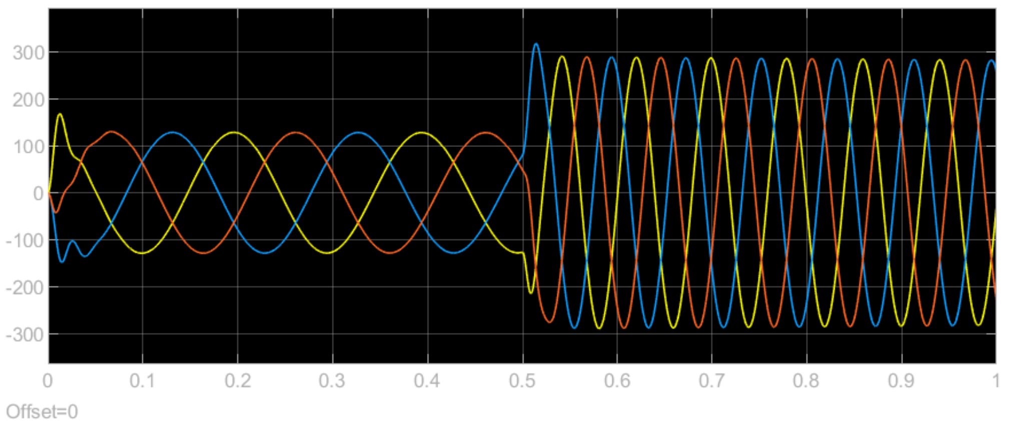


Figure 26: The ABC current from the motor model.

As it is seen on figure 26, the amplitude of the current increases when a bigger load is put on the motor.
The d- and q-current can be seen on figure 27.

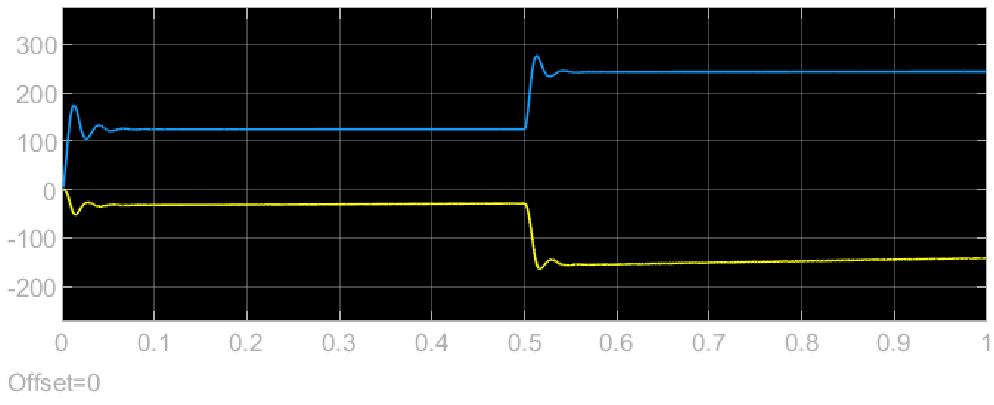


Figure 27: The d- and q-current from the motor model. The yellow curve is the d-current and the blue curve is the q-current.

On figure 27 it can be seen that the q-current is not going toward the set reference. It can also be seen that the d-current is very slowly going towards zero.
On the figure 28 and 29 the speed and the torque of the motor can be seen.

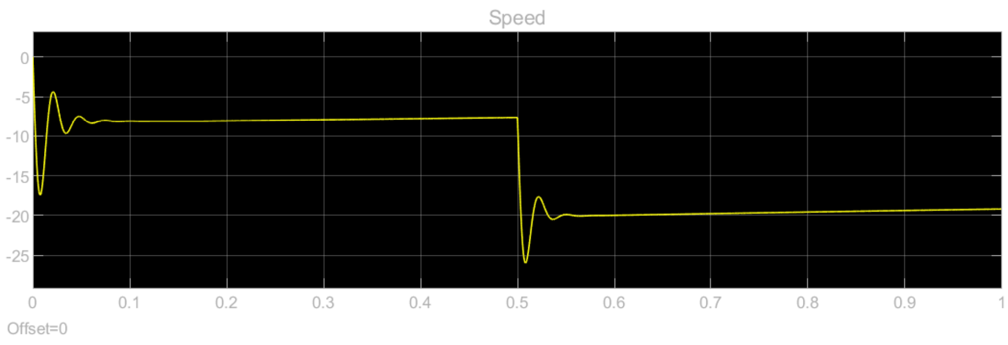


Figure 28: The speed of the motor.

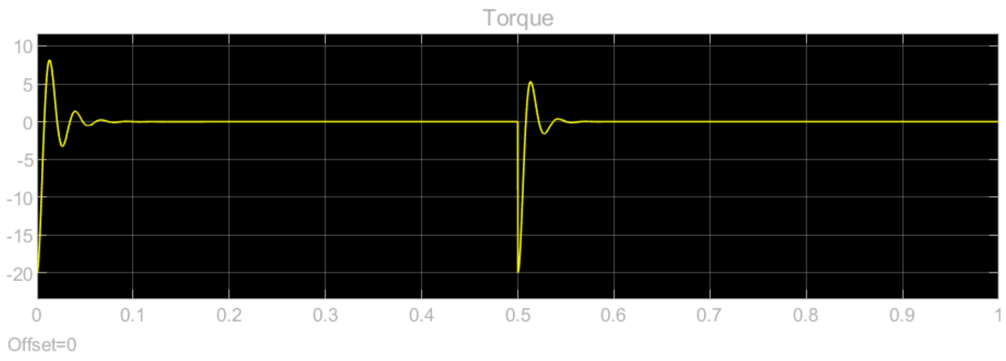


Figure 29: The torque of the motor.

Subconclusion

A model of the PMSM is made in the dq-frame, and it is controlled with field orientated control. For converting the three rotating phases going into the motor, to a d- and q-vector in a rotating frame, the Clarke and Park transformation is used. A PI-controller is used to control the d- and q-currents in the motor. Some minor problems have resulted in the model not working as expected.

7 Embedded system

The embedded system's task is to handle the control for the inverter.
The section is divided into three subsections. First the overall system architecture, then the low-level drivers implemented in the FPGA part of the controller and lastly the processing system and interface with a computer.

7.1 Software structure

The control is implemented on a Xilinx Zybo board which is a controller consisting of a FPGA part and a dual-core ARM Cortex-A9 processor.
Low-level drivers are implemented in the logic to allow the modules to run in parallel. The field orientated control as well as the interface to a PC is implemented on the processor which gives the possibility to have a higher abstraction.
An overview of the system can be seen on figure 30. In the top is the processing system (PS) with an Interrupt Service Routine (ISR) and the control loop. The control loop output its result into a piece of block RAM from where the programmable logic (PL) can access it. The processing system also manages the interface to the PC which is done through UART.

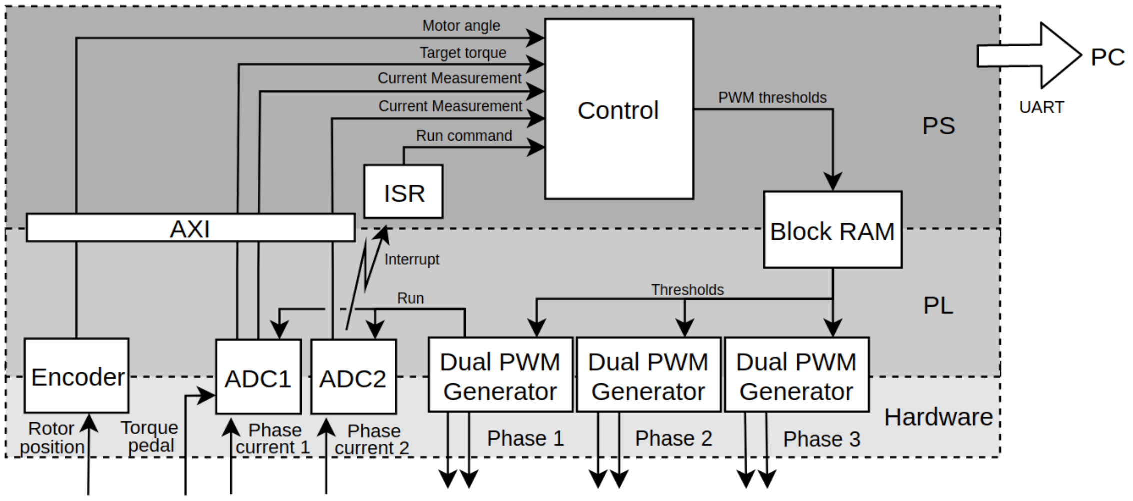


Figure 30: Embedded system overview

In the programmable logic the drivers for three dual PWM generators are implemented. Each PWM generator control one phase in the inverter. The PWM generators all generate a pulse in the middle of their PWM periods. One of these are parsed to the ADC's and used to trigger a reading of the phase currents and the torque pedal. When the ADC's are done converting all inputs an interrupt is produced triggering the ISR in the processing system.
The ADC readings as well as the encoder readings are parsed to the processing system through the AXI interface.
On figure 31 the flowchart of the control loop implemented on the processing system can be seen. The system is waiting to receive the run command from the ISR which is triggered once every PWM period, see section 7.3.1 for more about the ISR.

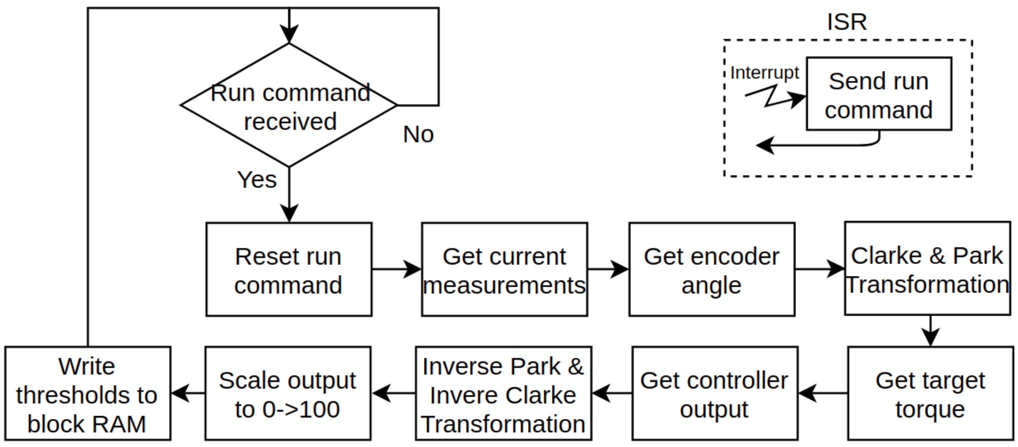


Figure 31: Processing System Overview

When the run command is received it is first reset to get ready for the next command. Then the current measurements from the two ADC's are read into the system as well as the rotor angle from the encoder. A Clarke and a Park Transformation is performed. The torque pedal position is read and the two PI controllers as discussed in section 6.3 perform the control. The result from the controllers are then transformed with an Inverse Park and an Inverse Clarke Transformation which produces three control signals. These signals are then linearly mapped to the range $0 \rightarrow 100$, which is what the PWM generators are compatible with. The thresholds are then written to a piece of block RAM shared between the PS and the PL. The PS will then wait for the next run command.

7.2 Programmable Logic (PL)

To take advantage of the parallel and super fast nature of FPGA logic all the low-level drivers are placed in logic. The section will go through the development and use of the modules used which includes a PWM module, an encoder module and an ADC module.

7.2.1 PWM module

Each phase of the inverter is controlled by a PWM signal for the high-side and low-side of each leg.

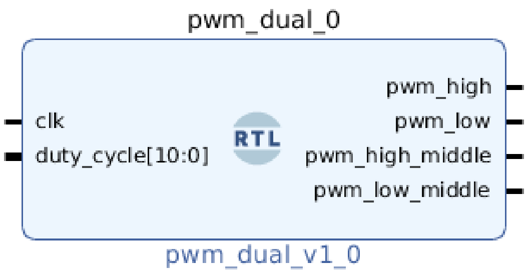


Figure 32: The PWM module made to control both high-side and low-side transistors of a leg in the inverter.

The module shown on figure 32 is made to control a whole leg. Three instances of the module can be used to control all three phases.
The module takes in a clock signal and a target duty cycle. The module then output two PWM signals inverse of each other and two pulse signals each sending out a pulse in the middle of their respective PWM signal.

Counter

A counter is used for implementing the PWM. For every rising edge of the input clock the counter takes one step up or down depending on its current counting direction. When it reaches one of its outer limits the counting direction is switched as can be seen on figure 33.

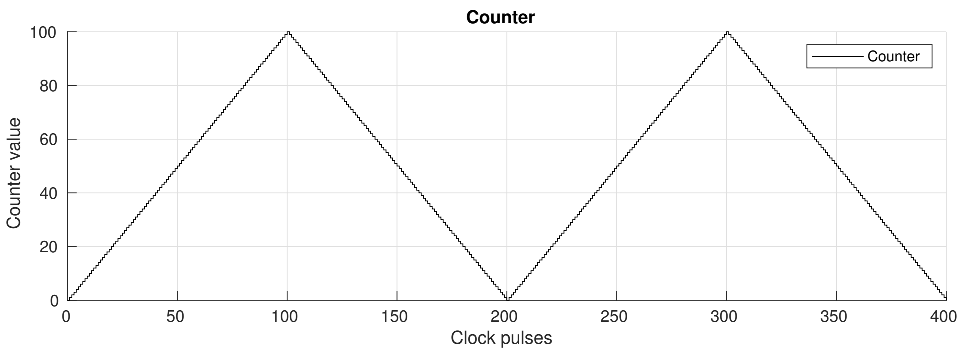


Figure 33: How the counter is counted up and down over time.

The resolution of the PWM duty cycle is chosen to be 1% which means the counter takes 100 steps from minimum to maximum. The PWM is made as a phase correct PWM which means it will have the same phase no matter the duty cycle. Therefore the counter counts both up and down per period which results in 200 steps per period.
The signal out of the PWM generator should be the same frequency as the inverter components are designed for in the earlier sections which is $10kHz$. The input clock is already prescaling the output frequency by the number of counter steps but this is not enough and therefore in order for the PWM to have the correct frequency an additional prescaler is added. The prescaler value is found with equation 50c.

$$f_s = \frac{f_{pl}}{c_{steps} \cdot x \cdot 2} \tag{50a}$$

$$x = \frac{f_{pl}}{f_s \cdot c_{steps} \cdot 2} \tag{50b}$$

$$x = \frac{125MHz}{10kHz \cdot 200 \cdot 2} = 31.25 \sim 31 \tag{50c}$$

$$\frac{125MHz}{200 \cdot 31 \cdot 2} = 10.08kHz \tag{50d}$$

The prescaler is constructed with a counter counting up a value, x. When the value is reached the output clock is flipped and the counter is reset. Such a prescaler has

a build in prescaling of 2 which is added besides the additional prescaler leading to equation 50a.
Where f_s is the output frequency of the PWM signal, c_{steps} is the number of counter steps in a period, f_{pl} is the logic clock which is $125MHz$ and x is the additional prescaler.
With the additional prescaler the PWM module outputs PWM signals with a frequency of $10.08kHz$.

PWM

To handle both a high-side PWM and low-side PWM each of the signals have a threshold and the output signal changes from high to low or opposite when the counter crosses the threshold as can be seen in figure 34. The two thresholds are spaced out with a static deadtime between them which is discussed later in this section. The deadtime on figure 34 has been exaggerated to make it more visible.

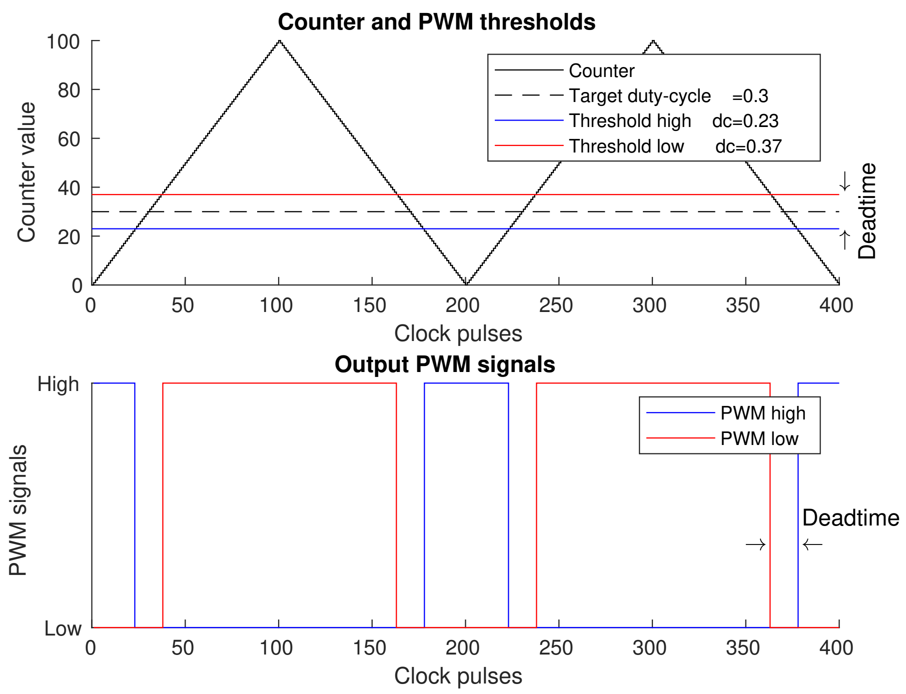


Figure 34: The top shows the target duty-cycle together with the thresholds compared to the PWM counter. The bottom shows the output PWM signals. The deadtime is exaggerated to make it more visible.

The high-side PWM is high when the counter is below the high-side threshold, $counter < th_{high}$, otherwise it is low.
The low-side PWM is high when the counter is above the low-side threshold, $counter > th_{low}$ otherwise it is low.

The VHDL code can be seen below.

```
-- Control of the high side PWM
pwm_high <= HIGH when (counter < threshold_high) else LOW;
-- Control of the low side PWM
pwm_low  <= HIGH when (counter > threshold_low) else LOW;
```

The VHDL code to control the PWM signals.

To avoid shorting the supply deadtime is inserted between turning off one transistor and turning on the other.
On figure 35 the conducted current for each transistor is drawn next to the PWM signals with deadtime being bigger than the transistor turn off time, $dead\ time > t_{turn\ off}$, which results in one transistor turning completely off before the other turns on.

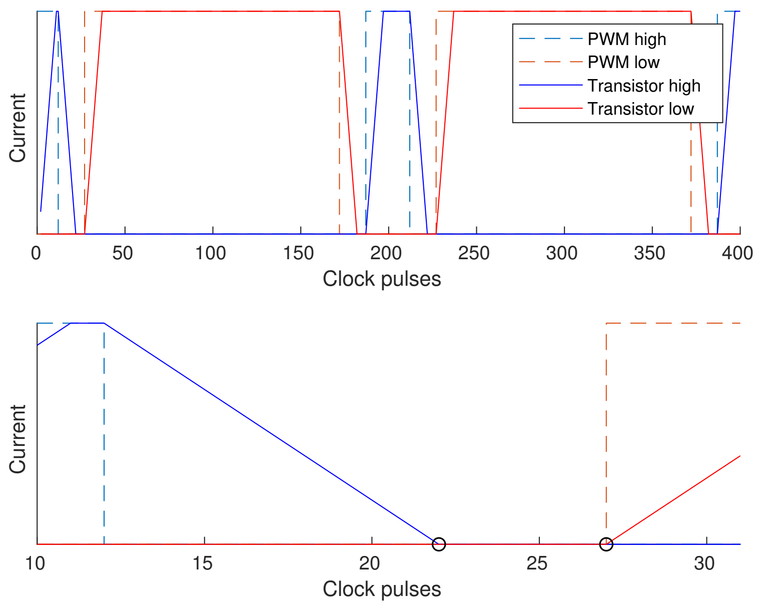


Figure 35: PWM with deadtime and transistor conduction curves for systems with $dead\ time > t_{turn\ off}$

On figure 36 the conducted current for each transistor is drawn next to the PWM signals but this time the deadtime is smaller than the transistor turn off time, $dead\ time < t_{turn\ off}$, which results in both transistors conducting at the same time. When both transistors conduct the supply is shorted which should be avoided especially when the supply consist of batteries.

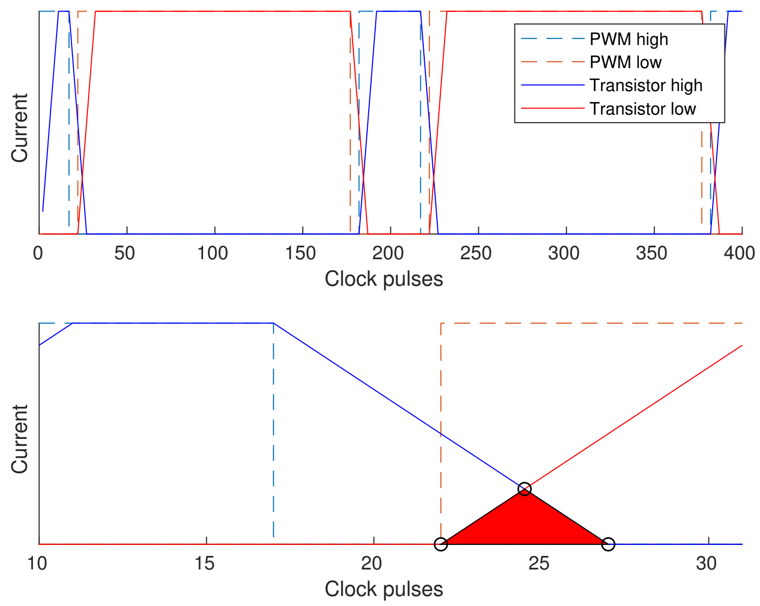


Figure 36: PWM with deadtime and transistor conduction curves for systems with $dead\ time < t_{turn\ off}$

It is therefore important to choose the deadtime big enough to avoid shorting but also not so big that the systems performance is unnecessarily reduced.

To avoid compromising the required deadtime at edge cases where one of the thresholds tries to move above 100% or below 0% the thresholds are found in two different ways.
When the high-side threshold is greater than the counter minimum plus half of the deadtime it is placed half of the deadtime below the threshold.

$$th_{duty\ cycle} \geq c_{min} + \frac{t_{dead\ time}}{2} \Rightarrow \tag{51}$$

$$th_{high} = th_{duty\ cycle} - \frac{t_{dead\ time}}{2} \tag{52}$$

Otherwise the threshold is clamped to the counter bottom edge $th_{high} = counter_{min}$ which is possible because the PWM is switched when $th_{high} > counter$. If the threshold comes too close to the edge the threshold will be clamped and the PWM will not switch. Had the PWM being switched when $th_{high} \geq counter$ this would not have worked.
The low-side threshold is placed half of the deadtime above the threshold when the duty cycle is at least half of the deadtime below the counter max.

$$th_{duty\ cycle} \leq c_{max} - \frac{t_{dead\ time}}{2} \Rightarrow \tag{53}$$

$$th_{low} = th_{duty\ cycle} + \frac{t_{dead\ time}}{2} \tag{54}$$

Otherwise the threshold is clamped to the top counter edge.
The VHDL code to find the two thresholds can be seen below. The variable *duty_cycle* is parsed into the PWM module. *DEADTIME*, *HALF_DEADTIME*, *COUNT_MIN* and *COUNT_MAX* are all constants.

```vhdl
-- Get half of deadtime
HALF_DEADTIME(6 downto 0) <= DEADTIME(7 downto 1);
-- Find the threshold for the high-side PWM
threshold_high <= duty_cycle - HALF_DEADTIME when
                (duty_cycle >= COUNT_MIN + HALF_DEADTIME) else COUNT_MIN;
-- Find the threshold for the low-side PWM
threshold_low  <= duty_cycle + HALF_DEADTIME when
                (duty_cycle <= COUNT_MAX - HALF_DEADTIME) else COUNT_MAX;
```

The VHDL code to find the thresholds for the high and low side PWM signals. The conditions and equations used are 51, 52, 53 and 54.

The turn off times for the transistors was found in section 5.4.4.2 to be $75ns$. The way deadtime is implemented it has to be defined as some even amount of ticks at the PWM frequency. The minimum deadtime step, dt_{res}, in this system is calculated with equation 55.

$$dt_{res} = \frac{T}{2} \cdot \frac{dt_{min}}{c_{max}} \tag{55a}$$

$$dt_{res} = \frac{2}{fs} \cdot \frac{dt_{min}}{c_{max}} \tag{55b}$$

$$dt_{res} = \frac{2}{10kHz} \cdot \frac{2}{100} = 4\mu s \tag{55c}$$

Which gives the system a safety margin of ~ 53 on the deadtime between the two PWM signals.

ADC Pulses

To trigger a new reading by the ADC the PWM generator module has a build-in pulse mechanism that outputs a pulse in the middle of each of the PWM periods as can be seen on figure 37. Only the pulse in the middle of the high-side PWM is used to trigger the ADC. The reason for measuring in the middle of the PWM is to avoid majority of the ringing produced on the signals from switching.

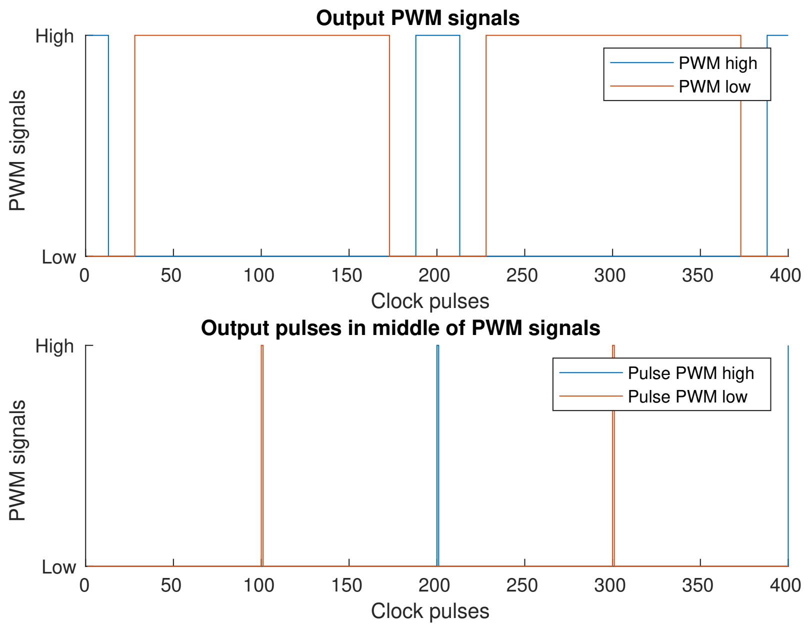


Figure 37: The top graph shows the high and low side PWM signals. The bottom graph shows the ADC pulses triggered in the middle of each PWM signal.

The code for evaluating the signal states can be seen below. When the counter reaches its bottom edge the high pulse is set high and as soon as the counter starts going up again the signal is set low. The low pulse is controlled in the same way but this happens at the high counter edge instead.

```
-- Output a pulse in the middle of the high PWM signal
pwm_high_middle <= HIGH when (counter <= COUNT_MIN) else LOW;

-- Output a pulse in the middle of the low PWM signal
pwm_low_middle <= HIGH when (counter >= COUNT_MAX) else LOW;
```

The VHDL code to control the ADC pulses.

Test of PWM module

To test the PWM generators a test scenario is set up. A simulated rotor angle and simulated currents are parsed into the control system and the outputs are measured with an oscilloscope. The maximum sine frequency expected out of the system is $333Hz$. To figure out how fast the simulated angle should change the time between each angle change is found. The angle is chosen to with 1 degree steps. That gives a change rate of:

$$change_{rate} = sin_{freq} \cdot 360^o \tag{56}$$

Where sin_{freq} is the target sine frequency. Which results in a time per angle of

$$t_{angle} = \frac{1}{change_{rate}} = \frac{1}{333Hz \cdot 360^o} = 8\mu s \tag{57}$$

Where t_{angle} is the time per angle.
No currents will run in the system so the currents need to be simulate as well. The angle is used to calculate three sinusoidal curves. The way the three currents are calculated can be seen in equation 58.

$$i_A = sin(angle), \quad i_B = sin(angle + 120^o), \quad i_C = sin(angle + 240^o) \tag{58}$$

Implementing this gives the PWM signal on one of the phases as seen in the top of figure 38 and the resulting sine curve on the bottom.

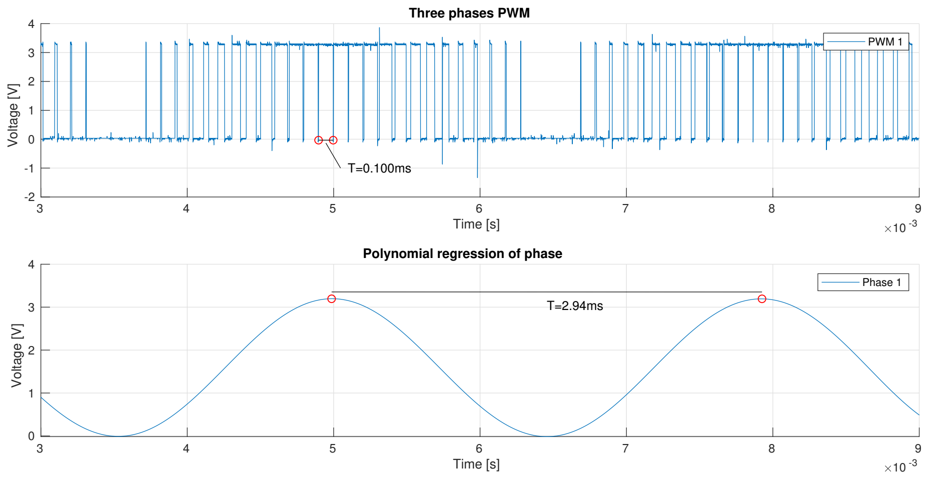


Figure 38: Testing of one phase. The top graph shows the PWM signals. The bottom graph shows the resulting sine curve found with a polynomial regression.

The PWM frequency can be found by measuring the time between the middle of two PWM periods as can be seen in equation 59.

$$f_s = \frac{1}{0.100 \cdot 10^{-3}} = 10kHz \tag{59}$$

With a precision of 6 decimals on the time period the frequency of the PWM signal turns out to be $10kHz$.
The resulting sine can be found by applying a polynomial regression to the PWM signal which results in a sinusoidal curve. The frequency is found from the inverse of the time period between two points exactly one period apart.

$$sin_{freq} = \frac{1}{2.94 \cdot 10^{-3}} = 340Hz \tag{60}$$

The frequency of the sine is $340Hz$.
On figure 38 only one of the phases are shown. In reality the system outputs three phases and all can be seen on figure 39. The periods in between the peak of the three phases are shown to determine the amount of phase shift. The time of one third of a period is calculated with 61.

$$\frac{1}{3} \cdot \frac{1}{340Hz} = 0.9804ms \tag{61}$$

By comparing the theoretical and measured periods between the phases it is determined that the phases are phase shifted with approximately $120^o = 2\pi/3\ rad$.

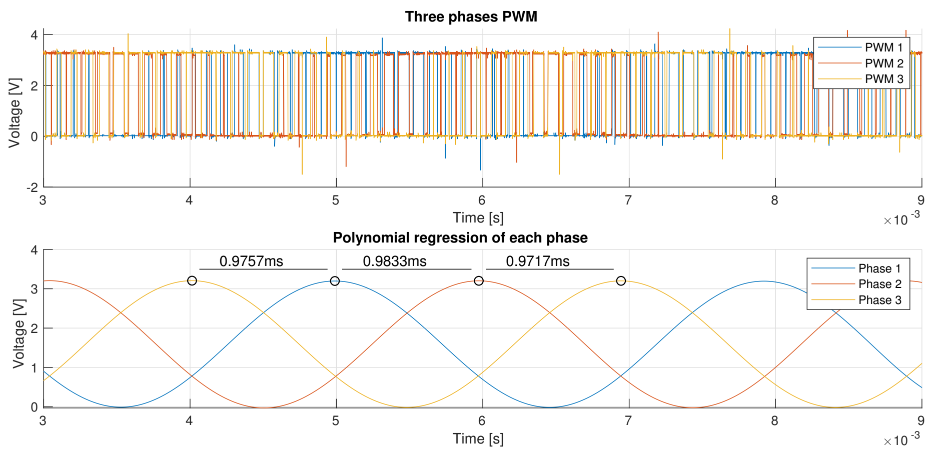


Figure 39: Testing of all three phases. The top graph shows the PWM signals. The bottom graph shows the resulting sine curves found with polynomial regressions.

7.2.2 Encoder

A driver module for the encoder on the motor was part of the material to get started on this project. [10]

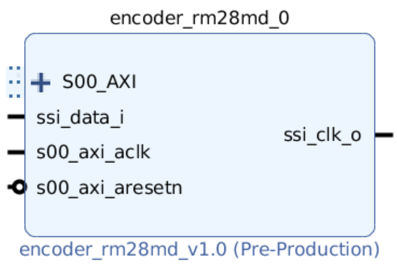


Figure 40: The encoder driver module.

The driver module, as seen on figure 40, supports an 8-bit encoder and takes care of handling the returned signal from the encoder itself. The module outputs the current rotor angle at a frequency determined by the PL clock and can be found with equation 62.

$$f_{encoder} = \frac{f_{pl}}{6708} = \frac{125MHz}{6708} = 18.634kHz \tag{62}$$

The system runs at $10kHz$ which means the control loop will always have a new encoder angle at every cycle.
The encoder is a 8 bit encoder which means it has a resolution of $2^8 = 255$ steps per revolution. The motor has 4 pole pairs which means the electric field inside the motor turns 4 times each time the rotor turns 1 time which means the resolution of the electric field angle is $1/4th$ of the mechanical rotor angle. The two angles are both shown in figure 41 where it can be seen that the electric angle moves 4 times faster than the rotor angle.

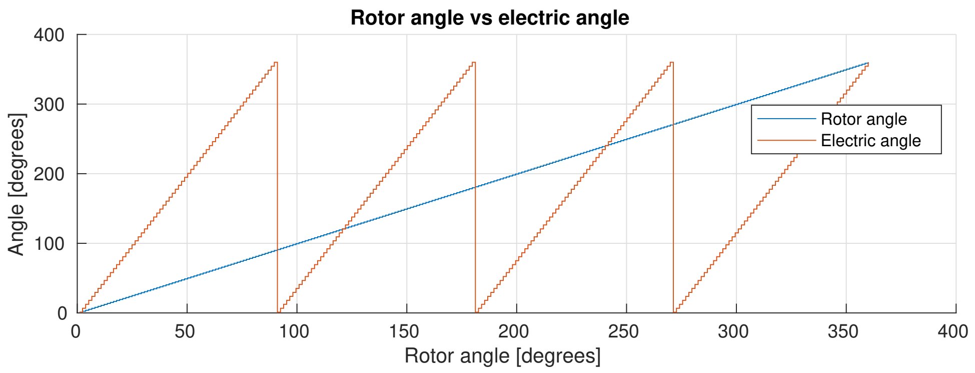


Figure 41: Electric angle shown again the rotor angle.

The resolution on the rotor angle and the electric angle can be calculated with equation 63 and equation 64 respectfully.

$$res_{rotor} = \frac{360^o}{2^8} = 1.412^o \tag{63}$$

$$res_{electric} = \frac{360^o}{2^6} = 5.625^o \tag{64}$$

The rotor position is returned as an 8 bit value from the encoder module, and before the angle can be used in the control system it is first mapped from $0 \rightarrow 255$ to an actual angle going from $0^o \rightarrow 359^o$. The mapping is done with equation 65.

$$angle = \frac{359}{255} \cdot position \tag{65}$$

The code for reading in the encoder angle can be found in appendix A.2.1.

7.2.2.1 Improve angle accuracy with linear interpolation

Low resolution and thereby big steps in the rotor angle results in uneven rotation of the motor especially at low speeds. To improve the angle before it is used in the control the real angle is approximated with the use of linear interpolation. It is assumed that the rotor speed is approximately the same for each encoder value step. This assumption is not completely correct because it would mean the motor does not change speed. The rotor acceleration and deceleration is assumed slow enough compared to the encoder frequency to not result in big errors.
The formula for linear interpolation [9] can be seen in equation 66.

$$y = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1) + y_1 \tag{66}$$

Figure 42 shows the scenario where the real angle, at point p_2, can be found from the measured angle, p_1, if x_1,x_2,y_1 and y_2 are known. The equation can be converted to fit the system by setting the two y values equal to the amount of change between the current and last step $\Delta a = y_2 - y_1$ and setting the x values to be the step width $x_{step} = x_2 - x_1$. Assuming that the speed is approximately the same for the current step as it was on the last. Which results in equation 67.

$$y = \frac{\Delta a}{x_{step}} \Delta x + y_1 \tag{67}$$

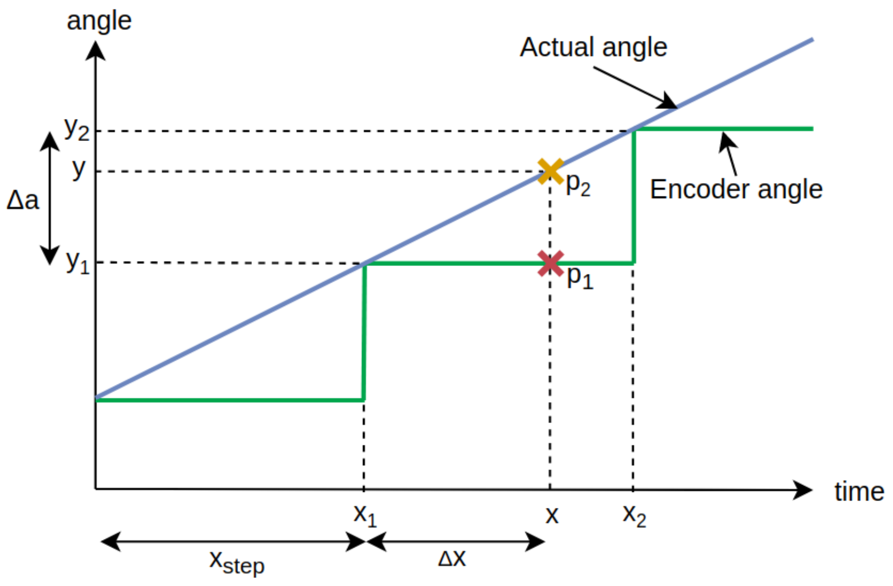


Figure 42: Ideal linear interpolation.

The x-axis is discrete time and will be kept track of in relation to the number of samples passing and therefore x will be denoted n. To know how fast the encoder values change over time this frequency is calculated with equation 68.

$$f = \frac{v_{[^o/s]}}{res_{rotor}} = 5000[RPM] \cdot 6 \cdot \frac{360^o}{2^8} = 42188Hz \tag{68}$$

The frequency is more than four times faster than the sampling frequency which means that not every encode value step will be sampled. The interpolation will only work if the is at least 2 samples on a step. To find the maximum speed where the interpolation works it is found where the encoder step frequency is lower than $10kHz$.

$$10kHz > v_{RPM} \cdot 6 \cdot res_{rotor} \tag{69a}$$

$$v_{RPM} < \frac{10kHz}{6 \cdot res_{rotor}} \tag{69b}$$

$$v_{RPM} < \frac{10kHz \cdot 2^8}{6 \cdot 360} \tag{69c}$$

$$v_{RPM} < 1185RPM \tag{69d}$$

When the rotor speed is less than $1185RPM$ the system will sample at least 1 time per encoder step.
Equation 67 can be further changed to fit the system.

$$a_i = \frac{\Delta a}{n_{step}} n_{samples} + a \tag{70}$$

Where a_i is the interpolated angle, n_{step} is the number of samples on the last step, $n_{samples}$ is the number of samples before the current sample on the current step, a is the angle received from the encoder, as can be seen in figure 43.

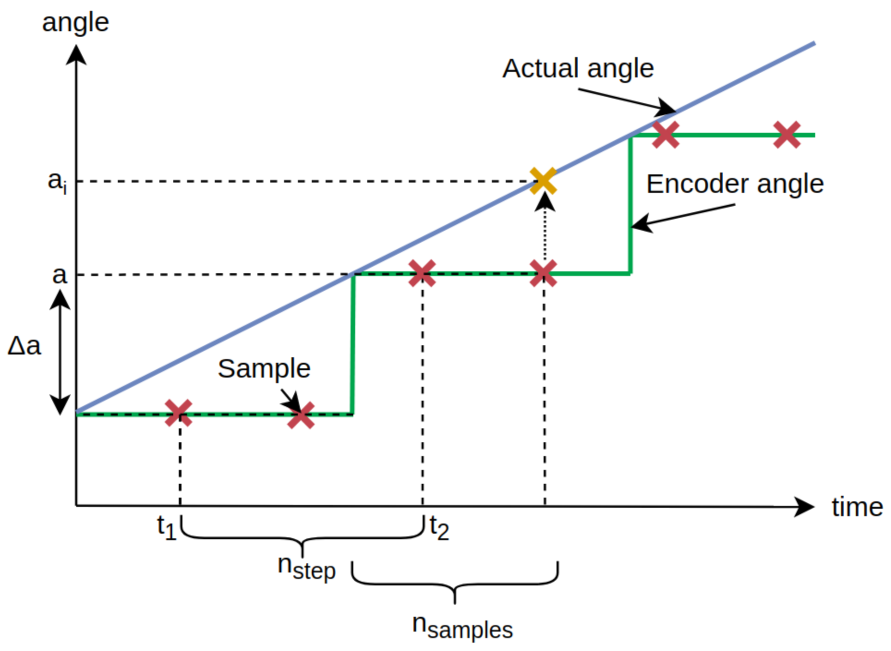


Figure 43: Practical version of linear interpolation.

Equation 70 can be rewritten to have the two counters as the fraction resulting in equation 71.

$$a_i = \frac{n_{samples}}{n_{step}} \Delta a + a \tag{71}$$

$n_{samples}$ are a count of the number of samples on the current step. The counter starts at 0 which means $n_{samples} \leq n_{steps}$ if the speed is approximately the same for the current and last encoder step.

$$0 \leq \left(\frac{n_{samples}}{n_{step}} \right) \leq 1 \tag{72}$$

Which means that the output of the interpolation is limited to

$$a \leq a_i \leq (a + \Delta a) \tag{73}$$

The finished interpolation algorithm can be seen in the code snippet 1 below.
Every time the function is called a counter, *time*, is incremented and this counter keeps track of the time.
Line 5 to 8 handle the first time the interpolation is used and it updates the variable t_1 which is the time of the left edge of the counter n_{step}.
Line 10 to 15 handles the interpolation until the right edge of n_{step} is updated, this is done with the variable t_2. For both the edges the value from the encoder is saved from the encoder to handle the step size in case one or more steps are skipped.
Line 17 to 29 handle the actual running algorithm. The sample counter is incremented. If a new step is reached the left edge is set the the last right edge, $t\_1 = t\_2$, and the new right edge is updated, $t\_2 = angle$.
The step width is calculated as well as the step size which results in all the variable ready to calculate the interpolated angle on line 28 as per equation 71.

```
/* Linear Interpolation Algorithm */
f32 interpolateAngle(f32 angle){
  time++;                               // Keep track of time
  f32 angleInterpolated = angle;        // Default value of angle
  /* First time used */
  if(t1 == 0){
    t1 = time;                          // Update time of t1
    t1v = angle;                        // Update angle of t1
  }
  /* If t2 has not been updated for the first time yet */
  if(t2 == 0){
    if(angle != t1v){                   // Check if first step happens
      t2 = time;                        // If so update time for t2
      t2v = angle;                      // Update angle of t2
    }
  }
  /* For all other steps than the first two */
  if(t1 != 0 && t2 != 0){               // Do the actual interpolation
    nSamples++;                         // Keep track of samples on current step
    if(angle != t2v){                   // If new step happens
      t1 = t2;                          // Move t2 to t1
      t1v = t2v;                        // Update value of t1
      t2 = time;                        // Update t2 to current time
      t2v = angle;                      // Update value of t2 to current angle
      nSamples = 0;                     // Reset number of samples on step
    }
    u32 nStep = t2 - t1;                // Get time on last step (approximation)
    f32 angleStep = t2v-t1v;            // Get angle step
    angleInterpolated = nSamples/nStep * angleStep + angle; // Calculate new angle
  }
  return angleInterpolated;             // Return new angle
}
```

Listing 1: Interpolation algorithm implemented on the embedded system.

To measure the improvement of the interpolated angle compared to the encoder angle a sweep of the motor speed from 1RPM to 5000RPM has been performed. The sweep was done with the algorithm implemented in Matlab and the error between the interpolated angle and the actual angle calculated. The results are shown on figure 44. The interpolated error is smaller for speeds less than 1185RPM otherwise the interpolation error is the same as the general error.

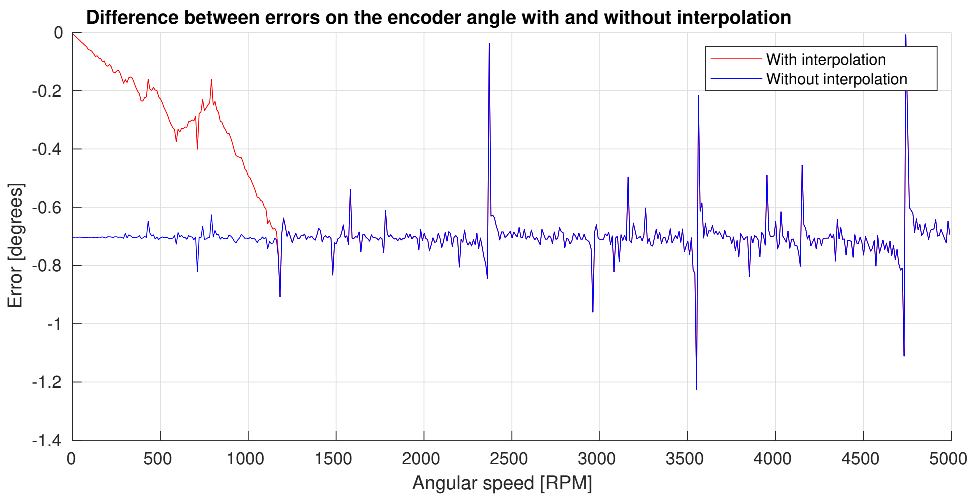


Figure 44: Frequency sweep of angular error with and without linear interpolation.

7.2.3 Analog-to-Digital Converter (ADC)

To measure the phase currents once per PWM cycle and also the torque reference from the torque pedal an ADC is used. The Zynq has two 12 bit ADC's which enables it to sample two signals simultaneously. It is important to sample the phase currents at the same time to avoid distortion by having a small time delay between current samples.
Due to the symmetry between the three phases only two of the phases needs to be measured and the last can be calculated with: $I_C = -(I_A + I_B)$. The calculation of the last phase is done in the processing system.
To control the ADC the IP block *XADC* is used which can be seen on figure 45.

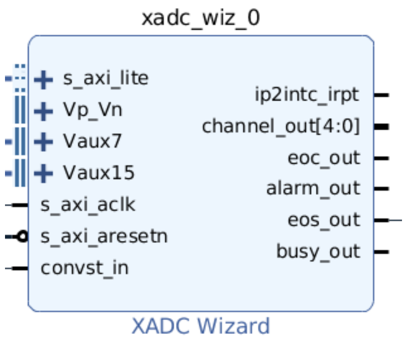


Figure 45: IP core to handle the two ADC in the Zynq.

The block is configured to be triggered by an external signal which is connected to one of the ADC pulses produced by a PWM module.
While the phase currents are measured the torque pedal position is also measured. When all signals are measured the signal end-of-sequence, *eos_out*, outputs a pulse. The *eos* signal is used to trigger the interrupt on the processing system as can be seen on figure 46.

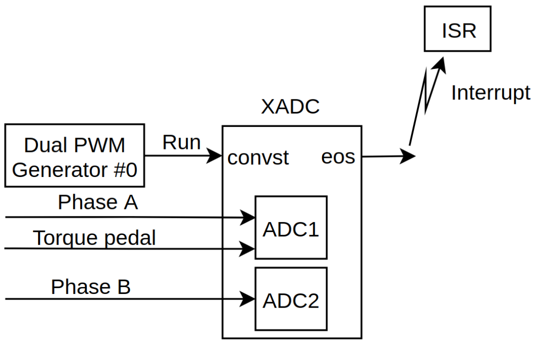


Figure 46: Function diagram showing the signals in and out of the ADC module.

Subconclusions

A dual PWM module is developed that can produce two individual and inverted PWM signals with dead time and ADC trigger pulses. Tests show that three of these modules can be used at the same time phase shifted 120°. The encoder module IP is included and a higher level algorithm is developed to improve the readings. Lastly the ADC's are configured and linked to the data pins, trigger and an interrupt is set up.

7.3 Processing System (PS)

All the high level control and communication is placed on the processing system to get the benefit of higher abstraction on a processor. The section is structured in the same way as data is processed. First the ISR and how the PL triggers the PS, then the FOC and lastly how the result is transferred back to the PL. The section will end with the communication interface between the embedded system and a computer.

7.3.1 Interrupt Service Routine

Every time an interrupt is triggered the ISR is called which sends a run command to the control loop to run. The interrupts are triggered with the same frequency as the PWM is running with, which is $10kHz$. Normal running behavior could look like the scenario shown on figure 47.

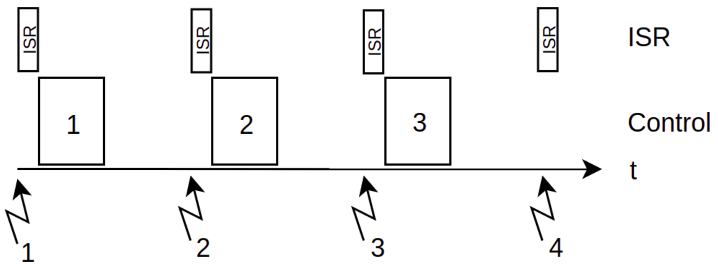


Figure 47: Normal running behavior of the interrupts coming in triggering the ISR which then triggers the control loop task.

Every time an interrupt is received the control loop is run and there is always some time between the control is run where the processor is idle.
The unlikely case where the control takes more time than the time between two interrupts can be seen in figure 48. For robustness the system should be able to handle this situation.
If the run command used by the ISR to trigger the control is a simple boolean variable set high by the ISR and reset as the first step in the control loop it is possible to get an interrupt while executing control. The running control will then finish and the new request will directly start executing.

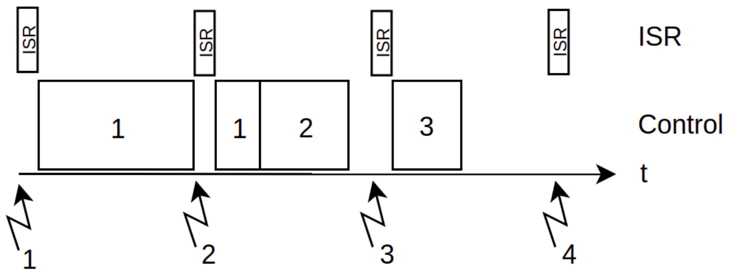


Figure 48: Unlikely scenario where a control task takes longer than the time between two interrupts.

Problems arise from this solution if the very unlikely event happen where multiple interrupts happen during the same control loop cycle as can be seen on figure 49.

Every interrupt before the last will be overlooked. In some systems overseeing an interrupt can be catastrophic but in this case it will improve the performance of the system compared to the alternative of executing all control tasks as can be seen on figure 50.

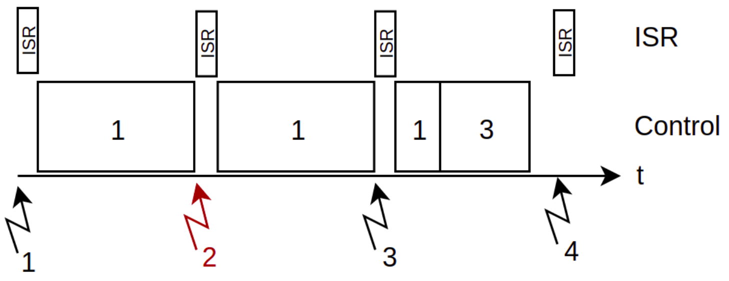


Figure 49: The scenario of a control task running for much longer than expected and skipping interrupts.

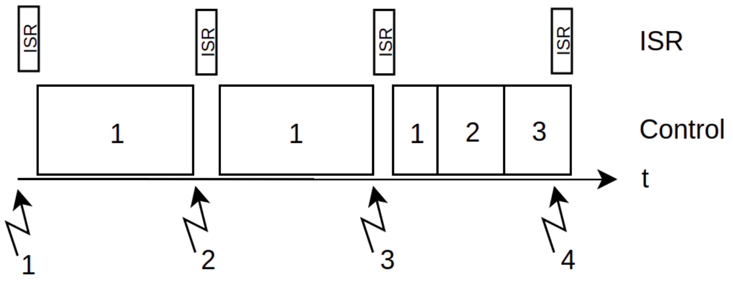


Figure 50: The scenerio of a control task running for much longer than expected but no interrupts are overseen.

Executing every control task will result in the system handling old data which is not relevant anymore.
Therefore the run command used by the ISR to trigger the control loop is a simple boolean variable.

7.3.2 Clarke Transformation and Park Transformation

The control type chosen for the system is vector control also called field-oriented control and this requires Clarke/Park transformations as well as their inverse counterparts. The transformations are implemented in the processing system and to limit the amount of calculations needed for each transformation all constants are defined beforehand.

Clarke Transform

The embedded implementation of the Clark transformation as per equation 26 can be seen in code sample 2. The function creates two results and therefore instead of returning the values, the variables are declared outside the function calls and a pointer to the variables are parsed into the function.

```c
/* The Clarke function */
void clarke(f32 *iAlpha, f32 *iBeta, f32 iA, f32 iB, f32 iC){
  *iAlpha = TWO_THIRDS * iA - ONE_THIRD * iB - ONE_THIRD * iC;
  *iBeta  = ONE_OVER_SQRT_THREE * iB - ONE_OVER_SQRT_THREE * iC;
}
```

Listing 2: Embedded Clarke Transformation.

Park Transform

The embedded implementation of the Park transformation as per equation 28 can be seen in code sample 3. The transformation involves calculating the sine and cosine of the angle. These two functions are implemented by using look-up tables and are discussed in section 7.3.2.1.

```c
/* The Park function */
void park(f32 *iD, f32 *iQ, f32 iAlpha, f32 iBeta, f32 angle){
  const f32 cosAngle = fastCos(angle);
  const f32 sinAngle = fastSin(angle);
  *iD = cosAngle * iAlpha + sinAngle * iBeta;
  *iQ = -sinAngle * iAlpha + cosAngle * iBeta;
}
```

Listing 3: Embedded Park Transformation.

The inverse Clarke and inverse Park transformations can be found in appendix A.1 and A.1.

7.3.2.1 Sine and Cosine

The trigonometric functions sine and cosine are implemented with a look-up table to improve performance. The sine function looks up a given angle in the table to find the output of the function. The cosine uses the sine function but offset the angle by 90°.
To limit the amount of memory used only one value is saved per degree which results in 360 saved values of type *f32* which is based on the type *float*. The variables are 4 bytes resulting in $1440 bytes \sim 1.5 kB$ used. The maximum error that can exist because of the quantization is the biggest step between two values in the data set. The largest error is where the sine/cosine has the largest slope, and this is where it crosses 0 on the y-axis. The error is positive when the slope is positive and negative when the slope is negative. The maximum error is $\pm 1.75\%$ as can be seen on figure 51.

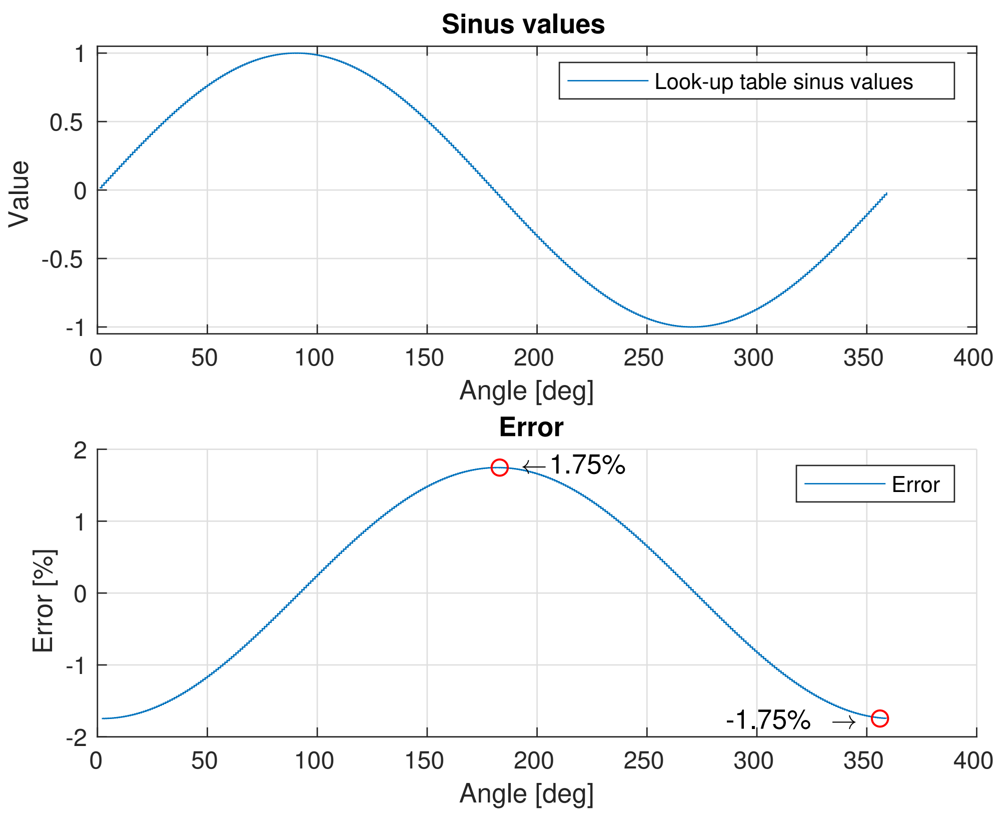


Figure 51: Error between the real sine value and the look-up table value.

The code to perform the sine and cosine can be found in appendix A.3.2.

7.3.3 Controller

The PI-controller is implemented in *c* on the processing system in a object orientated way where each of the two controllers are defined together with their gains. When the system is booted up the controllers are initialized with the values and prepared for usage. The code for this can be seen in code sample 4.

```c
/* Declare controllers */
Controller cQ, cD;
static f32 kpQ = 1, kiQ = 1, kpD = 1, kiD = 1;

void initControllers(){
    /* Initialize controllers with gains */
    initController(&cQ, kpQ, kiQ);
    initController(&cD, kpD, kiD);
}
```

Listing 4: Initialization of PI-controller.

When the motor is running the only thing needed to be done is to input the reference value and the current measured value to the controller and get a new output as can be seen in code sample 5. The function 'getOutput()' handles the controller gains, keeping track of the integrator part and integrator windup.

```c
    /* Run controller */
  f32 outD = getOutput(&cD, 0, iD);
  f32 outQ = getOutput(&cQ, getTargetTorque(), iQ);
```

Listing 5: Usage of PI-controller.

The implementation of the 'getOutput()' function can be seen in code sample 6. First the controller gains *kp* and *ki* are retrieved. Then the controller's integrator part is updated and then retrieved.
The output of the controller is found with the following equation.

$$u = k_p \cdot e + k_i \cdot int \tag{74}$$

Where u is the output, e is the error which is the target minus the input, $e = target - input$, int is the integrator part and k_p and k_i are the controller gains. The integral part of the controller might experience integrator windup because the control execution is much faster than the motor acceleration. To handle this a simple output limitation is implemented to limit the controller output.

```
/* Function to get the next output of a controller with a new input */
f32 getOutput(Controller *c, f32 target, f32 input){
  f32 error = target - input;                         // Calculate error

  // Collect controller data
  const f32 kp  = getKp(c);
  const f32 ki  = getKi(c);
  addToIntegrator(c, error);
  const f32 integrator = getIntegrator(c);

  f32 output =  (kp * error) + (ki * integrator);     // Calculate new output

  // Anti integrator windup. Limit the output to within set limits
  if(output > MAX_OUTPUT){                            // Check upper limit
    output = MAX_OUTPUT;                              // Limit output
  }
  if(output < MIN_OUTPUT){                            // Check lower limit
    output = MIN_OUTPUT;                              // Limit output
  }
  return output;                                      // Return new output
}
```

Listing 6: Implementation of PI-controller 'get output'-function.

7.3.4 Communication between PL and PS

Sending thresholds from the processing system to the PWM low-level modules are done through a piece of block RAM. The build-in block RAM of the Zybo consist of a 36kbit storage area with two independent access ports.
The PWM modules expect 8 bit thresholds which results in 24 bits or 3 bytes being used of the block RAM.
A mere 3 bytes account for around 0.08% of the total block RAM. While this is a waste of the block RAM, it was chosen because of it being easily accessible from both the PS and PL and easy to set up.
When the control task on the processing system has run and produced the thresholds for each phase they are written into the first 3 bytes of the block RAM as can be seen on figure 52.

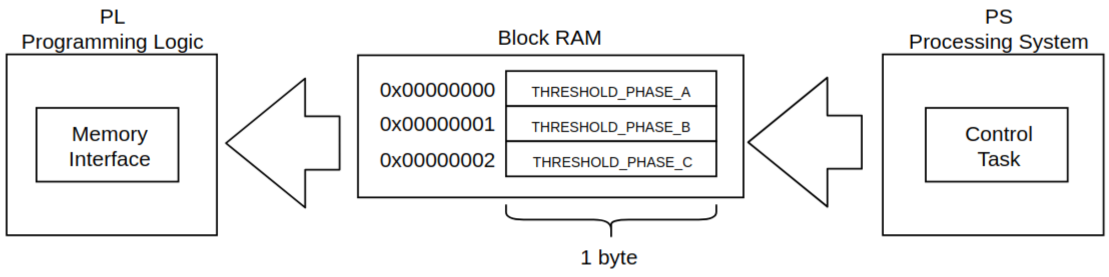


Figure 52: Communication from PS to PL through block RAM.

From here the memory interface module in the logic can collect the thresholds and parse them unto the three PWM modules as can be seen on figure 53. The PWM modules then use the thresholds to produce the PWM signals as discussed in section 7.2.1.

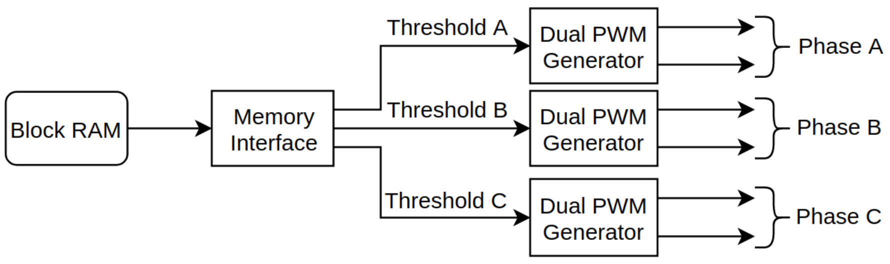


Figure 53: The data flow in the PL from the block RAM to the PWM generators.

7.3.5 Computer interface

An interface between the embedded controller and a computer is implemented to monitor the system, start the inverter and change the gains for the controllers. The interface is made in a simple low-level way by sending ASCII strings via UART to the embedded controller. The strings have a predefined structure and limited error handling. The string structure can be seen in table 3.

# character	1	2	3	4	5	6
Function	ID	Read / Write	Value			
Valid Values	$0 \rightarrow 9$	'r'/'w'	$0 \rightarrow 9999$			

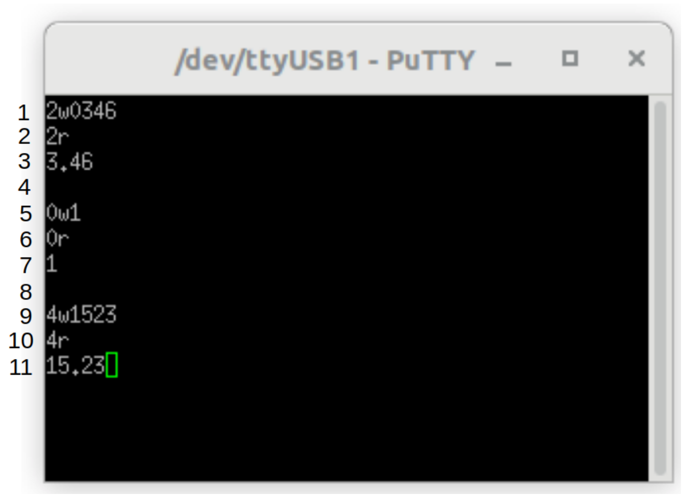
Table 3: Structure of the interface messages from the computer to the embedded controller.

The first character of the message contains an ID of the variable that the message is referring to. The ID's of all variables included can be seen in table 6.
The second character describes if the message is a read or write command.
The third till sixth character are only used for the write command. They contain the value to which the variable should be changed to. The format of the value can be seen in table 6. To handle digits especially for the controller gains the values from $0000 \rightarrow 9999$ are mapped to $00.00 \rightarrow 99.99$ before being added to the controller.

ID	Valid values	Valid commands	Variable	Unit
0	0 = Stop 1 = Run	'r'/'w'	Inverter state	-
1	-	'r'	Speed	Hz
2	$0 \rightarrow 9999$	'r'/'w'	Ki_D	1/100 *Example: 1234 = 12.34*
3	$0 \rightarrow 9999$	'r'/'w'	Kp_D	1/100 *Example: 1234 = 12.34*
4	$0 \rightarrow 9999$	'r'/'w'	Ki_Q	1/100 *Example: 1234 = 12.34*
5	$0 \rightarrow 9999$	'r'/'w'	Kp_Q	1/100 *Example: 1234 = 12.34*

Table 4: List of variables that can be accessed through the communication between a computer and the embedded controller.

An example of how the communication can be seen on figure 54.
Line 1 is a command to write the value 03.46 to the variable with ID=2 which is *Ki_D*.
Line 2 is a read command of the same variable.
Line 3 is the reply from the embedded system.
Line 5 is a write command to write the value *1* to the variable with ID=0 which translates to sending a start command to the inverter. Line 6 and 7 are the same as 2 and 3 but for the inverter state.
Line 9 is a command to write the value 15.23 to the gain *Ki_Q*.



```
 1 2w0346
 2 2r
 3 3.46
 4
 5 0w1
 6 0r
 7 1
 8
 9 4w1523
10 4r
11 15.23
```

Figure 54: Example of the computer interface used for communication between the embedded controller and a computer.

Subconclusion

A short ISR is developed to simply trigger the actual control loop in the processor. When the control is triggered it runs Clarke and Park Transformations, angle interpolation, and two PI controllers. The controller output is then converted back into PWM thresholds with an Inverse Park and Inverse Clarke transformation and the result is put into block RAM for the PL to use.

8 Discussion

The section will discuss the few test result gathered throughout the project but mostly go into detail with areas where problems might come up. A crucial factor of the project was that the inverter never was build due to time constraints and therefore it was never tested. Which results in some solutions only being backed up by calculations and simulations and not physical tests.
First the discussion will go through the possible problems with the hardware. Then the control and thermal characteristics are discussed and lastly the embedded system.

8.1 Power board

8.1.1 Construction

Constructing the inverter in the intended way might turn out difficult. While mounting the bus bars across all three phase PCB's and mounting the capacitors directly on top of the bus bar might be a good idea electrically, it can turn out to be near impossible in practice.
The bus bars are made of solid copper which have a high thermal conductivity and therefore soldering anything directly to the bars are difficult. Heating up all the PCB's and the bus bar in an oven or heating plate is the only option. Getting all phase PCB's to align and solder correctly to the bus bars requires high precision and it might not be feasibly to use the same PCB's for a round two if something goes wrong. On top of mounting the bus bars to the PCB's the capacitors has to be mounted at the same time.
A compromise with a bit worse electrical characteristics but better construction might need to be used.
If the budget would allow, a larger heat sink would be a better choice. Being able to fix the output cables to the heat sink could help with improving the mechanical stability of the assembly and prevent breaking of the output screw terminals.

8.1.2 Capacitors

Finding out if the capacitance built in is enough to smooth out everything is still left to be tested.

8.2 Driver board

A mistake has caused only one of the split gate resistors being placed in the schematic. A resistor closer to the gate should also have been placed. Using an SMD diode could also lead to a better layout, but utilizing components from the stock was chosen to keep down the cost.

8.3 Control

A fully working model of the system was not achieved. In some cases the model works as expected one of these being when the load on the motor increases resulting

in an increase in amplitude of the three phase currents. As can be seen on figure 26 in section 6.5.
When the load of the motor increases the torque of the motor drops, and then stabilizes at zero again after around $40ms$ to $50ms$. When the load changes the torque momentarily drops the same amount and then returns back to zero. As can be seen on figure 29 in section 6.5.
On figure 28 in section 6.5 the speed of the motor can be seen. It can be seen that the speed is negative which is caused by the torque being divided by the inertia, before it is integrated into the speed. This results in the speed dropping when the load of the motor is changed. Combining that with the speed not increasing fast enough between the acceleration drops. The speed should not be negative when the torque produced by the electrical field is more than the load of the motor.
Figure 27 in section 6.5 shows the d- and q-currents. There it can be seen that the q-current is not going towards the set reference. It can also be seen that the d-current is going towards zero but slowly. This suggest that the PI-controller is not set correctly, or there is an error in the model.

8.4 Embedded software

8.4.1 PWM module

As can be seen in the end of section 7.2.1 where the PWM modules are tested. The module work and it can produce a sine curve at the selected frequency. Three module instances can be used phase shifted 120° resulting in a three-phase signal.

8.4.2 Minimizing the deadtime in the PWM module

The deadtime between the high-side and low-side PWM's are ~ 53 times longer than the turn off time of the transistors used. The deadtime is mainly limited by the resolution of the internal counter in the PWM module. By increasing the counter limits to 1000 instead of 100 the resolution would become 10 times higher. The deadtime resolution would also become 10 times higher which would bring the minimum deadtime down to $400ns$ only ~ 5.3 times larger than necessary.

8.4.3 Linear interpolation

The linear interpolation use a couple of assumptions and it is difficult to quantify the expected benefit from having it. As shown in section 7.2.2.1 a more precise rotor angle can be achieved for speeds less than $1185RPM$ for moments with low to no acceleration or deceleration. The error caused by acceleration and deceleration is not analyzed, only a static simulation has been made. To precisely determine how big of an advantage or disadvantage the interpolation can have in a real scenario, the algorithm would need to be dynamically tested.

8.4.4 Waste of block RAM

Section 7.3.4 describes how the communication of PWM thresholds are send from the processing system to the logic through a piece of block RAM. Only 0.08% of the total block RAM is actually in use which is a terrible waste. The interface between the two systems should have been done using the AXI interface instead.

9 Conclusion

The three-phase inverter was designed with the requirements of an electric go-kart in mind.
As price was the primary concern the smallest possible PCB area was desirable which brought other trade-offs. To minimize the size of the power and driver boards and in turn improve the overall modularity, the PCB's were made to only handle one phase and three boards were then used to handle the three-phases. For power board an aluminum PCB was used to have a high heat conductivity which allows for the use of SMD MOSFET's as the high power transistors. The driver board was build to be mounted directly on top of the power board to minimize driver signal path lengths and parasitic inductances.
The motor is controlled on the Zynq with a PI controller in a field orientated way. Each phase is controlled with a dual PWM module with deadtime between the two PWM signals implemented in the FPGA. The angle out of the encoder has a low resolution so a more accurate angle is approximated at low speeds with the use of a linear interpolation algorithm.
Since the inverter was not built it is not possible to back up the theoretical work.
It was not possible to make a fully working model of a controlled motor. A tuned discrete controller was therefore not implemented in the project.

10 Symbol list

Symbol	Meaning		Symbol	Meaning
Δa	Angle difference		a	Current Rotor Angle
a_i	Interpolated Rotor Angle		C_{GD}	Gate-Drain Capacitance
c_{steps}	PWM counter steps		dt_{res}	Dead Time Resolution
dt_{min}	Dead Time Minimum		$encoder_{freq}$	Encoder Frequency
e	Error		f_s	Switching Frequency
f_{pl}	FPGA Clock Frequency		G_{sw}	Switching Charge
I	Current		I_D	Drain Current/
$I_{GSource}$	Sourced Gate Current		I_{GSink}	Sinked Gate Current/
$I_{GSink/MOSFET}$	Sinked Gate Current per MOSFET		$I_{GSource/MOSFET}$	Sourced Gate Current per MOSFET
$I_{Gsource_{t}otal}$	Total switch ON current		$I_{Gsink_{t}otal}$	Total switch OFF current
i_a, i_b, i_c	A, B and C currents		i_α, i_β	Alpha and Beta currents
i_d, i_q	D and Q currents		k_i	Integrator Gain
k_p	Proportional Gain		L_d	Inductance in D direction
L_q	Inductance in Q direction		λ_m	Self inductance
N	Number of Transistors		φ	Angle
P	Pole pairs		P_{con}	Conduction Power Loss for all MOSFET's
$P_{con/MOSFET}$	Conduction Power Loss Per MOSFET		$P_{con/MOSFET}$	Conduction Power Loss Per MOSFET
P_{sw}	Switching Loss for all MOS-FET's		$P_{sw/MOSFET}$	Switch Loss per MOSFET
Q_G	Gate Charge		Q_{GD}	Gate-Drain Charge
Q_{GS}	Gate-Source Charge		n_{step}	Sample count on encoder step
$n_{samples}$	Sample count on current step		r	Read
R	Resistor (Generic)		$R\#$	Resistor Number #
R_{pot}	Potmeter Resistance		R_{DS-ON}	Drain-Source ON Resistance
$R_{G_{c}ommon}$	Common Gate Resistance		$R_{G_{FET_i}nt}$	Internal Gate-Source Resistance of a MOSFET
R_{pullup}	Pull-up Resistor		R_s	Stator Resistance
res_{rotor}	Encoder resolution on the rotor angle		$res_{electric}$	Encoder resolution on the electric field angle
sin_{freq}	Sine Frequency		t_1, t_2	Specific Times
t_{angle}	Time per angle		t_{on}	Turn on time
t_{off}	Turn off time		$th_{dutycycle}$	Duty Cycle Threshold
th_{high}	High-side Threshold		th_{low}	Low-side Threshold
$t_{deadtime}$	Dead time		u	Output
V	Voltage		V_{batt}	Battery Voltage
V_{DC}	Supply Voltage		V_{ref}	Reference Voltage
V_{Miller}	Miller-plateau of a MOSFET		v_{rpm}	Rotational speed in RPM
w	Write		Δx	Difference in x
x_{step}	Encoder Step Width			

Table 5: Symbol list

11 Abbreviation list

Abbreviation	Meaning
AC	Alternating Current
ADC	Analog to Digital Conversion
AXI	Advanced eXtensible Interface
BLDC	Brush Less DC Motor
CMRR	Common-Mode Rejection Ratio
CSI	Common Source Inductance
DC	Direct Current
EOS	End Of Sequence
FET	Field-Effect Transistor
FOC	field oriented control
FPGA	Field-Programmable Gate Array
IP	Intellectual Property (Block)
ISR	Interrupt Service Routine
MLCC	Multi-layer ceramic capacitor
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
PCB	Printed Circuit Board
PI	Proportional Integral (Controller)
PL	Programmable Logic
PMAC	Permanent Magnet AC Motor
PMSM	Permanent Magnet Synchronous Motor
PS	Processing System
PWM	Pulse-Width Modulation
RAM	Random Access Memory
SDU	University of Southern Denmark
SiC	Silicon Carbide
SMD	Surface Mount Device
UART	Universal Asynchronous Receiver/Transmitter
VHDL	Very high speed integrated circuit Hardware Description Language

Table 6: Abbreviation list

List of Figures

12 References

All links to websites was working on the day of the hand-in *28-05-2019*.

References

[1] Project 1. semester - S19 - Electronics - ver1.pdf,
Karsten Holm Andersen, (2019)

[2] http://www.motenergy.com/me1117.html Parameters taken from the production desciption Motenergy ©2011

[3] Direct Torque Control of a Permanent Magnet synchronous Motor
David Ocen, Master's Degree ProjectStockholm, Sweden 2005.
https://www.diva-portal.org/smash/get/diva2:582454/FULLTEXT01.pdf

[4] Texas Instruments TIDA-00364 reference design,
Pawan Nayak, N. Navaneeth Kumar,
http://www.ti.com/tool/TIDA-00364

[5] Infineon IPB017N10N5 MOSFET datasheet,
https://www.infineon.com/dgdl/Infineon-IPB017N10N5-DS-v02_01-EN.pdf?fileId=5546d4624a75e5f1014ac4a981111eed

[6] ON Semiconductor 1N5821G Schottky diode datasheet,
https://docs-emea.rs-online.com/webdocs/1573/0900766b81573229.pdf

[7] Silicon Labs Si88261BAD gate driver datasheet,
https://docs-emea.rs-online.com/webdocs/1299/0900766b81299254.pdf

[8] Cosel MGS1R5 isolated power supply datasheet,
https://docs-emea.rs-online.com/webdocs/1690/0900766b81690263.pdf

[9] Linear interpolation,
https://www.encyclopediaofmath.org/index.php/Linear_interpolation

[10] AM256 – angular magnetic encoder IC,
Data-sheet-AM256-magnetic-encoder-IC.pdf

[11] The Engineering toolbox,
https://www.engineeringtoolbox.com/thermal-conductivity-metals-d_858.html

[12] UVZ2A471MHD electrolitic capacitor datasheet,
https://docs-emea.rs-online.com/webdocs/13c7/0900766b813c774f.pdf

13 Appendix

A Code Samples

The appendix contains code samples from various important and interesting sections of the program code implemented on the Xilinx Zynq controller.

A.1 Clarke / Park Transformations

Clarke Transformation

The embedded implementation of the Clarke Transformation.

```c
/* The Clarke function */
void clarke(f32 *iAlpha, f32 *iBeta, f32 iA, f32 iB, f32 iC){
  *iAlpha = TWO_THIRDS * iA - ONE_THIRD * iB - ONE_THIRD * iC;
  *iBeta  = ONE_OVER_SQRT_THREE * iB - ONE_OVER_SQRT_THREE * iC;
}
```

Listing 7: Embedded Clarke Transformation.

Park Transformation

The embedded implementation of the Park Transformation.

```c
/* The Park function */
void park(f32 *iD, f32 *iQ, f32 iAlpha, f32 iBeta, f32 angle){
  const f32 cosAngle = fastCos(angle);
  const f32 sinAngle = fastSin(angle);
  *iD = cosAngle * iAlpha + sinAngle * iBeta;
  *iQ = -sinAngle * iAlpha + cosAngle * iBeta;
}
```

Listing 8: Embedded Park Transformation.

Inverse Park Transformation

The embedded implementation of the Inverse Park Transformation.

```c
/* The inverse Park function */
void invPark(f32 *iAlpha, f32 *iBeta, f32 iD, f32 iQ, f32 angle){
  const f32 cosAngle = fastCos(angle);
  const f32 sinAngle = fastSin(angle);
  *iAlpha = cosAngle * iD - sinAngle * iQ;
  *iBeta  = sinAngle * iD + cosAngle * iQ;
}
```

Listing 9: Embedded Inverse Park Transformation.

Inverse Clarke Transformation

The embedded implementation of the Inverse Clarke Transformation.

```c
/* The inverse Clarke function */
void invClarke(f32 *iA, f32 *iB, f32 *iC, f32 iAlpha, f32 iBeta){
  *iA = iAlpha;
  *iB = -HALF * iAlpha + SQRT_THREE_OVER_TWO * iBeta;
  *iC = -HALF * iAlpha - SQRT_THREE_OVER_TWO * iBeta;
}
```

Listing 10: Embedded Inverse Clarke Transformation.

A.2 Encoder

A.2.1 Read rotor angle

The code in to read out a rotor position and convert it to degrees.

```
/* Read the rotor angle */
f32 getRotorAngle(){
    u8 position = RM28MD_POSITION & 0x000000FF;     // Only the first byte is valid
    f32 angle   = 359/255 * position;               // Map position (0->255) to
                                                    // angle (0->359)
    return angle;                                   // Return angle
}
```

Listing 11: Function to read an angle from the encoder. The angle is returned in degrees. label

A.2.2 Linear Interpolation

The linear interpolation algorithm as discussed in section 7.2.2.1. The algorithm takes in low resolution encoder angles and depending on the time between encoder angle changes the readings are used to approximate a more accurate angle.

```
/* Linear Interpolation Algorithm */
f32 interpolateAngle(f32 angle){
  time++;                               // Keep track of time
  f32 angleInterpolated = angle;        // Default value of angle
  /* First time used */
  if(t1 == 0){
    t1 = time;                          // Update time of t1
    t1v = angle;                        // Update angle of t1
  }
  /* If t2 has not been updated for the first time yet */
  if(t2 == 0){
    if(angle != t1v){                   // Check if first step happens
      t2 = time;                        // If so update time for t2
      t2v = angle;                      // Update angle of t2
    }
  }
  /* For all other steps than the first two */
  if(t1 != 0 && t2 != 0){               // Do the actual interpolation
    nSamples++;                         // Keep track of samples on current step
    if(angle != t2v){                   // If new step happens
      t1 = t2;                          // Move t2 to t1
      t1v = t2v;                        // Update value of t1
      t2 = time;                        // Update t2 to current time
      t2v = angle;                      // Update value of t2 to current angle
      nSamples = 0;                     // Reset number of samples on step
    }
    u32 nStep = t2 - t1;                // Get time on last step (approximation)
    f32 angleStep = t2v-t1v;            // Get angle step
    angleInterpolated = nSamples/nStep * angleStep + angle; // Calculate new angle
  }
  return angleInterpolated;             // Return new angle
}
```

Listing 12: Implemetation of angle intepolation algorithm.

A.3 Control

A.3.1 PI Controller

The code for determining the next output value of the PI controller. The function takes in a controller, a target value and an input. It then calculates the controller output from its gains and integrator. The function includes integrator windup limitation. The section discussing the controller can be found in section 7.3.3.

```c
/* Function to get the next output of a controller with a new input */
f32 getOutput(Controller *c, f32 target, f32 input){
  f32 error = target - input;                       // Calculate error

  // Collect controller data
  const f32 kp  = getKp(c);
  const f32 ki  = getKi(c);
  addToIntegrator(c, error);
  const f32 integrator = getIntegrator(c);

  f32 output =  (kp * error) + (ki * integrator);    // Calculate new output

  // Anti integrator windup. Limit the output to within set limits
  if(output > MAX_OUTPUT){                          // Check upper limit
    output = MAX_OUTPUT;                            // Limit output
  }
  if(output < MIN_OUTPUT){                          // Check lower limit
    output = MIN_OUTPUT;                            // Limit output
  }
  return output;                                    // Return new output
}
```

Listing 13: Implementation of the embedded PI controller.

A.3.2 Sine and Cosine

The implemented version of sine and cosine. The two functions are implemented with the help of a look-up table. The section discussing these can be found in section 7.3.2.1.

```c
/* Sine lookup table */
#define SIN_N     360  // Number of data points in the sine constant array
const f32 sineValues[] = {0,0.0174524064372835,0.0348994967025010,...};

/* Function that returns sin(angle) */
f32 fastSin(f32 angle){
  unsigned int index = (unsigned int)angle;       // Truncate decimals
  return (f32)sineValues[index % SIN_N];      //
}

/* Function that returns cos(angle) */
f32 fastCos(f32 angle){
  f32 newAngle = (f32)((int)(angle + 90) % (int)360);
  return (f32)fastSin(newAngle);
}
```

Listing 14: Sine and cosine implemented with look-up tables.

B Component list

The section includes bills of material for each of the designed PCB's as well as heat sink and bus bar for construction of the inverter.

B.1 Driver board

The components needed for the driver board.

Name	**Value**	**Unit Price**	**Description**
C1, C8, C15, C22	$1nF$	1.082 DKK	MLCC
C2, C9, C16, C23	$10nF$	1.378 DKK	MLCC
C3, C10, C17, C24	$100nF$	0.689 DKK	MLCC
C4, C11, C18, C25	$1\mu F$	2.169 DKK	MLCC
C12, C26, C13, C27	$10\mu F$	1.02 DKK	MLCC
D1, D2	-	3.3 DKK	1N5821G Schottky diode
J1, J2, J7, J8	-	3.34 DKK	Pin header, double row, 02x02
J3, J4, J5, J6	-	3.34 DKK	Pin header, single row, 01x02
R1, R2	0Ω	0.08 DKK	Resistor
U1, U3	-	107.41 DKK	MGS1R5
U2, U4	-	20.78 DKK	SI8261BAC-C-IS
PCB	-	17.40 DKK	1 piece
Total		335.9 DKK	Full price per phase
Total		1007.7 DKK	Full price for whole inverter

B.2 Power board

The components needed for the power board.

Name	**Value**	**Unit Price**	**Description**
C1,C2	$33nF$	1.25 DKK	MLCC
C3,C4,C5,C6,C7,C8	$1\mu F$	2.169 DKK	MLCC
4 unmarked capacitors on bus bar	$470\mu F$	6.5 DKK	Nichicon VZ electrolyte capacitor
J1, J2	-	3.00 DKK	Driver connector. Generic connector, single row
J7	-	20.00 DKK	Würth screw terminal
J8, J9, J10, J11	-	7.00 DKK	Samtec SSM series connector, 02x02
Q2, Q3	-	50.62 DKK	IPB017N10N5, N-Channel SiC MOS-FET
R5,R6	-	0.08 DKK	Resistor
U1, U2	-	50.62 DKK	IPB017N10N5-1
PCB	-	34.70 DKK	1 piece
		591.676 DKK	Full price per phase
		1775 DKK	Full price for whole inverter

B.3 Analog board

The components needed for the analog board.

Name	**Value**	**Unit Price**	**Description**
C1, C3, C13, C14, C16, C17	$1nF$	1.082 DKK	MLCC
C2, C15	$10nF$	1.378 DKK	MLCC
C4, C5, C6, C11, C18, C19	$10\mu F$	1.02 DKK	MLCC
C7, C8, C9, C10, C21, C22	$0.1\mu F$	0.689 DKK	MLCC
C12, C23, C26	$100nF$	0,689 DKK	MLCC
D1	-	7.47 DKK	1.235V Micropower Voltage Reference Diodes, TO-92
J301, J302, J304, J305	-	3,34 DKK	2.54mm Pitch, 10 Way, 2 Row, Straight Pin Header, Through Hole
R1, R2, R3, R12, R13, R16	10Ω	0.08 DKK	Resistor
R4, R5, R17, R18	680Ω	0.08 DKK	Resistor
R6, R19, R20	220Ω	0.08 DKK	Resistor
R7, R9	$1k\Omega$	0.08 DKK	Resistor
R8	470Ω	0.08 DKK	Resistor
R11	$3.3k\Omega$	0.08 DKK	Resistor
R14, R15	$10k\Omega$	0.08 DKK	Resistor
U1	-	5 DKK	LM324, Low-Power, Quad-Operational Amplifiers
U3, U4	-	82.16 DKK	AD620, Low Cost, Low Power, Instrumentation Amplifier
PCB	-	20.70 DKK	1 piece
		234.2 DKK	Full price per phase
		702.6 DKK	Full price for whole inverter

B.4 General

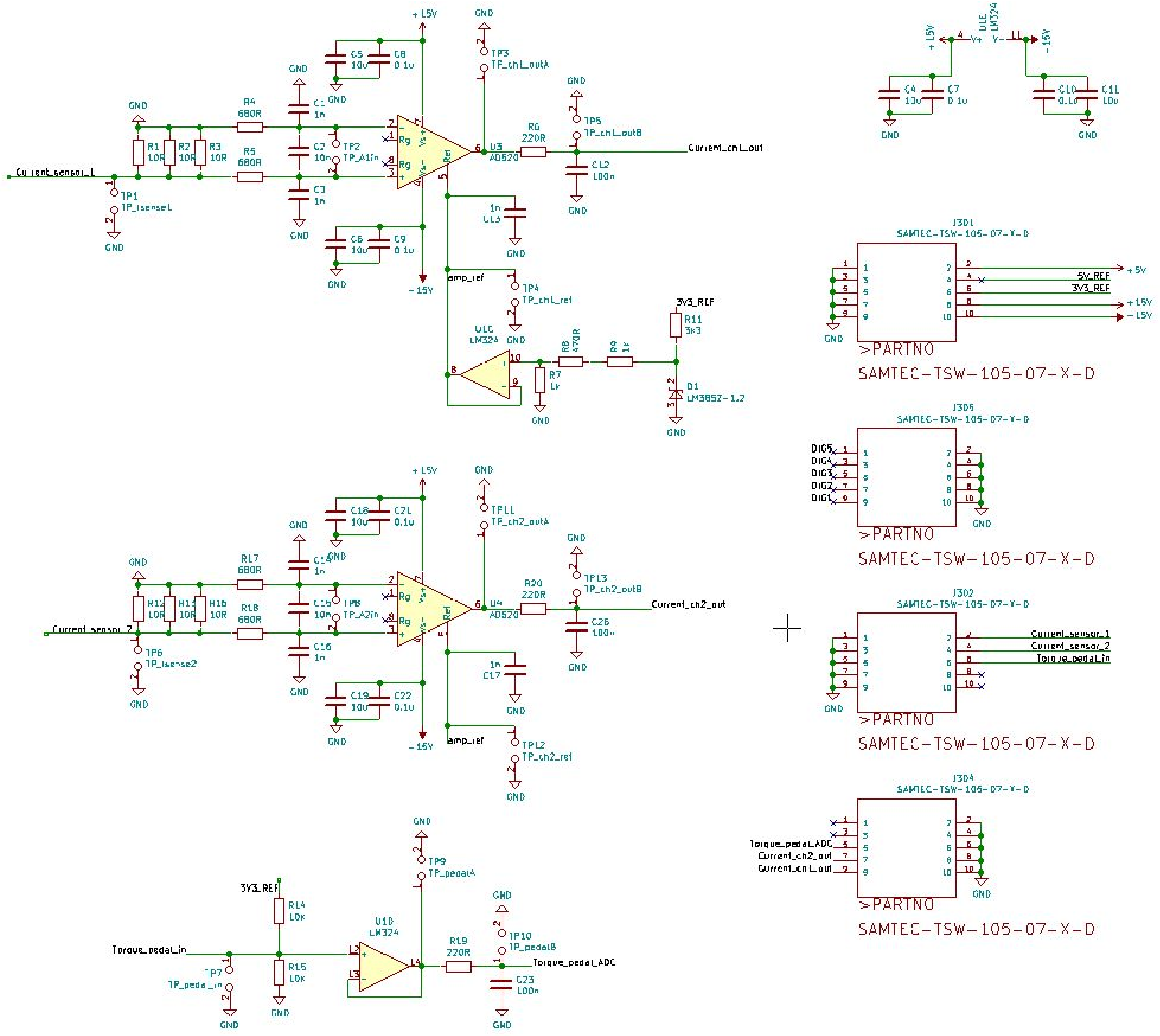
The components commonly needed for the whole assembly.

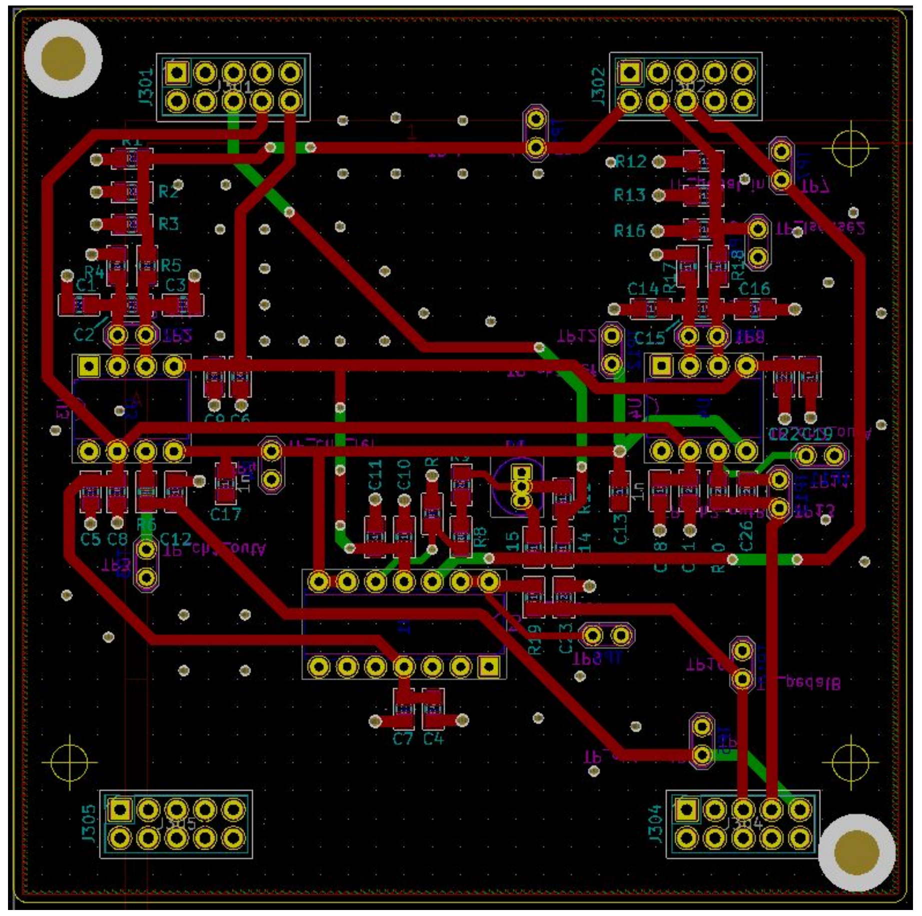
Name	**Value**	**Unit Price**	**Description**
Heat sink	$200mm \times 100mm \times 0mm$	160.00 DKK	Fischer SK 47/100 SA
Bus bar	$1m \times 15mm \times 6mm$	150.00 DKK	copper bar
Power cable	$1m$	150.00 DKK	Prysmian 6181Y
Crimp terminal	-	2.3 DKK	RS 841-7623
		466.9 DKK	Full price for inverter assembly

The full price of all components for the inverter assembly is 3485.4 DKK.

C Analog board schematic and layout

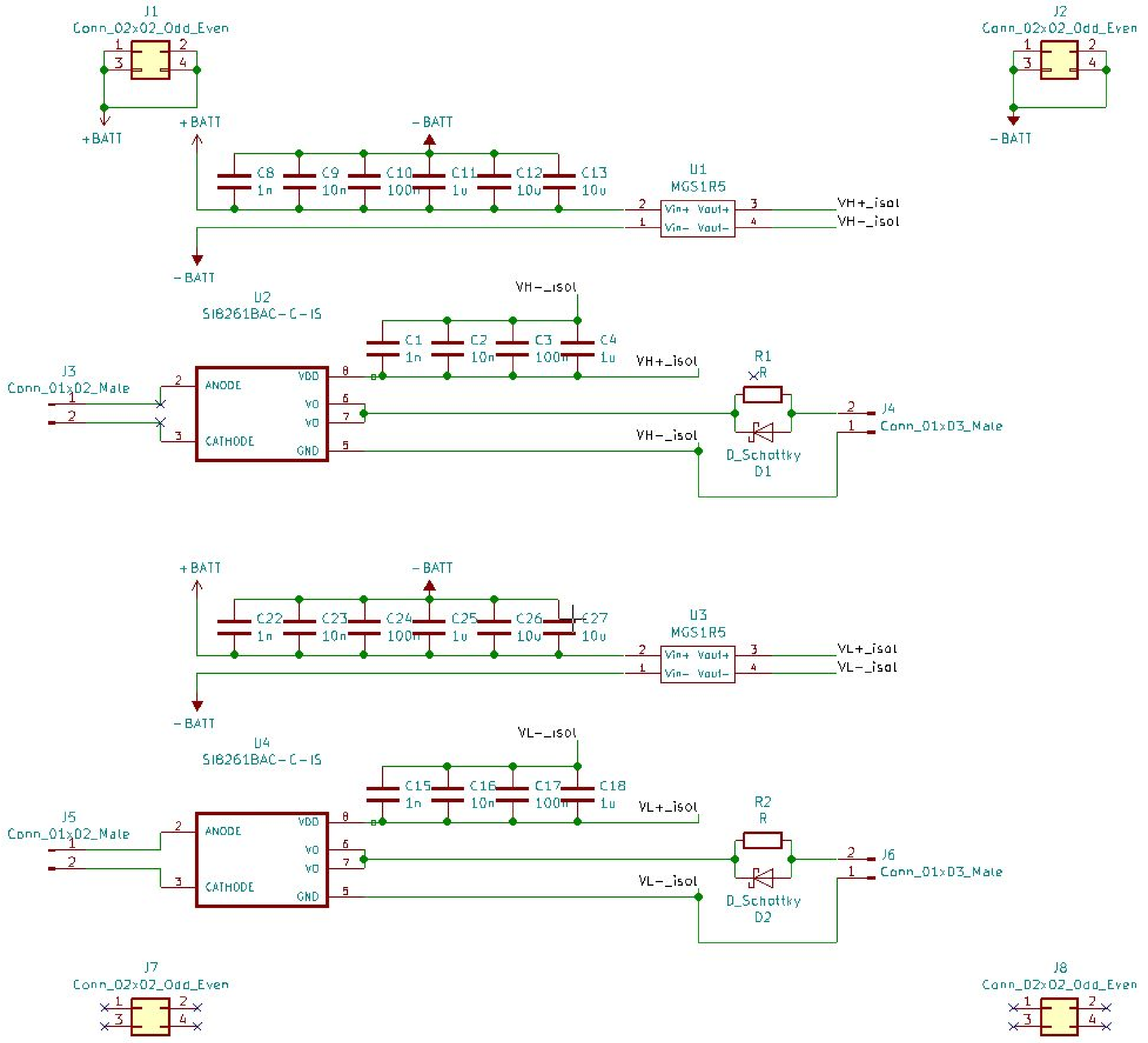
The schematics and layout of the analog board are shown in this section.

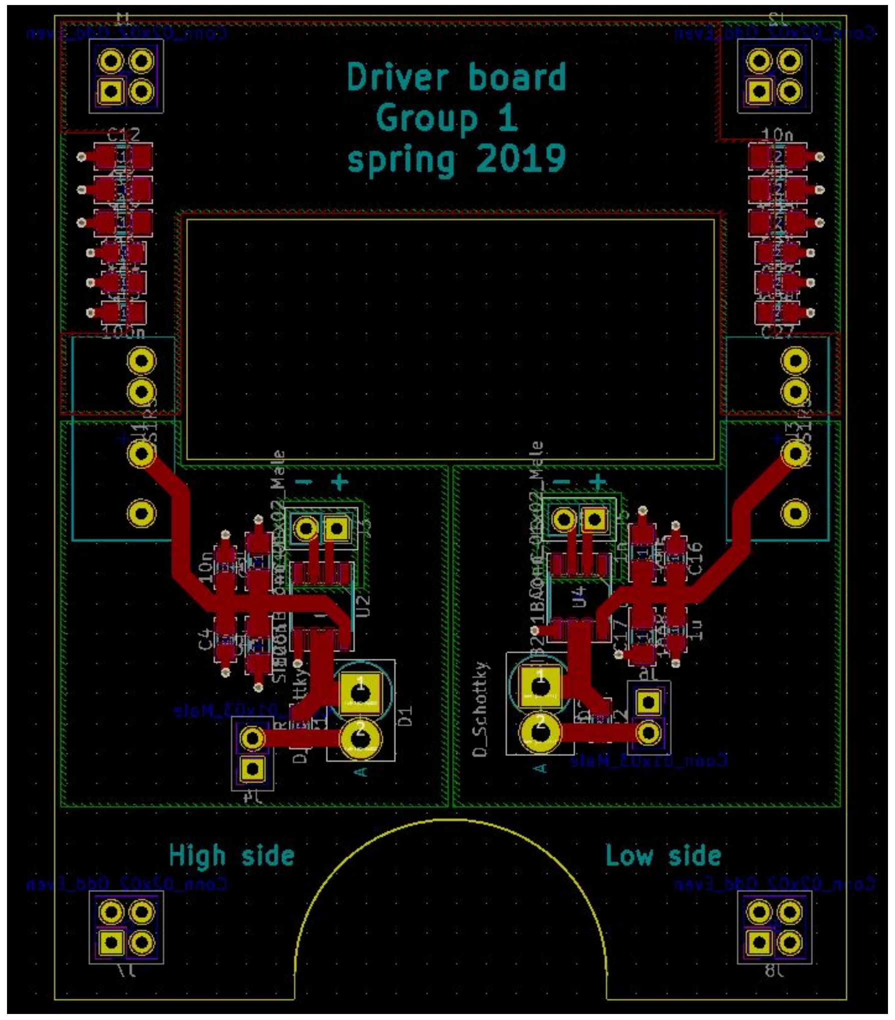




D Driver board schematic and layout

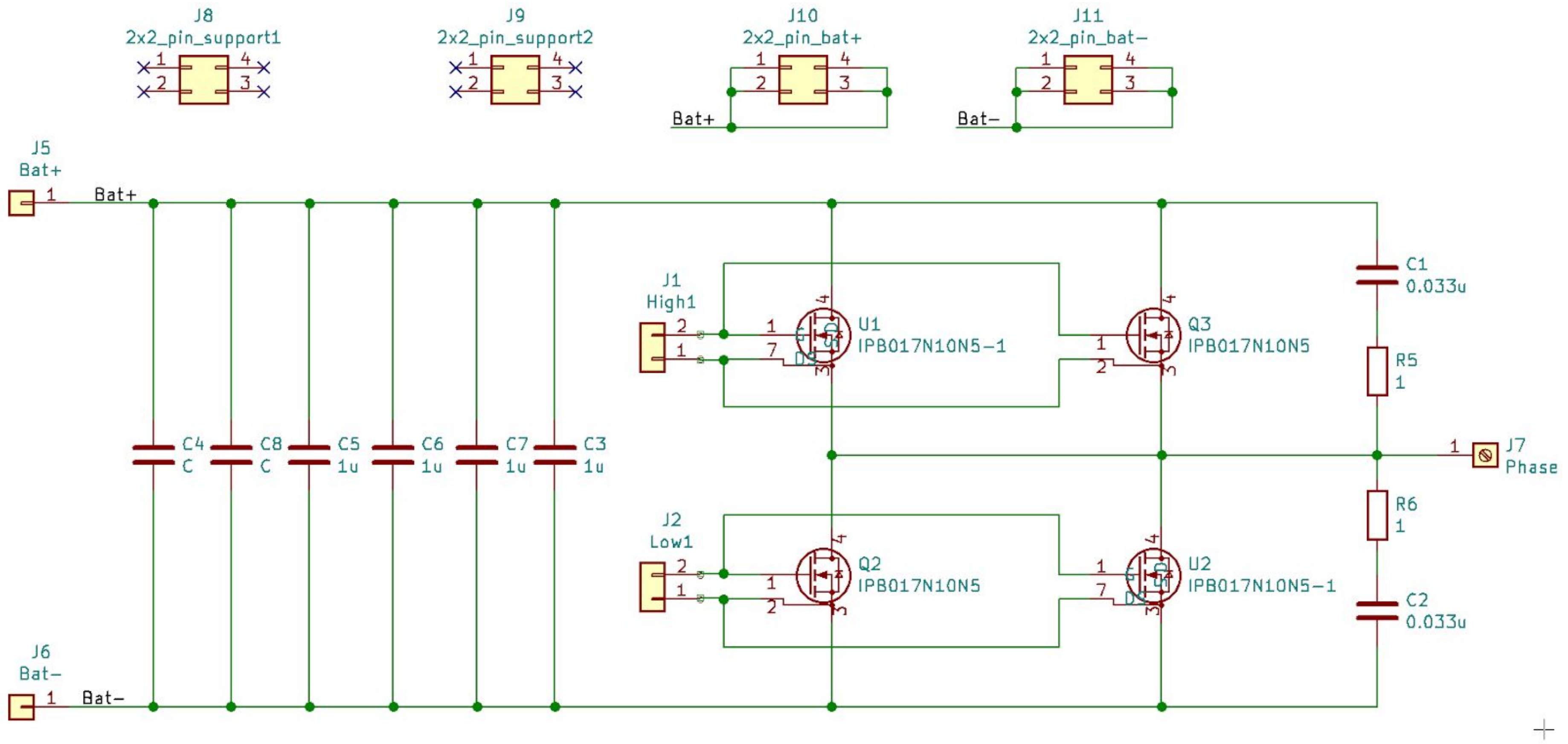
The schematics and layout of the driver board are shown in this section.

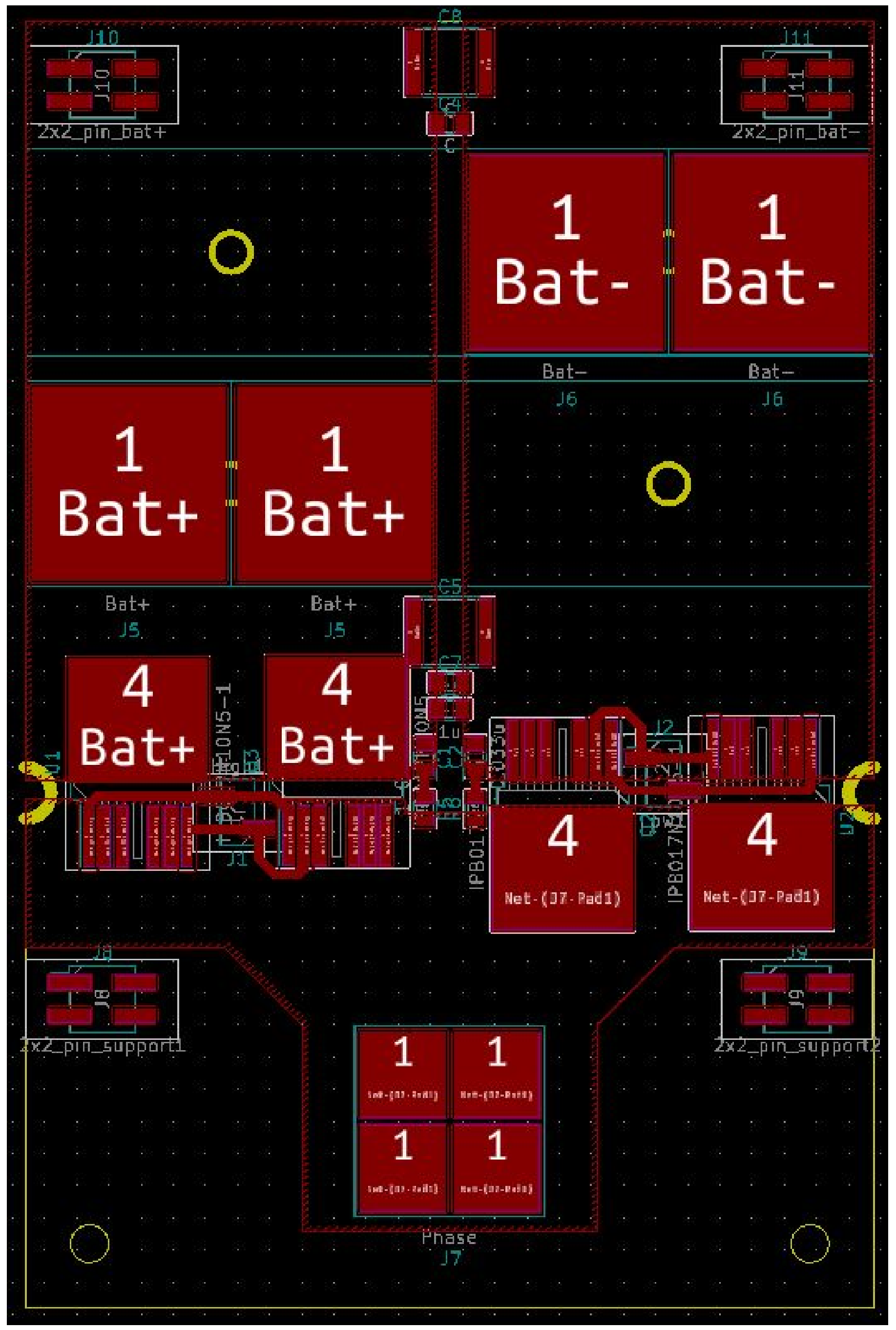




E Power board schematic and layout

The schematics and layout of the power board are shown in this section.


J8
2x2_pin_support1
J9
2x2_pin_support2
J10
2x2_pin_bat+
J11
2x2_pin_bat−
Bat+
Bat−
J5
Bat+
J6
Bat−
J1
High1
J2
Low1
U1
IPB017N10N5−1
Q3
IPB017N10N5
Q2
IPB017N10N5
U2
IPB017N10N5−1
C1
0.033u
C2
0.033u
R5
1
R6
1
C4
C
C8
C
C5
1u
C6
1u
C7
1u
C3
1u
J7
Phase


J10
2x2_pin_bat+
J11
2x2_pin_bat−
1
Bat-
Bat−
J6
1
Bat+
Bat+
J5
4
Net-(J7-Pad1)
J8
2x2_pin_support1
J9
2x2_pin_support2
Phase
J7